19th Scandinavian Symposium on Algorithm Theory

SWAT 2024, June 12–14, 2024, Helsinki, Finland

Edited by

Hans L. Bodlaender

LIPIcs

Editors

Hans L. Bodlaender
Utrecht University, The Netherlands
h.l.bodlaender@uu.nl

ACM Classification 2012
Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-318-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-318-8.

Publication date
June, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.SWAT.2024.0

ISBN 978-3-95977-318-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Regular Papers

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 19th Scandinavian Symposium on Algorithm Theorym, which takes place from June 12–14, 2024 in Helsinki, Finland. Since 1988, the Scandinavian Symposium on Algorithm Theory has been held every two years. The former name of the conference was the Scandinavian Workshop on Algorithm Theory, and the meetings with the former and current name use the acronym SWAT. SWAT alternates with sister conference WADS, the Algorithms and Data Structures Symposium. SWAT is usually hosted in a Nordic country, and WADS is usually hosted in Canada.

In total, 102 papers were submitted to SWAT 2024. The papers were assigned to at least three members of the Program Committee. Each paper received at least three reviews, by members of the Program Committee or external reviewers. The Program Committee selected 40 papers for presentation at the conference and publication in these proceedings. This selection was based on originality, quality, and relevance to the topic of the conference. The average quality of the submitted papers was very high, and even with the larger number of accepted papers in the program of SWAT 2024, many excellent papers could not be selected.

In addition to the selected papers, SWAT 2024 has three invited lectures by distinguished scientists:

- Karen Aardal (Technical University Delft, the Netherlands): Machine-learning augmented enumeration for integer optimization, and beyond.
- Greg Bodwin (University of Michigan, US): Turán-type problems in Theoretical Computer Science.
- Michał Pilipczuk (University of Warsaw, Poland): Well-structured graphs through the lens of logic.

Many thanks are due to the members of the Program Committee and all subreviewers for their hard work to evaluate and discuss the submitted papers. The members of the Program Committee were:

- Anders Aamand, MIT, US,
- Ahmad Biniaz, University of Windsor, Canada,
- Hans Bodlaender, Utrecht University, the Netherlands (chair),
- Flavia Bonomo, University of Buenos Aires, Argentina,
- Kevin Buchin, Technical University Dortmund, Germany,
- Pål Grønås Drange, University of Bergen, Norway,
- Franziska Eberle, Technische Universität Berlin, Germany,
- Klim Efremenko, Ben Gurion University, Israel,
- Gramoz Goranci, University of Vienna, Austria,
- Kazuo Iwama, National Tsing Hua University, Taiwan,
- Matthew Johnson, Durham University, UK,
- Matthew Katz, Ben Gurion University, Israel,
- Jan Kratochvíl, Charles University, Czech Republic,
- Stefan Kratsch, Humboldt-Universität zu Berlin, Germany,
- Łukasz Kowalik, University of Warsaw, Poland,
- Anil Maheshwari, Carleton University, Canada,
- George B. Mertzios, Durham University, UK,
- Yoshio Okamoto, The University of Electro-Communications, Japan,
- Yota Otachi, Nagoya University, Japan,
- Sang-il Oum, Institute for Basic Science / KAIST, Korea,

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Sharath Raghvendra, NCSU, US,
- Laura Sanita, Bocconi University of Milan, Italy,
- Subhash Suri, University of California, Santa Barbara, US,
- Ioan Todinca, University of Orleans, France,
- Jara Uitto, Aalto University, Finland,
- Tandy Warnow, University of Illinois at Urbana - Champaign, US,
- Prudence Wong, University of Liverpool, UK,
- Meirav Zehavi, Ben Gurion University, Israel.

The organization of SWAT 2024 was done by the Local Organization committee, consisting of:

- Jara Uitto, Aalto University, Finland,
- Mikko Koivisto, University of Helsinki, Finland,
- Sándor Kisfaludi-Bak, Aalto University, Finland, and
- Alexandru Paler, Aalto University, Finland.

The series of SWAT conferences is guided by the SWAT Steering Committee. The members of the Steering Committee are:

- Per Austrin, KTH, Sweden (chair),
- Fedor Fomin, University of Bergen, Norway,
- Inge Li Gørtz, Technical University of Denmark, Denmark,
- Magnús Halldórsson, Reykjavik University, Iceland,
- Jukka Suomela, Aalto University, Finland.

I want to thank all authors who submitted a manuscript to the conference, all members of the Program Committee and all subreferees for all the work in the evaluation and selection of papers, the Local Organizing Committee for their great efforts to organize the conference with all its different aspects, the SC chair Per Austrin for helpful advice and support, and the three invited speakers Karen Aardal, Greg Bodwin and Michał Pilipczuk for their willingness to give an invited lecture at SWAT 2024.

Hans Bodlaender, April 2024

List of Authors

Akanksha Agrawal (1)
Indian Institute of Technology Madras,
Chennai, India

Stav Ashur (2)
Department of Computer Science,
University of Illinois, Urbana, IL, USA

Lora Bailey (3)
Department of Mathematics, Grand Valley State
University, Allendale, MI, USA

Girish Balakrishnan (4)
Indian Institute of Technology Madras,
Chennai, India

Robert D. Barish (5)
Division of Medical Data Informatics, Human
Genome Center, Institute of Medical Science,
University of Tokyo, Japan

Cristina Bazgan (6)
Université Paris-Dauphine, PSL Research
University, CNRS, UMR 7243, LAMSADE,
Paris, France

Jesse Beisegel (7)
Institute of Mathematics, Brandenburg
University of Technology, Cottbus, Germany

Matthias Bentert (8)
University of Bergen, Norway

Ioana O. Bercea (9)
KTH Royal Institute of Technology,
Stockholm, Sweden

Magnus Berg (10)
University of Southern Denmark,
Odense, Denmark

Gaétan Berthe (11)
LIRMM, Université de Montpellier, CNRS,
Montpellier, France

Sayan Bhattacharya (12)
University of Warwick, UK

Therese Biedl (13)
David R. Cheriton School of Computer Science,
University of Waterloo, Canada

Philip Bille (14)
Technical University of Denmark,
Lyngby, Denmark

Heather Smith Blake (3)
Department of Mathematics and Computer
Science, Davidson College, NC, USA

Lotte Blank (15)
University of Bonn, Germany

Prosenjit Bose (13)
School of Computer Science, Carleton University,
Ottawa, Canada

Marin Bougeret (11)
LIRMM, Université de Montpellier, CNRS,
Montpellier, France

Joan Boyar (16)
Department of Mathematics and Computer
Science, University of Southern Denmark,
Odense, Denmark

Gerth Stølting Brodal (17)
Aarhus University, Denmark

Josh Brunner (34)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Kevin Buchin (18)
Faculty of Computer Science,
TU Dortmund University, Germany

Maike Buchin (18)
Faculty of Computer Science,
Ruhr University Bochum, Germany

Benjamin Merlin Bumpus (19)
University of Florida, Gainesville, FL, USA

Sergio Cabello (1)
Faculty of Mathematics and Physics, University
of Ljubljana, Slovenia;
Institute of Mathematics, Physics and
Mechanics, Ljubljana, Slovenia

Sankardeep Chakraborty (4)
University of Tokyo, Japan

Kenneth C. Cheung (34)
NASA Ames Research Center,
Moffett Field, CA, USA

Nina Chiarelli (7)
FAMNIT and IAM, University of Primorska,
Koper, Slovenia

Aleksander B. G. Christiansen (20)
Technical University of Denmark,
Lyngby, Denmark

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Garner Cochran (3)
Department of Mathematics and Computer Science, Berry College, Mount Berry, GA, USA

Martín Costa (12)
University of Warwick, UK

Alex Crane (8)
University of Utah, Salt Lake City, UT, USA

Seyed Parsa Darbouy (21)
Department of Computing Science, University of Alberta, Canada

Erik D. Demaine (34)
Massachusetts Institute of Technology, Cambridge, MA, USA

Jenny Diomidova (34)
Massachusetts Institute of Technology, Cambridge, MA, USA

Pål Grønås Drange (8)
University of Bergen, Norway

Anne Driemel (15)
University of Bonn, Germany

Fedor V. Fomin (22)
University of Bergen, Norway

Nathan Fox (3)
Department of Quantitative Sciences, Canisius University, Buffalo, NY, USA

Zachary Friggstad (21, 23)
Department of Computing Science, University of Alberta, Canada

Auguste H. Gezalyan (25)
Department of Computer Science, University of Maryland, College Park, MD, USA

Petr A. Golovach (22)
University of Bergen, Norway

Daniel Gonçalves (11)
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Christine Gregg (34)
NASA Ames Research Center, Moffett Field, CA, USA

Joachim Gudmundsson (18)
Faculty of Engineering, The University of Sydney, Australia

Bernd Gärtner (24)
Department of Computer Science, ETH Zürich, Switzerland

Sariel Har-Peled (2, 26)
Department of Computer Science, University of Illinois, Urbana, IL, USA

Della H. Hendrickson (34)
Massachusetts Institute of Technology, Cambridge, MA, USA

Jakob Bæk Tejs Houen (9)
BARC, University of Copenhagen, Denmark

Kien C. Huynh (27)
Linköping University, Sweden

Bart M. P. Jansen (19, 28)
Eindhoven University of Technology, The Netherlands

Matthew Johnson (29)
Durham University, UK

Naonori Kakimura (30)
Department of Mathematics, Keio University, Yokohama, Japan

Vishwas Kalani (24)
Department of Computer Science and Engineering, I.I.T. Delhi, India

Shahin Kamali (10, 16)
York University, Toronto, Canada

Michael Kaufmann (1)
Department of Computer Science, Tübingen University, Germany

Soo H. Kim (25)
Wellesley College, MA, USA

Tuukka Korhonen (22)
University of Bergen, Norway

Irina Kostitsyna (31, 34)
TU Eindhoven, The Netherlands; KBR at NASA Ames Research Center, Moffett Field, CA, USA

Ekkehard Köhler (7)
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Kim S. Larsen (16)
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

Ali Mohammad Lavasani (16)
Department of CSSE, Concordia University, Montreal, Canada

Michael Levet (3, 32)
Department of Computer Science,
College of Charleston, SC, USA

Yaqiao Li (16)
Department of CSSE, Concordia University,
Montreal, Canada

Carlos Lopez (25)
Montgomery Blair High School,
Silver Spring, MD, USA

Anand Louis (33)
Indian Institute of Science, Bangalore, India

Meghana M. Reddy (24)
Department of Computer Science,
ETH Zürich, Switzerland

Reem Mahmoud (3)
Department of Computer Science, Virginia
Commonwealth University, Richmond, VA, USA

Barnaby Martin (29)
Durham University, UK

Wouter Meulemans (24)
Department of Mathematics and Computer
Science, TU Eindhoven, The Netherlands

Martin Milanič (7)
FAMNIT and IAM, University of Primorska,
Koper, Slovenia

Babak Miraftab (13)
School of Computer Science,
Carleton University, Ottawa, Canada

Joseph S. B. Mitchell (27)
Stony Brook University, NY, USA

Daniel Mock (35)
RWTH Aachen University, Germany

David M. Mount (25)
Department of Computer Science, University of
Maryland, College Park, MD, USA

Peter Muršič (7)
FAMNIT, University of Primorska,
Koper, Slovenia

N. S. Narayanaswamy (4)
Indian Institute of Technology Madras,
Chennai, India

Ofer Neiman (36)
Ben-Gurion University of the Negev,
Beer-Sheva, Israel

Yakov Nekrich (14)
Michigan Technological University,
Houghton, MI, US

Linh Nguyen (27)
Stony Brook University, NY, USA

André Nichterlein (6)
Algorithmics and Computational Complexity,
Technische Universität Berlin, Germany

Tim Ophelders (31)
Utrecht University, The Netherlands;
TU Eindhoven, The Netherlands

Christian Ortlieb (37)
Institute of Computer Science,
University of Rostock, Germany

Yota Otachi (38)
Graduate School of Informatics,
Nagoya University, Japan

Rasmus Pagh (9)
BARC, University of Copenhagen, Denmark

Sukanya Pandey (29)
Utrecht University, The Netherlands

Denis Pankratov (16)
Department of CSSE, Concordia University,
Montreal, Canada

Nadav Panski (12)
Tel Aviv University, Israel

Irene Parada (31)
Universitat Politècnica de Catalunya,
Barcelona, Spain

Rameesh Paul (33)
Indian Institute of Science, Bangalore, India

Daniël Paulusma (29)
Durham University, UK

Tom Peters (31)
TU Eindhoven, The Netherlands

Solon P. Pissis (14)
CWI, Amsterdam, The Netherlands;
Vrije Universiteit, Amsterdam, The Netherlands

Lukas Plätz (18)
Faculty of Computer Science,
Ruhr University Bochum, Germany

Valentin Polishchuk (27)
Linköping University, Sweden

Arka Ray (33)
Indian Institute of Science, Bangalore, India

Jean-Florent Raymond (11)
Univ Lyon, EnsL, CNRS, LIP, F-69342,
Lyon Cedex 07, France

Felix Reidl (8)
Birkbeck, University of London, UK

Eliot W. Robson (26)
Department of Computer Science,
University of Illinois, Urbana, IL, USA

Benjamin Rockel-Wolff (39)
Research Institute for Discrete Mathematics,
University of Bonn, Germany

Puck Rombach (32)
Department of Mathematics and Statistics,
University of Vermont, Burlington, VT, USA

Peter Rossmanith (35)
RWTH Aachen University, Germany

Eva Rotenberg (20)
Technical University of Denmark,
Lyngby, Denmark

Kunihiko Sadakane (4)
University of Tokyo, Japan

Mohammad R. Salavatipour (40)
Department of Computer Science,
University of Alberta, Edmonton, Canada

Saket Saurabh (1, 22)
Institute of Mathematical Sciences, HBNI,
Chennai, India

Robert Scheffler (7)
Institute of Mathematics, Brandenburg
University of Technology, Cottbus, Germany

Ildikó Schlotter (30)
HUN-REN Centre for Economic and Regional
Studies, Budapest, Hungary;
Budapest University of Technology and
Economics, Hungary

Jens M. Schmidt (37)
Institute of Computer Science,
University of Rostock, Germany

Idan Shabat (36)
Ben-Gurion University of the Negev,
Beer-Sheva, Israel

Roohani Sharma (1)
University of Bergen, Norway

Tetsuo Shibuya (5)
Division of Medical Data Informatics, Human
Genome Center, Institute of Medical Science,
University of Tokyo, Japan

Nicholas Sieger (32)
Department of Mathematics, University of
California San Diego, La Jolla, CA, USA

Inne Singgih (3)
Department of Mathematical Sciences,
University of Cincinnati, OH, USA

Daniel Skora (25)
Indiana University, Bloomington, IN, USA

Siani Smith (29)
University of Bristol, UK;
Heilbronn Institute for Mathematical Research,
Bristol, UK

Shay Solomon (12)
Tel Aviv University, Israel

Willem Sonke (31)
TU Eindhoven, The Netherlands

Bettina Speckmann (24, 31)
Department of Mathematics and Computer
Science, TU Eindhoven, The Netherlands

Grace Stadnyk (3)
Department of Mathematics, Furman University,
Greenville, SC, USA

Zofia Stefankovic (25)
Stony Brook University, Stony Brook, NY, USA

Miloš Stojaković (24)
Department of Mathematics and Informatics,
Faculty of Sciences, University of Novi Sad,
Serbia

Blair D. Sullivan (8)
University of Utah, Salt Lake City, UT, USA

Hao Sun (23)
University of Alberta, Canada

Akira Suzuki (38)
Graduate School of Information Sciences,
Tohoku University, Sendai, Japan

Yuma Tamura (38)
Graduate School of Information Sciences,
Tohoku University, Sendai, Japan

MIT-NASA Space Robots Team (34)
Massachusetts Institute of Technology,
Cambridge, MA, USA;
NASA Ames Research Center,
Moffett Field, CA, USA

Lea Thiel (18)
Faculty of Computer Science,
Ruhr University Bochum, Germany

Lijiangnan Tian (40)
Department of Computer Science,
University of Alberta, Edmonton, Canada

Yushi Uno (1)
Graduate School of Informatics, Osaka
Metropolitan University, Sakai, Japan

Erik Jan van Leeuwen (29)
Utrecht University, The Netherlands

Sofia Vazquez Alferez (6)
Université Paris-Dauphine, PSL Research
University, CNRS, UMR 7243, LAMSADE,
Paris, France

Jaime Venne (19)
Eindhoven University of Technology,
The Netherlands

Ruben F. A. Verhaegh (28)
Eindhoven University of Technology,
The Netherlands

Juliette Vlieghe (20)
Technical University of Denmark,
Lyngby, Denmark

Alexander Wiedemann (3)
Department of Mathematics,
Randolph-Macon College, Ashland, VA, USA

Sebastian Wild (17)
University of Liverpool, UK

Alexander Wolff (1)
Universität Würzburg, Germany

Sampson Wong (18)
Department of Computer Science,
University of Copenhagen, Denmark

Eliminating Crossings in Ordered Graphs

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India

Sergio Cabello
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

Michael Kaufmann
Department of Computer Science, Tübingen University, Germany

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India

Roohani Sharma
University of Bergen, Norway

Yushi Uno
Graduate School of Informatics, Osaka Metropolitan University, Sakai, Japan

Alexander Wolff
Universität Würzburg, Germany

Abstract

Drawing a graph in the plane with as few crossings as possible is one of the central problems in graph drawing and computational geometry. Another option is to remove the smallest number of vertices or edges such that the remaining graph can be drawn without crossings. We study both problems in a book-embedding setting for *ordered graphs*, that is, graphs with a fixed vertex order. In this setting, the vertices lie on a straight line, called the *spine*, in the given order, and each edge must be drawn on one of several pages of a book such that every edge has at most a fixed number of crossings. In book embeddings, there is another way to reduce or avoid crossings; namely by using more pages. The minimum number of pages needed to draw an ordered graph without any crossings is its (fixed-vertex-order) *page number*.

We show that the page number of an ordered graph with n vertices and m edges can be computed in $2^m \cdot n^{\mathcal{O}(1)}$ time. An $\mathcal{O}(\log n)$-approximation of this number can be computed efficiently. We can decide in $2^{\mathcal{O}(d\sqrt{k}\log(d+k))} \cdot n^{\mathcal{O}(1)}$ time whether it suffices to delete k edges of an ordered graph to obtain a d-*planar* layout (where every edge crosses at most d other edges) on *one* page. As an additional parameter, we consider the size h of a *hitting set*, that is, a set of points on the spine such that every edge, seen as an open interval, contains at least one of the points. For $h = 1$, we can efficiently compute the minimum number of edges whose deletion yields fixed-vertex-order page number p. For $h > 1$, we give an XP algorithm with respect to $h + p$. Finally, we consider *spine+t-track drawings*, where some but not all vertices lie on the spine. The vertex order on the spine is given; we must map every vertex that does not lie on the spine to one of t tracks, each of which is a straight line on a separate page, parallel to the spine. In this setting, we can minimize in $2^n \cdot n^{\mathcal{O}(1)}$ time either the number of crossings or, if we disallow crossings, the number of tracks.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Fixed parameter tractability; Human-centered computing → Graph drawings; Mathematics of computing → Graph theory

Keywords and phrases Ordered graphs, book embedding, edge deletion, d-planar, hitting set

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.1

Related Version *Full Version*: `https://arxiv.org/abs/2404.09771`

Funding Funded in part by Science and Engineering Research Board, Startup Research Grant (SRG/2022/000962). Funded in part by the Slovenian Research and Innovation Agency (P1-0297, J1-2452, N1-0218, N1-0285). Funded in part by the EU (ERC, KARST, project no. 101071836).

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 1; pp. 1:1–1:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the EU or the ERC. Neither the EU nor the granting authority can be held responsible for them. Partially supported by JSPS KAKENHI grant no. JP17K00017, 20H05964, and 21K11757.

Acknowledgements We thank the organizers of the 2023 Dagstuhl Seminar "New Frontiers of Parameterized Complexity in Graph Drawing", where this work was initiated.

1 Introduction

Many crossings typically make it hard to understand the drawing of a graph, and thus much effort in the area of Graph Drawing has been directed towards reducing the number of crossings in drawings of graphs. In terms of parameterized complexity, several facets of this problem have been considered. For example, there are FPT algorithms that, given a graph G and an integer k, decide whether G can be drawn with at most k crossings [17, 21]. Crossing minimization has also been considered in the setting where each vertex of the given graph must lie on one of two horizontal lines. This restricted version of crossing minimization is an important subproblem in drawing layered graphs according to the so-called *Sugiyama framework* [33]. There are two variants of the problem; either the vertices on both lines may be freely permuted or the order of the vertices on one line is given. These variants are called two-layer and one-layer crossing minimization, respectively. For both, FPT algorithms exist [22, 23]. Zehavi [38] has surveyed parameterized approaches to crossing minimization.

Surprisingly, crossing minimization remains NP-hard even when restricted to graphs that have a planar subgraph with just one edge less [9]. Another way to deal with crossings is to remove a small number of vertices or edges such that the remaining graph can be drawn without crossings. In fact, it is known that vertex deletion to planarity is FPT with respect to the number of deleted vertices [18, 20, 28]. However, the running times of these algorithms depends at least exponentially on the number of deleted vertices. On the kernelization front, there exists an $\mathcal{O}(1)$-approximate kernel for vertex deletion to planarity [19], whereas vertex deletion to outerplanarity is known to admit an (exact) polynomial kernel [13].

In this paper, we focus on another model to cope with the problem of crossing edges, namely *book embeddings*, drawings where the vertices lie on a straight line, called the *spine*, and each edge must be drawn on one of several halfplanes, called *pages*, such that the drawing on each page is crossing-free (planar) or such that each edge has at most a constant number d of crossings (that is, the drawing is d-planar). We consider the variant of the problem where the order σ of the vertices is given and fixed. The minimum number of pages to draw an (ordered) graph without any crossings is its (fixed-vertex-order) *page number*.

In this paper, we study the problem of designing parameterized algorithms, where the possible parameters are the number k of edges to be deleted, the number c of allowed crossings per edge, the number p of pages, and their combinations.

Problem description. Given a graph G, let $V(G)$ denote the vertex set and $E(G)$ the edge set of G. An *ordered graph* (G, σ) consists of a graph G and an ordering of the vertices of G, that is, a bijective map $\sigma\colon V(G) \to \{1, \dots, |V(G)|\}$. Henceforth, we specify every edge (u, v) of (G, σ) such that $\sigma(u) < \sigma(v)$. For two edges $e = (u, v)$ and $e' = (u', v')$ of an ordered graph (G, σ), we say that e and e' *cross with respect to* σ if their endpoints interleave, that is, if $\sigma(u) < \sigma(u') < \sigma(v) < \sigma(v')$ or if $\sigma(u') < \sigma(u) < \sigma(v') < \sigma(v)$. The ordered graph models the scenario where the vertices of G are placed along a horizontal line in the given order σ and all the edges are drawn above the line using curves that cross as few times as possible. Whenever e and e' cross with respect to σ, their curves must intersect. Whenever

e and e' do *not* cross with respect to σ, their curves can be drawn without intersections; for example, we may use halfcircles. In this setting, we get a drawing such that two edges of G cross precisely if and only if they cross with respect to σ. Given a positive integer d, we say that an ordered graph (G, σ) is *d-planar* if every edge in G is crossed by at most d other edges (where 0-planar simply means planar).

In this paper, we focus on fast parameterized algorithms for the following problem.

Edge Deletion to p-Page d-Planar

Input: An ordered graph (G, σ) and positive integers k, p, and d.

Parameters: k, p, d

Question: Does there exist a set S of at most k edges of G such that $(G - S, \sigma)$ is p-page d-planar?

We stress that we view p and d, though they appear in the problem name, not as constants, but as parameters.

Related work. Given an ordered graph (G, σ), its *conflict graph* $H_{(G,\sigma)}$ is the graph that has a vertex for each edge of G and an edge for each pair of crossing edges of G. Note that $H_{(G,\sigma)}$ is a *circle graph*, that is, the intersection graph of chords of a circle, because two chords in a circle intersect if and only if their endpoints interleave.

We can express Edge Deletion to 1-Page d-Planar as the problem of deleting from $H_{(G,\sigma)}$ a set of at most k vertices such that the remaining graph has maximum degree at most d. For general graphs, this problem is called Vertex Deletion to Degree-d [31]; it admits a quadratic kernel [15, 37].

Testing whether (G, σ) has (fixed-vertex-order) page number p (without any edge deletions) is equivalent to the p-colorability of the conflict graph $H_{(G,\sigma)}$. For $p = 2$, it suffices to test whether the conflict graph $H_{(G,\sigma)}$ is bipartite. An alternative approach, discussed by Masuda, Nakajima, Kashiwabara, and Fujisawa [29], is to add to G a cycle connecting the vertices along the spine in the given order, and then test for planarity. Another possibility is to use 2-Sat. For $p = 4$, Unger [34] showed that the problem is NP-hard. For $p = 3$, he [35] claimed an efficient solution, but recently his approach was shown to be incomplete [4].

Edge Deletion to p-Page Planar is the special case where $d = 0$; it can be interpreted as deletion of as few vertices as possible in the conflict graph $H_{(G,\sigma)}$ to obtain a p-colorable graph. For $p = 1$, the problem can be solved by finding a maximum independent set in a circle graph, which takes linear time [16, 30, 36]; see Lemma 3 in Section 2. Edge Deletion to 2-Page Planar can be phrased as Odd Cycle Transversal in the conflict graph, which means that it is FPT with respect to the number of edges that must be deleted [32]. The case $p = 2$ can also be modeled as a (geometric) special case of Almost 2-Sat (variable), which can be solved in $2.3146^k \cdot n^{\mathcal{O}(1)}$ time, where k is the number of variables that need to be deleted so that the formula becomes satisfiable [27, Corollary 5.2].

Masuda et al. [29] showed that the problem Fixed-Order 2-Page Crossing Number is NP-hard. In this problem, we have to decide, for each edge of the given ordered graph (G, σ), whether to draw it above or below the spine, so as to minimize the number of crossings.

Bhore, Ganian, Montecchiani, and Nöllenburg [7] studied the fixed-vertex-order page number and provide an algorithm to compute it with running time $2^{\mathcal{O}(\mathrm{vc}^3)}n$, where vc is the vertex cover number of the graph. They also proved that the problem is fixed-parameter tractable parameterized by the pathwidth (pw) of the *ordered* graph, with a running time of $\mathrm{pw}^{\mathcal{O}(\mathrm{pw}^2)}\, n$. Note that the pathwidth of an ordered graph is in general not bounded by the vertex cover number [7]. This has been improved by Liu, Chen, Huang, and Wang [26]

Table 1 New and known results concerning Edge Deletion to p-Page d-Planar.

k	p	d	add. param.	ref.	result (runtime, ratio, or kernel size)	
0	min	0	–	Cor. 2	EXP:	$2^m n^{\mathcal{O}(1)}$
0	min	0	–	Thm. 4	approx:	ratio $\mathcal{O}(\log n)$
0	min	param.	–	Cor. 5	approx:	ratio $\mathcal{O}((d+1)\log n)$
param.	1	param.	–	Thm. 6	FPT:	$2^{\mathcal{O}(d\sqrt{k}\log(d+k))}\cdot n^{\mathcal{O}(1)}$
min	param.	0	–	Sect. 4	EXP:	$4^m n^{\mathcal{O}(1)}$
min	param.	0	$h=1$	Thm. 9	P:	$\mathcal{O}(m^3\log n\log\log p)$
min	param.	0	h	Thm. 12	XP:	$\mathcal{O}(m^{(4h-2)p+3}\log n\log\log p)$
0	–	min	t	Thm. 14	EXP:	$2^n n^{\mathcal{O}(1)}$
0	–	min	min t	Cor. 15	EXP:	$2^n n^{\mathcal{O}(1)}$
param.	1	param.	–	[15, 37]	kernel:	quadratic
0	≥ 4	0	–	[34]	NPC.	
0	≤ 2	0	–	folklore	P:	linear time; e.g., via 2-Sat
min	1	0	–	e.g., [16]	P:	linear time
param.	2	0	–	[32]	FPT:	Odd Cycle Transversal
0	2	min	–	[29]	NPC.	
0	min	0	vc	[25]	FPT:	$(d+2)^{\mathcal{O}(\mathrm{vc}^3)}n$
0	min	0	pw	[25]	FPT:	$(d+2)^{\mathcal{O}(\mathrm{pw}^2)}n$
0	param.	cr	pw	[26]	FPT:	$n\cdot(\mathrm{cr}+2)^{\mathcal{O}(\mathrm{pw}^2)}$
0	param.	param.	pw	[26]	FPT:	$2^{\mathcal{O}(\mathrm{pw}^2)}n$; no poly. pw-kernel

to $2^{\mathcal{O}(\mathrm{pw}^2)}n$. They also showed that the problem does not admit a polynomial kernel if parameterized only by pw (unless NP $\subseteq$ coNP/poly). Moreover, they gave an algorithm that checks in $(\mathrm{cr}+2)^{\mathcal{O}(\mathrm{pw}^2)}n$ time whether a graph with n vertices and pathwidth pw can be drawn on a given number of pages with at most cr crossings in total.

Liu, Chen and Huang [25] considered the problem Fixed-Order Book Drawing with bounded number of crossings per edge: decide if there is a p-page book-embedding of G such that the maximum number of crossings per edge is upper-bounded by an integer d. This problem was posed by Bhore et al. [7]. Liu et al. showed that this problem, when parameterized by both the maximum number d of crossings per edge and the vertex cover number vc of the graph, admits an algorithm running in $(d+2)^{\mathcal{O}(\mathrm{vc}^3)}n$ time. They also showed that the problem, when parameterized by both d and the pathwidth pw of the vertex ordering, admits an algorithm running in $(d+2)^{\mathcal{O}(\mathrm{pw}^2)}n$ time.

All these problems can be considered also in the setting where we can choose the ordering of the vertices along the spine; see, for instance, [7, 11].

Our contribution. For an overview over our results and known results, see Table 1. First, we show that the fixed-vertex-order page number of an ordered graph with m edges and n vertices can be computed in $2^m \cdot n^{\mathcal{O}(1)}$ time; see Section 2. We use subset convolution [8]. Alternatively, given a budget p of pages, we can compute a p-page book embedding with the minimimum number of crossings. By combining the greedy algorithm for Set Cover with an efficient algorithm for Maximum Independent Set in circle graphs [16, 30, 36], we obtain an efficient $\mathcal{O}((d+1)\log n)$-approximation algorithm for the fixed-vertex-order d-planar page number.

Second, we tackle Edge Deletion to 1-Page d-Planar; see Section 3. We show how to decide in $2^{\mathcal{O}(d\sqrt{k}\log(d+k))}\cdot n^{\mathcal{O}(1)}$ time whether deleting k edges of an ordered graph suffices to obtain a d-planar layout on *one* page. Note that our algorithm is subexponential in k.

Third, we consider the problem Edge Deletion to p-Page Planar; see Section 4. As an additional parameter, we consider the size h of a *hitting set*, that is, a set of points on the spine such that every edge, seen as an open interval, contains at least one of the points. For $h = 1$, we can efficiently compute the smallest set of edges whose deletion yields fixed-vertex-order page number p. For $h > 1$, we give an XP algorithm with respect to $h + p$.

Finally, we consider *spine+t-track drawings*; see Section 5. In such drawings, some but not all vertices lie on the spine. The vertex order on the spine is again given, but now we must map every vertex that does not lie on the spine to one of t tracks, each of which is a straight line on a separate page, parallel to the spine. Using subset convolution, we can minimize in $2^n \cdot n^{\mathcal{O}(1)}$ time either the number of crossings or, if we disallow crossings, the number of tracks.

We close with some open problems; see Section 6.

2 Computing the Fixed-Vertex-Order Page Number

Let (G, σ) be an ordered graph, and let p be a positive integer. In this section, we consider *p-page book-embeddings* of (G, σ): the vertices of G are placed on a *spine* ℓ according to σ, there are p *pages* (halfplanes) sharing ℓ on their boundary, and for each edge we have to decide on which page it is drawn. The aim is to minimize the total number of crossings for a given number of pages, or minimize the number of pages to attain no crossings; see Figure 1.

Let $\mathrm{cr}_p(G, \sigma)$ be the minimum number of crossings over all possible assignments of the edges of $E(G)$ to the p pages. As discussed in the introduction, we can decide in linear time whether $\mathrm{cr}_2(G, \sigma) = 0$, but in general, computing $\mathrm{cr}_2(G, \sigma)$ is NP-hard [29]. The fixed-vertex-order page number of (G, σ) is the minimum p such that $\mathrm{cr}_p(G, \sigma) = 0$.

▶ **Theorem 1.** *Given an ordered graph (G, σ) with n vertices and m edges, and a positive integer p, we can compute the values $\mathrm{cr}_1(G, \sigma), \ldots, \mathrm{cr}_p(G, \sigma)$ in $2^m \cdot n^{\mathcal{O}(1)}$ time. In particular, given a budget p of pages, we can compute a p-page book embedding with the minimum number of crossings within the given time bound.*

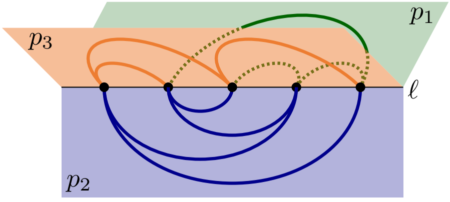
Proof. Consider a fixed-vertex-order graph $((V, E), \sigma)$ with n vertices and m edges. We need to consider only the case $p < m$ because, for $p \geq m$, it obviously holds that $\mathrm{cr}_p((V, E), \sigma) = 0$.

First note that, for any fixed $F \subseteq E$, we can easily compute $\mathrm{cr}_1((V, F), \sigma)$ in $\mathcal{O}(|F|^2) = \mathcal{O}(m^2)$ time by checking the order of the endpoints of each pair of edges. It follows that we can compute $\mathrm{cr}_1((V, F), \sigma)$ for all subsets $F \subseteq E$ in $2^m \cdot n^{\mathcal{O}(1)}$ time.

For every $q > 1$ and every $F \subseteq E$, we have the recurrence

$$\mathrm{cr}_q((V, F), \sigma) \;=\; \min \{\mathrm{cr}_1((V, F'), \sigma) + \mathrm{cr}_{q-1}((V, F \setminus F'), \sigma) \mid F' \subseteq F\}\,. \quad (1)$$

Here, $F' \subseteq F$ corresponds to the edges that in the drawing go to one page, and thus $F \setminus F'$ goes to the remaining $q - 1$ pages, where we can optimize over all choices of $F' \subseteq F$.



■ **Figure 1** A 3-page book embedding of K_5 with fixed vertex order. For each edge, we can choose on which page it is drawn. Note that K_5 cannot be drawn on two pages without crossings.

From the recurrence in Equation (1) we see that, for $q > 1$, the function $F \mapsto \mathrm{cr}_q((V,F),\sigma)$ is, by definition, the *subset convolution* of the functions $F \mapsto \mathrm{cr}_1((V,F),\sigma)$ and $F \mapsto \mathrm{cr}_{q-1}((V,F),\sigma)$ in the (min, +) ring. Since $\mathrm{cr}_q((V,F),\sigma)$ takes integer values from $\{0,\dots,m^2\}$ for every q and F, it follows from [8] that one can obtain $\mathrm{cr}_q((V,F),\sigma)$ for all $F \subseteq E$ in $2^m \cdot n^{\mathcal{O}(1)}$ time, for a fixed $q > 1$, assuming that $\mathrm{cr}_1((V,F),\sigma)$ and $\mathrm{cr}_{q-1}((V,F),\sigma)$ are already available. Therefore, we can compute the values $\mathrm{cr}_q((V,F),\sigma)$ for $q \in \{2,\dots,p\}$ in $2^m \cdot n^{\mathcal{O}(1)}$ time since $p \leq m < n^2$. ◀

▶ **Corollary 2.** *The fixed-vertex-order page number of a graph with n vertices and m edges can be computed in $2^m \cdot n^{\mathcal{O}(1)}$ time.*

▶ **Lemma 3.** *Given an ordered graph (G,σ), we can compute in polynomial time a smallest subset $S \subseteq E(G)$ such that $\mathrm{cr}_1(G-S,\sigma) = 0$.*

Proof. Consider the *conflict graph $H_{(G,\sigma)}$* of (G,σ), already defined in the introduction. Note that $H_{(G,\sigma)}$ is a circle graph. Therefore, a largest independent set in $H_{(G,\sigma)}$ corresponds to a largest subset F of edges with $\mathrm{cr}_1((V,F),\sigma) = 0$, which corresponds to a minimum set $S \subseteq E(G)$ such that $\mathrm{cr}_1(G-S,\sigma) = 0$. Finally, note that a largest independent set in circle graphs can be computed in polynomial time [16, 30, 36]. ◀

▶ **Theorem 4.** *We can compute an $\mathcal{O}(\log n)$-approximation to the fixed-vertex-order page number of a graph with n vertices in polynomial time.*

Proof. Let $((V,E),\sigma)$ be the given ordered graph, and let OPT be its fixed-vertex-order page number. Define the family $\mathcal{F} = \{F \subseteq E \mid \mathrm{cr}_1((V,F),\sigma) = 0\}$. Consider the Set Cover instance $(E,\mathcal{F})$, where E is the universe and $\mathcal{F} \subseteq 2^E$ is a family of subsets of E. A feasible solution of this Set Cover instance is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that $\bigcup \mathcal{F}' = E$. The task in Set Cover is to find a feasible solution of minimum cardinality.

Each feasible solution $\mathcal{F}'$ to the Set Cover instance $(E,\mathcal{F})$ corresponds to a fixed-vertex-order drawing of (G,σ) with $|\mathcal{F}'|$ pages. Similarly, each fixed-vertex-order drawing of (G,σ) with p pages represents a feasible solution to Set Cover with p sets. In particular, the size of the optimal solution to the Set Cover instance $(E,\mathcal{F})$ is equal to OPT, the fixed-vertex-order page number of (G,σ).

Consider the usual greedy algorithm for Set Cover, which works as follows. Set $E_1 = E$ and $i = 1$. While $E_i \neq \emptyset$, we set F_i to be the element of $\mathcal{F}$ that contains the largest number of edges from E_i, increase i, and set $E_i = E_{i-1} \setminus F_{i-1}$. Let $i^\star$ be the maximum value of i with $E_i \neq \emptyset$. Thus $E_{i^\star+1} = \emptyset$, and the algorithm finishes. It is well known that $i^\star \leq \mathrm{OPT} \cdot \log |E|$; see for example [12, Section 5.4]. Therefore, this greedy algorithm yields an $\mathcal{O}(\log |V|)$-approximation for our problem.

Finally, note that the greedy algorithm can be implemented to run efficiently. Indeed, F_i can be computed from E_i in polynomial time because of Lemma 3, and the remaining computations in every iteration are trivially done in polynomial time. The number of iterations is polynomial because $i^\star \leq |E|$. ◀

▶ **Corollary 5.** *We can compute an $\mathcal{O}((d+1)\log n)$-approximation to the fixed-vertex-order d-planar page number of a graph with n vertices in polynomial time.*

Proof. Consider first an ordered graph (H,σ) that is d-planar if drawn on a single page, with $d > 0$. Let F_d be the subset of $E(H)$ such that each edge in F_c participates in exactly d crossings, and let S_d be a maximal subset of F_d such that no two edges in S_d cross each other. Then, $(H - S_d, \sigma)$ is $(d-1)$-planar because each edge of H has fewer than d crossings, is in S_c, or is crossed by some edge in S_d. It follows by induction that (H,σ) can be embedded in $d+1$ pages without crossings.

Consider now the input ordered graph (G, σ) and let OPT_d be the minimum number of d-planar pages needed for (G, σ). By the argument before applied to each page, we know that the minimum number of planar pages, OPT_0, is at most $(d+1)\,\mathrm{OPT}_d$. Using Theorem 4, we obtain a drawing of (G, σ) without crossings with at most $\mathrm{OPT}_0 \cdot \mathcal{O}(\log n) \le (d+1)\,\mathrm{OPT}_d \cdot \mathcal{O}(\log n)$ (planar) pages, where $n = |V(G)|$. Such a drawing is of course also d-planar. ◀

3 Edge Deletion to 1-Page d-Planar

The main result of this section is as follows.

▶ **Theorem 6.** *Edge Deletion to 1-Page d-Planar admits an algorithm with running time $2^{\mathcal{O}(d\sqrt{k}\log(d+k))} \cdot n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph and k is the number of edges to be deleted.*

In other words, we obtain a subexponential fixed-parameter tractable algorithm for Edge Deletion to 1-Page d-Planar parameterized by k, the number of edges to be deleted; note that we consider d to be a constant here (although we made explicit how the running time depends on d). Our algorithm to prove Theorem 6 has two steps. First it branches on edges that are crossed by at least $d + \sqrt{k}$ other edges. When such edges do not exist, we show that the conflict graph $H_{(G,\sigma)}$ has treewidth $\mathcal{O}(d + \sqrt{k})$. This is done by showing that the conflict graph has balanced separators. Finally the bound on the treewidth allows us to use a known (folklore) algorithm [24] for Vertex Deletion to Degree-d whose dependency is singly exponential in the treewidth of $H_{(G,\sigma)}$.

3.1 Branching

Let $\texttt{cross}_{(G,\sigma)}(e)$ denote the set of edges of G that cross e with respect to σ. We drop the subscript (G, σ) when it is clear from the context. We show that we can use branching to reduce any instance to a collection of instances where each edge e of the graph satisfies $|\texttt{cross}(e)| < d + \sqrt{k}$. In particular we show the following lemma.

▶ **Lemma 7.** *Let (G, σ, k) be an instance of Edge Deletion to 1-Page d-Planar. There is a $2^{\mathcal{O}(d\sqrt{k}\log(d+k))} \cdot n^{\mathcal{O}(1)}$-time algorithm that outputs $2^{\mathcal{O}(d\sqrt{k}\log(d+k))}$ many instances of Edge Deletion to 1-Page d-Planar $(G_1, \sigma, k_1), \ldots, (G_r, \sigma, k_r)$ such that for each $i \in [r]$, G_i is a $(d + \sqrt{k})$-planar graph, and (G, σ, k) is a Yes-instance of Edge Deletion to 1-Page d-Planar if and only if (G_i, σ, k_i) is a Yes-instance of Edge Deletion to 1-Page d-Planar for some $i \in [r]$.*

Proof. Let e be an edge of G with $|\texttt{cross}(e)| \ge d + \lceil\sqrt{k}\rceil$. If $|\texttt{cross}(e)| > d + k$, then e must be deleted, as we cannot afford to keep e and delete enough edges from $\texttt{cross}(e)$. If $|\texttt{cross}(e)| \le d + k$, then either e must be deleted or at least $|\texttt{cross}(e)| - d$ many edges from $\texttt{cross}(e)$ must be deleted, so that at most d edges of $\texttt{cross}(e)$ stay. This results in the following branching rule, where we return an OR over the answers of the following instances:

1. Recursively solve the instance $(G - e, \sigma, k - 1)$. This branch is called the *light* branch.
2. If $|\texttt{cross}(e)| > d + k$, we do not consider other branches. Otherwise, for each subset X of $\texttt{cross}(e)$ with $|\texttt{cross}(e)| - d$ many edges, recursively solve the instance $(G - X, \sigma, k - |X|)$. Each of these branches is called a *heavy* branch.

We are going to show that the recursion tree has $2^{\mathcal{O}(d\sqrt{k}\log(d+k))}$ branches. Note that the number of possible heavy branches at each is node is

$$\binom{|\texttt{cross}(e)|}{|\texttt{cross}(e)| - d} = \binom{|\texttt{cross}(e)|}{d} \le \binom{d+k}{d} \le (d+k)^d.$$

To prove the desired upper bound, we interpret the branching tree as follows. First note that, in each node, we have at most $(d+k)^d$ heavy branches. We associate a distinct word over the alphabet $\Sigma = \{0, 1, \ldots, (d+k)^d\}$ to each leaf (or equivalently each root to leaf path) of the recurrence tree. For each node of the recurrence tree, associate a character from Σ with each of its children such that the child node corresponding to the light branch gets the character 0 and the other nodes (corresponding to the heavy branches) get a distinct character from $\Sigma \setminus \{0\}$. Now a word over the alphabet Σ for a leaf ℓ of the recurrence tree is obtained by taking the sequence of characters on the nodes of the root to leaf ℓ path in order. In order to bound the number of leaves (and hence the total number of nodes) of the recurrence tree, it is enough to bound the number of such words. The character 0 is called a *light* label and all other characters are called *heavy* labels. Recall that a light label corresponds to the branch where k drops by 1, while the heavy labels correspond to the branches where k drops by $|\texttt{cross}(e)| - d \geq \sqrt{k}$. This implies that each word (that is associated with the leaf of the recurrence tree) has at most $\sqrt{k}$ heavy labels. In order to bound the number of such words, we first guess the places in the word that are occupied by heavy labels and then we guess the (heavy) labels themselves at these selected places. All other positions have the light label on them and there is no choice left. Hence, the number of such words is upper-bounded by

$$\sum_{i=0}^{\sqrt{k}} \binom{k}{i} ((d+k)^d)^i \leq \sqrt{k} \binom{k}{\sqrt{k}} ((d+k)^d)^{\sqrt{k}} = 2^{\mathcal{O}(d\sqrt{k}\log(d+k))}.$$

This shows that the number of such words is bounded by $2^{\mathcal{O}(d\sqrt{k}\log(d+k))}$, and hence the number of leaves (and nodes) of the recurrence tree is bounded by $2^{\mathcal{O}(d\sqrt{k}\log(d+k))}$. ◀

3.2 Balanced Separators in the Conflict Graph

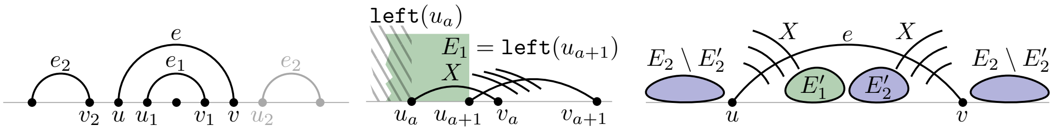
Let (G, σ) be an ordered graph. For any edge $e = (u, v)$ of G, let $\texttt{span}_{(G,\sigma)}(e)$ be the set of all edges $(u', v') \neq e$ of G such that $\sigma(u) \leq \sigma(u') \leq \sigma(v') \leq \sigma(v)$. For example, in Figure 2a, $\texttt{span}(e) = \{e_1\}$. For any vertex w of G, let $\texttt{left}_{(G,\sigma)}(w)$ be the set of all edges (u, v) of G such that $\sigma(u) < \sigma(v) \leq \sigma(w)$. Whenever it is clear from the context, we will drop the subscript (G, σ). We say that an edge e of G is *maximal* if G contains no edge e' such that $e \in \texttt{span}(e')$.

▶ **Lemma 8** (Balanced Separator in the Conflict Graph). *If (G, σ) is an ordered d-planar graph, then G contains a set X of at most $3(d+1)$ edges such that $E(G) \setminus X = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, $|E_1| \leq 2m/3$, $|E_2| \leq 2m/3$, and no edge $e_1 \in E_1$ crosses an edge $e_2 \in E_2$ with respect to σ.*

The proof in the full version of this paper [1] considers three cases depending on the spans of the edges of G; see Figure 2. Either there exists an edge $e = (u, v) \in E(G)$ such that $m/3 \leq |\texttt{span}(e)| \leq 2m/3$ (case 1), or for every edge $e \in E(G)$, it holds that $|\texttt{span}(e)| \leq m/3$ (case 2), or there exists an edge $e \in E(G)$ such that $|\texttt{span}(e)| > 2m/3$ (case 3). Note that every outer d-planar graph contains a balanced *vertex* separater of size at most $2d + 3$ [10].

3.3 Proof of Theorem 6

We now need to establish a relation between the treewidth of the graph and the size of a balanced separator in it. For this we use the result of Dvořák and Norin [14] that shows a linear dependence between the treewidth and the *separation number* of a graph: the separation number of a graph is the smallest integer s such that every subgraph of the given



(a) Case 1. **(b)** Case 2. **(c)** Case 3.

Figure 2 Case distinction for the proof of Lemma 8.

graph has a balanced separator of size at most s. A balanced separator in a graph H is a set of vertices B such that the vertex set of $H - B$ can be partitioned into two parts V_1 and V_2 such that $E(V_1, V_2) = \emptyset$ and $|V_1|, |V_2| \leq 2|V(H)|/3$. In other words, they show that if the separation number of the graph is s, then the treewidth of such a graph is $\mathcal{O}(s)$.

Recall that (G, σ, k) is an instance of Edge Deletion to 1-Page d-Planar. By Lemma 8, if the ordered graph (G, σ) is $(d + \sqrt{k})$-planar, then the conflict graph $H_{(G,\sigma)}$ has a balanced separator of size at most $3(d + \sqrt{k} + 1)$. Thus, due to the result of Dvořák and Norin [14], the treewidth of $H_{(G,\sigma)}$ is $\mathcal{O}(d + \sqrt{k})$.

Given a graph with N vertices and treewidth tw, one can compute, in $(d + 2)^{\text{tw}} \cdot N^{\mathcal{O}(1)}$ time, the smallest set of vertices whose deletion results in a graph of degree at most d [24]. Applying this result to the conflict graph $H_{(G,\sigma)}$, which has at most $|V(G)|^2 = n^2$ vertices and treewidth $\mathcal{O}(d + \sqrt{k})$, we conclude that Edge Deletion to 1-Page d-Planar can be solved in $2^{\mathcal{O}((d+\sqrt{k}) \log d)} \cdot n^{\mathcal{O}(1)}$ time if the given ordered graph (G, σ) is $(d + \sqrt{k})$-planar.

From Lemma 7, we can assume, at the expense of a multiplicative factor of $2^{\mathcal{O}(d\sqrt{k} \log(k+d))} \cdot n^{\mathcal{O}(1)}$ on the running time, that the given ordered graphs (G, σ) to consider are $(d + \sqrt{k})$-planar. Thus, given (G, σ, k), we can solve Edge Deletion to 1-Page d-Planar in $2^{\mathcal{O}(d\sqrt{k} \log(d+k))} \cdot n^{\mathcal{O}(1)}$ time. This concludes the proof of Theorem 6.

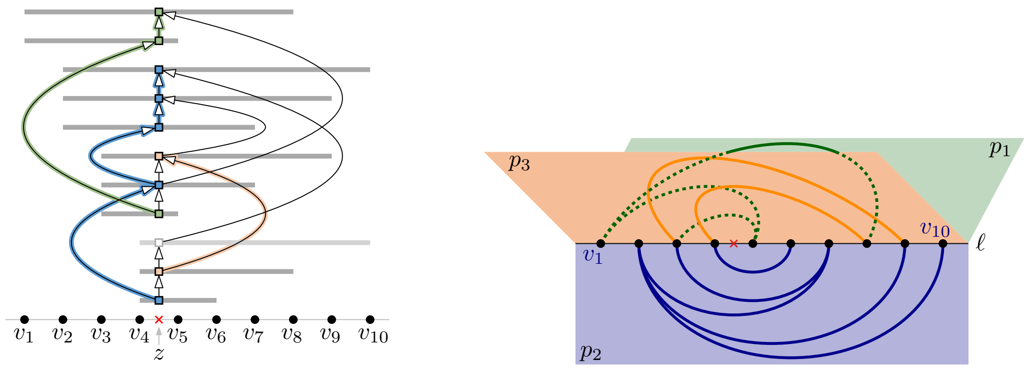
4 Edge Deletion to p-Page Planar

In this section we treat the problem Edge Deletion to p-Page Planar, which is the special case of Edge Deletion to p-Page d-Planar for $d = 0$. It can be solved by brute force in $\mathcal{O}((p + 1)^m \cdot n^2)$ time: For each mapping of the m edges to the p pages, with the "+1" to mark edge deletion, check for each pair of edges assigned to the same page whether they intersect. It can also be solved in $4^m \cdot n^{\mathcal{O}(1)}$ time: for each of the 2^m subsets of $E(G)$, use Corollary 2 to decide whether its fixed-vertex-order page number is at most p.

We now consider a new parameter in addition to p. The edge set of an ordered graph (G, σ) corresponds to a set of open intervals on the real line; namely every edge (u, v) of G is mapped to the interval $(\sigma(u), \sigma(v))$. Given a set $\mathcal{I}$ of intervals, a *hitting set* for $\mathcal{I}$ is a set of points on the real line such that each interval contains at least one of the points. Note that a hitting set can be much smaller than a vertex cover: an ordered graph (G, σ) with a hitting set of size 1 can have linear vertex cover number (e.g., $G = K_{n,n}$). Given a set $\mathcal{I}$ of m open intervals, a minimum-size hitting set for $\mathcal{I}$ can be found in $\mathcal{O}(m \log m)$ time by a simple greedy algorithm.

For two edges $(u, v), (u', v')$ of (G, σ), we say that (u, v) *contains* (u', v') if the interval $(\sigma(u), \sigma(v))$ contains the interval $(\sigma(u'), \sigma(v'))$. If (u, v) and (u', v') cross with respect to σ, then there is no containment, otherwise one contains the other.

Hitting set of size 1. We start by treating the following special case of Edge Deletion to p-Page Planar. Given an ordered graph (G, σ), a point z on the real line that is contained in every interval defined by $E(G)$, a number p of pages, and a threshold $k \geq 0$, we



(a) Intervals corrsponding to the edges of G; auxiliary graph G^+ without transitive edges.

(b) Optimal solution for (a): only the edge (v_4, v_{10}) is deleted; the pages correspond to the colored paths in (a).

Figure 3 Instance with hitting set of size 1 and optimal solution for three pages.

want to decide whether there is a set $E' \subseteq E(G)$ of size at most k such that that $G - E'$ can be drawn without crossings on p pages (respecting vertex order σ). Note that if there is a hitting set of size 1, then G is necessarily bipartite and that $z \notin \sigma(V(G))$. We show that Edge Deletion to p-Page Planar can be solved efficiently if $h(G, \sigma) = 1$.

Alam et al. [3] have called this setting *separated*; they showed that the *mixed* page number of an ordered $K_{n,n}$ is $\lceil 2n/3 \rceil$ in this case. While we study the (usual) page number of an ordered graph where each page corresponds to a stack layout, the mixed page number asks for the smallest number of stacks and queues (where nested edges are not allowed on the same page) needed to draw an ordered graph.

▶ **Theorem 9.** *Given an ordered graph (G, σ) with n vertices, m edges, and $h(G, \sigma) = 1$, Edge Deletion to p-Page Planar can be solved in $\mathcal{O}(m^3 \log n \log \log p)$ time.*

Proof. From (G, σ) we construct an acyclic directed auxiliary graph G^+, from which we then construct an s–t flow network $\mathcal{N}$ such that an integral maximum s–t flow of minimum cost in $\mathcal{N}$ corresponds to p vertex-disjoint directed paths in G^+ of maximum total length, and each path in G^+ corresponds to a set of edges in G that can be drawn without crossings on a single page in a book embedding of (G, σ). The set E' of edges that need to be deleted from G such that $G - E'$ has page number p corresponds to the vertices of G^+ that do not lie on any of the p paths.

We now describe these steps in detail. The auxiliary graph G^+ has a node for each edge of G and an arc from edge node (a, b) to edge node (a', b') if in (G, σ) the edge (a', b') contains the edge (a, b) (meaning that the edges do not cross); see Figure 3. Hence G^+ has exactly m nodes and at most $\binom{m}{2}$ edges, and can be constructed from (G, σ) in $\mathcal{O}(m^2)$ time.

The s–t flow network $\mathcal{N}$ is defined as follows. For each node v of G^+, introduce two vertices v_{in} and v_{out} in $\mathcal{N}$, connected by the arc $(v_{\text{in}}, v_{\text{out}})$ of capacity 1 and cost -1. All other arcs in $\mathcal{N}$ have cost 0. For each arc (u, v) of G^+, add the arc $(u_{\text{out}}, v_{\text{in}})$ of capacity 1 to $\mathcal{N}$. Then add to $\mathcal{N}$ new vertices s, s', and t, the edge (s, s') of capacity p, and the edges $\{(s', v_{\text{in}}), (v_{\text{out}}, t) \colon v \in V(G^+)\}$ of capacity 1. Summing up, $\mathcal{N}$ has $2m + 3$ vertices, at most $\binom{m}{2} + 3m + 1$ edges, and can be constructed from G^+ in $\mathcal{O}(m^2)$ time.

Due to the edge (s, s'), a maximum flow in $\mathcal{N}$ has value at most p. If $m \geq p$ (otherwise the instance is trivial, and no edge has to be deleted), then a maximum flow has value exactly p. Since all edge capacities and costs are integral, the minimum-cost circulation algorithm of Ahuja, Goldberg, Orlin, and Tarjan [2] yields an integral flow. Since all edges (except for (s, s')) have edge capacity 1 and $\mathcal{N}$ is acyclic, the edges (except for (s, s')) with non-zero flow form p paths of flow 1 from s' to t that are vertex-disjoint except for their endpoints. These paths (without s' and t) correspond to vertex-disjoint paths in G^+. Due to the negative cost of the edges of type $(v_{\mathrm{in}}, v_{\mathrm{out}})$, the flow maximizes the number of such edges with flow. This maximizes the number of vertices in G^+ that lie on one of the p paths. This, in turn, maximizes the number of edges of G that can be drawn without crossings on p pages in a book embedding of (G, σ). Given a flow network with n' vertices, m' edges, maximum capacity U, and maximum absolute cost value C, the algorithm of Ahuja et al. runs in $\mathcal{O}(n'm'(\log\log U)\log(n'C))$ time. In our case, $n' \in \mathcal{O}(m)$, $m' \in \mathcal{O}(m^2)$, $U = p$, and $C = 1$. Hence computing the maximum flow of minimum cost in $\mathcal{N}$ takes $\mathcal{O}(m^3 \log n \log\log p)$ time. This dominates the time needed to construct G^+ and $\mathcal{N}$. ◀

In our forthcoming algorithm, we will use an extension of this result, as follows. Two subsets $E', E'' \subset E(G)$ are *compatible* if $|E'| = |E''|$ and there is an enumeration $e'_1, \dots, e'_{|E'|}$ of E' and an enumeration $e''_1, \dots, e''_{|E'|}$ of E'' such that e'_i is contained in e''_i for all $i \in [|E'|]$. Note that we may have $E' \cap E'' \neq \emptyset$.

▶ **Lemma 10.** *Given an ordered graph (G, σ) with n vertices, m edges, $h(G, \sigma) = 1$, and subsets $E', E'' \subset E(G)$ with $p = |E'| = |E''|$, we can decide, in $\mathcal{O}(m^3 \log n \log\log p)$ time, whether E' and E'' are compatible and, if yes, solve a version of* Edge Deletion to p-Page Planar *where, on each page, one edge of E' is contained in all other edges and one edge of E'' contains all other edges on that page.*

Proof. We adapt the proof of Theorem 9 by modifying the flow network $\mathcal{N}$ that is considered. More precisely, we insert arcs from s' only to the edges $e' \in E'$, and we insert arcs to t only from the edgs $e'' \in E''$. No other arcs go out from s' nor go into t.

Note that E' and E'' are compatible if and only if the value of the maximum flow in the modified flow network is exactly p. ◀

Our technique, based on flows, does not allow us to enforce a pairing of the edges in E' and in E''. With other words, we cannot select edges $e'_1, e'_2 \in E'$ and $e''_1, e''_2 \in E''$, and insist that e'_1 and e''_1 go to one page, and e'_2 and e''_2 go to another page. This difficulty will play an important role in our forthcoming extension.

An XP algorithm for the general case. Let H be a finite hitting set of (G, σ). We assume, without loss of generality, that $H \cap \sigma(V(G)) = \emptyset$. Given a subset $X \subseteq H$, we say that an edge (u, v) of G with $\sigma(u) < \sigma(v)$ *bridges* X if $\sigma(u) < \min X$, $\max X < \sigma(v)$, and X is the largest subset of H with this property. For each $X \subseteq H$, let E_X be the subset of edges of (G, σ) that bridge X. For example, in Figure 4, $|H| = 3$, and the edges in E_H lie in the outer gray region.

Consider any drawing of a subgraph of (G, σ) with edge set $\tilde{E}$ on p pages without crossings. For each page $q \in [p]$, let $\tilde{E}^q$ be the set of edges in $\tilde{E}$ that are on page q, and let $\mathcal{X}^q$ be the family of subsets of H bridged by some edge of $\tilde{E}^q$. Since there are no crossings on page q, the sets of $\mathcal{X}^q$ form a so-called *laminar* family: any two sets in $\mathcal{X}^q$ are either disjoint or one contains the other. For each $X \in \mathcal{X}^q$, let e^q_X be the smallest edge of $\tilde{E}^q$ that bridges X, and let f^q_X be the largest edge of $\tilde{E}^q$ that bridges X; it may be that $e^q_X = f^q_X$. Note that for each $X, Y \in \mathcal{X}^q$ with $X \subsetneq Y$, the edge e^q_Y contains f^q_X. We say that the *partial encoding* of $\tilde{E}$ on page q is $\mathcal{E}^q = \{(X, e^q_X, f^q_X) \mid X \in \mathcal{X}^q\}$ and the *encoding* of $\tilde{E}$ is $\langle \mathcal{E}^1, \dots, \mathcal{E}^p \rangle$.

When a set X is bridged on only one page of an optimal drawing, say $X \in \mathcal{X}^1$, then we just have to select as many edges as possible without crossing from those contained between

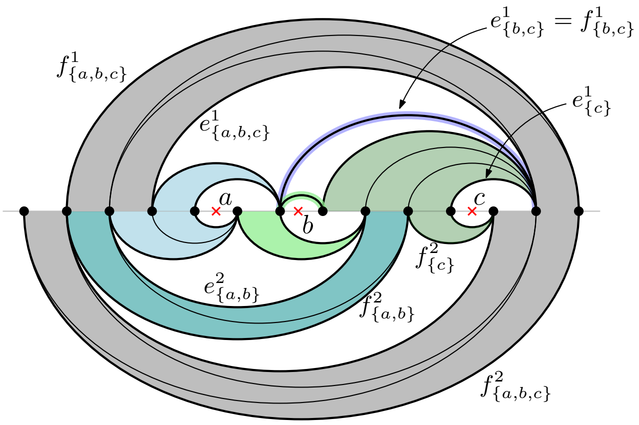


Figure 4 Encoding $\langle \mathcal{E}^1, \mathcal{E}^2 \rangle$ of a 2-page drawing for an instance with hitting set $H = \{a, b, c\}$ (red crosses). For each $X \subseteq H$ and page $q \in [2]$, the edges e^q_X and f^q_X (if they exist) are thicker than the other edges. Each colored region corresponds to a set of edges that bridge the same subset of H.

e^1_X and f^1_X, because the edges of E_X cannot appear in any other page. The challenge that we face is the following: when the same set X appears in $\mathcal{X}^q$ for different $q \in [p]$, the choices of which edges are drawn in each of those pages are not independent. However, we can treat all such pages together, exchanging some parts of the drawings from one page to another, as follows. For each $X \subseteq H$, let $Q_X = \{q \in [p] \colon X \in \mathcal{X}^q\}$ be the set of pages where some edges bridge X.

▶ **Lemma 11.** *Consider $\tilde{E} \subseteq E(G)$ that can be drawn in p pages without crossings, and let $\langle \mathcal{E}^1, \dots, \mathcal{E}^p \rangle$ be the corresponding encoding. For every $X \subseteq H$ with $Q_X \neq \emptyset$, let $\tilde{E}'_X = \{e^q_X \mid q \in Q_X\}$, let $\tilde{E}''_X = \{f^q_X \mid q \in Q_X\}$, and let F_X be the set of edges in E_X obtained when using Lemma 10 for $p' = |Q_X|$ pages with boundary edges $\tilde{E}'_X$ and $\tilde{E}''_X$. Then the ordered subgraph with edge set $\bigcup_X F_X$ can be drawn on p pages without crossings and contains at least as many edges as $\tilde{E}$.*

Proof. Consider a fixed $X \subseteq H$ with $Q_X \neq \emptyset$. For each $q \in Q_X$, let F^q_X be the set of edges in F_X that appear on the same page as $e^q_X \in \tilde{E}'_X$ when using the algorithm of Lemma 10. Since each element of $\tilde{E}''_X$ is on a different page, let $\sigma \colon Q_X \to Q_X$ be the permutation such that $f^{\sigma(q)}_X$ is the unique element of $\tilde{E}''_X$ in F^q_X.

We make a drawing of $\hat{E} := (\tilde{E} \setminus E_X) \cup F_X$ on p pages by assigning edges to pages, as follows. For each $q \in [p] \setminus Q_X$, we just set $\hat{E}^q = \tilde{E}^q$. For each $q \in Q_X$, let $\hat{E}^q$ be obtained from $\tilde{E}^{\sigma(q)}$ by removing the edges contained in $f^{\sigma(q)}_X$, adding the edges of F^q_X, and adding the edges of $\tilde{E}^q$ contained in e^q_X. For an example, see Figure 5. For each q, the edges of $\hat{E}^q$ can be drawn on a single page without crossings. This is obvious for $q \in [p] \setminus Q_X$. For $q \in Q_X$, this is true because e^q_X and $f^{\sigma(q)}_X$ act as shields between F^q_X and the other two groups of edges, one containing $f^{\sigma(q)}_X$ and the other contained in e^q_X.

Since $\tilde{E} \cap E_X = \left(\bigcup_{q \in Q_X} \tilde{E}^q\right) \cap E_X$ is a feasible solution for the problem solved in Lemma 10, we have $|\tilde{E} \cap E_X| \leq |F_X|$. Therefore $\hat{E} = (\tilde{E} \setminus E_X) \cup F_X$ is at least as large as $\tilde{E}$.

Summarizing: for a fixed X, we have converted $\tilde{E}$ into another set of edges $\hat{E}$ that is no smaller and can be drawn without crossings on p pages such that $F_X = \hat{E} \cap E_X$ and such that no edge outside E_X is changed (that is, $\tilde{E} \setminus E_X = \hat{E} \setminus E_X$). In general, the encoding $\langle \mathcal{E}^1, \dots, \mathcal{E}^p \rangle$ changes, but the sets $\tilde{E}'_X, \tilde{E}''_X$ remain unchanged for every set X. We now iterate this process for each $X \subseteq H$. The last set $\hat{E}$ that we obtain is $\bigcup_X F_X$ because every edge of $\tilde{E}$ is in E_X for some $X \subseteq H$. The result follows. ◀

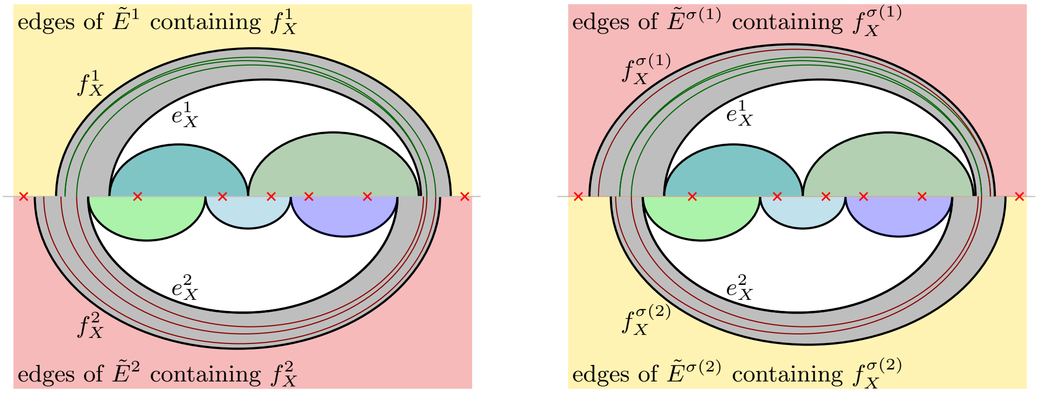

edges of $\tilde{E}^1$ containing f^1_X
f^1_X
e^1_X
e^2_X
f^2_X
edges of $\tilde{E}^2$ containing f^2_X
edges of $\tilde{E}^{\sigma(1)}$ containing $f^{\sigma(1)}_X$
$f^{\sigma(1)}_X$
e^1_X
e^2_X
$f^{\sigma(2)}_X$
edges of $\tilde{E}^{\sigma(2)}$ containing $f^{\sigma(2)}_X$

Figure 5 Left: A 2-page drawing of $\tilde{E}$. The gray region corresponds to the set $\tilde{E}_X = \tilde{E}^1_X \cup \tilde{E}^2_X$ when X is the set of the inner five red crosses. Right: drawing of a set $\hat{E} = (\tilde{E} \setminus E_X) \cup F_X$ where $\sigma(1) = 2$ and $\sigma(2) = 1$. Note that $\tilde{E}_X$ and $\hat{E}_X$ can be different; namely if $F_X \neq \tilde{E}^1_X \cup \tilde{E}^2_X$.

We now argue that, on a single page $q \in [p]$, the number of possible partial encodings $\mathcal{E}^q$ is at most m^{4h-2}. First note that $\mathcal{X}^q$ contains at most $2h-1$ sets: at most h sets in $\mathcal{X}^q$ are inclusionwise minimal, and any non-minimal element $X \in \mathcal{X}^q$ is obtained by joining two others. This means that $\mathcal{E}^q$ is characterized by selecting at most $4h-2$ edges e^q_X and f^q_X, and such a selection already determines implicitly the sets $\mathcal{X}^q$. When considering all pages together, there are at most $m^{(4h-2)\cdot p}$ encodings $\langle \mathcal{E}^1, \dots, \mathcal{E}^p \rangle$, and, for each $X \in \bigcup_{q\in[p]} \mathcal{X}^q$, we have to apply the algorithm of Lemma 10, which takes $\mathcal{O}(|E_X|^3 \log n \log\log p)$ time. Since the edge sets E_X are pairwise disjoint for different $X \subseteq H$, for each encoding we spend $\mathcal{O}(m^3 \log n \log\log p)$ time. Finally, we return the best among all encodings that give rise to a valid drawing without crossings. Since the encoding of an optimal solution will be considered at least once, Lemma 11 implies that we find an optimal solution. Therefore, the total running time is $\mathcal{O}(m^{(4h-2)\cdot p+3} \log n \log\log p)$. We summarize our result.

▶ **Theorem 12.** *Edge Deletion to p-Page Planar is in XP with respect to $h + p$.*

5 Multiple-Track Crossing Minimization

Let $G = (A \cup B, E)$ be a bipartite graph where all edges connect a vertex of A to a vertex of B and $A \cap B = \emptyset$. We further have a given linear order σ_A for the vertices of A. For the vertices of B we do not have any additional information or constraints. In this section we consider *spine+t-track drawings* of G, defined as follows:

- the vertices of A are placed on a line ℓ_0, called *spine*, in the order determined by σ_A;
- the vertices of B are placed on t different lines $\ell_1, \dots, \ell_t$ parallel to the spine; each line ℓ_q is placed on a different *page* (half-plane) π_q of a book;
- all pages $\pi_1, \dots, \pi_t$ have ℓ_0 as a common boundary and are otherwise pairwise disjoint;
- for each $q \in [t]$, the edges with endpoints in ℓ_0 and ℓ_q are drawn as straight-lines edges in the page π_q.

One can interpret this as a drawing in three dimension, as shown in Figure 6. Note that because the graph is bipartite and each edge has a vertex in A and a vertex in B, there are no edges connecting two vertices in the spine, and in particular there are no "nested" edges.

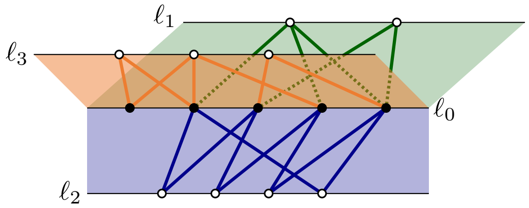
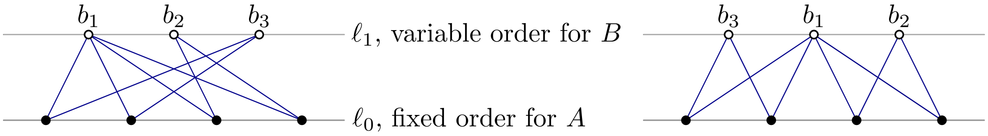


Figure 6 A spine+3-track drawing. In this example, B_1 has two vertices, B_2 has four vertices and B_3 has three vertices. The drawing has $2+5+2=9$ crossings.



b_1
b_2
b_3
ℓ_1, variable order for B
ℓ_0, fixed order for A
b_3
b_1
b_2

Figure 7 Two different orders σ_B give different number of crossings in the spine+1-track drawing: 10 on the left and 2 on the right.

To describe the drawing combinatorially, it suffices to partition B into sets $B_1, \dots, B_t$, one per line, and we have to decide for each B_q the order σ_{B_q} of the vertices B_q along ℓ_q. The number of crossings of the drawing is the sum of the number of crossings within each page, where the number of crossings within a page is the number of pairs of edges that cross each other. The value $\mathrm{cr}_t((A,\sigma_A),B,E)$ is the minimum number of crossings over all spine+t-track drawings, and the purpose of this section is to discuss its computation.

We start discussing spine+1-track drawings and its corresponding value $\mathrm{cr}_1((A,\sigma_A),B,E)$. See Figure 7 for examples of drawings. This is the minimum number of crossings in a two-layer drawing with the order on one layer, A in this case, fixed. We want to choose the order σ_B that minimizes the number of crossings. Let $\mathrm{cr}_1((A,\sigma_A),(B,\sigma_B),E)$ be the crossing number for a fixed order σ_B. Then $\mathrm{cr}_1((A,\sigma_A),B,E)$ is the minimum of $\mathrm{cr}_1((A,\sigma_A),(B,\sigma_B),E)$ when we optimize over all orders σ_B of B. The obvious approach is to try all different possible orders σ_B of B, compute $\mathrm{cr}_1((A,\sigma_A),(B,\sigma_B),E)$ for each of them, and take the minimum. This yields an algorithm with time complexity $2^{\mathcal{O}(n\log n)}$. We improve over this trivial algorithm as follows.

▶ **Theorem 13.** *We can compute* $\mathrm{cr}_1((A,\sigma_A),B,E)$ *in* $\mathcal{O}(2^n n)$ *time, where* $n=|A|+|B|$.

Proof. Construct a complete, directed, edge-weighted graph H as follows:

- $V(H)=B$
- put all directed edges in H;
- the directed edge (x,y) of H gets weight $c_{x,y}=\mathrm{cr}_1((A,\sigma_A),(\{x,y\},\sigma_{x,y}),E)$, where $\sigma_{x,y}$ is the order for $\{x,y\}$ that places x before y.

An ordering of B corresponds to a Hamiltonian path in H. Consider any Hamiltonian path in H defined by an order σ_B. Since each crossing happens between two edges incident to different vertices of B, we have

$$\mathrm{cr}_1((A,\sigma_A),(B,\sigma_B),E) = \sum_{\substack{x,y\in B\\ \sigma_B(x)<\sigma_B(y)}} c_{x,y} = \sum_{x\in B}\ \sum_{\substack{y\in B\\ \sigma_B(x)<\sigma_B(y)}} c_{x,y}. \tag{2}$$

With this interpretation, the task is to find in H a Hamiltonian path such that the sum of the $c_{\cdot,\cdot}$-weights from each vertex to all its successors is minimized. This problem is amenable to dynamic programming across subsets of vertices, as it is done for the Traveling Salesperson Problem; see [5] or [12, Section 6.6].

We define a table C by setting, for each $X \subseteq B$,

$$C[X] \;=\; \mathrm{cr}_1((A, \sigma_A), X, \{(a, x) \in E \mid x \in X, a \in A\}).$$

Then $C[X]$ is the number of crossings when we remove the vertices $B \setminus X$ from H. We are interested in computing $C[B]$ because $C[B] = \mathrm{cr}_1((A, \sigma_A), B, E)$.

We obviously have $C[X] = 0$ for each $X \subseteq B$ with $|X| \le 1$. Whenever $|X| > 1$, we use (2) and the definition of $C[X]$ to obtain the recurrence

$$C[X] \;=\; \min_{y \in X} \left(C[X \setminus \{y\}] + \sum_{x \in X \setminus \{y\}} c_{x,y} \right). \tag{3}$$

The proof of this is a standard proof in dynamic programming, where y represents the last vertex of X in the ordering.

Each value $c_{x,y}$ can be computed in $\mathcal{O}(\deg_G(x) + \deg_G(y)) = \mathcal{O}(n)$ time, which means that, over all pairs (x, y), we spend $\mathcal{O}(n^3)$ time. Each value $\eta[X, y] := \sum_{x \in X \setminus \{y\}} c_{x,y}$, defined for $X \subseteq B$ and $y \in X$, can be computed for increasing values of $|X|$ in constant time per value by noting that

$$\text{for every } X \subseteq B \text{ and distinct } y, z \in X\text{:} \quad \sum_{x \in X \setminus \{y\}} c_{x,y} = c_{z,y} + \sum_{x \in X \setminus \{y,z\}} c_{x,y}.$$

Therefore, we compute the value $\eta[X, y]$ for every $X \subseteq B$ and $y \in X$ in $\Theta\left(\sum_{k=0}^{n} \binom{n}{k} k\right) = \Theta(2^n n)$ total time. (The direct computation using the sums anew for each value would take $\Theta\left(\sum_{k=0}^{n} \binom{n}{k} k^2\right) = \Theta(2^n n^2)$, which is strictly larger.)

After this we can compute $C[X]$ for increasing values of $|X|$ using the recurrence of Equation (3), which means that we spend $\mathcal{O}(|X|)$ time for each X. This step also takes $\mathcal{O}(2^n n)$ time for all X. Finally we return $C[B]$. An optimal solution can be recovered using standard book-keeping techniques. ◀

Now we consider the case of arbitrary track number t.

▶ **Theorem 14.** *We can compute $\mathrm{cr}_t((A, \sigma_A), B, E)$ in $2^n n^{\mathcal{O}(1)}$ time for every $t > 1$, where $n = |A| + |B|$. For $t = 1$ and $t = 2$, the value can be computed in $O(2^n n)$ time.*

Proof. Once we fix a set B_q for the qth page, we can optimize the order σ_{B_q} independently of all other decisions. Therefore, we want to compute

$$\min \sum_{q=1}^{t} \mathrm{cr}_1((A, \sigma_A), B_q, E_q),$$

where E_q is the set of edges connecting vertices from A to B_q, and where the minimum is only over all the partitions $B_1, \ldots, B_t$ of B.

As we did in the proof of Theorem 13, for each subset $X \subseteq B$, we define

$$C[X] \;=\; \mathrm{cr}_1((A, \sigma_A), X, \{(a, x) \in E \mid x \in X, a \in A\}).$$

In the proof of Theorem 13 we argued that the values $C[X]$ can be computed in $\mathcal{O}(2^n n)$ time for all $X \subseteq B$ simultaneously.

We have to compute now

$$\min\left\{\sum_{q=1}^{t} C[B_q]\colon B_1,\ldots,B_t \text{ is a partition of } B\right\}.$$

The case of $t=1$ has been covered in Theorem 13. For $t=2$, we have to compute

$$\min\left\{C[B_1]+C[B\setminus B_1] \mid B_1\subseteq B\right\},$$

which can be done in $O(2^n)$ additional time iterating over all subsets B_1 of B.

For $t>2$, we use the algorithm of Björklund et al. [8] for subset convolution, as follows. Define for each $X\subseteq B$ and for $q\in[t]$ the "entry table"

$$\begin{aligned} T[X,q] &= \operatorname{cr}_q((A,\sigma_A),X,\{(a,x)\in E \mid x\in X, a\in A\}) \\ &= \min\left\{C[B_1]+\ldots+C[B_q] \mid B_1,\ldots,B_q \text{ is a partition of } X\right\}. \end{aligned}$$

We obviously have $T[X,1]=C[X]$ for all X. For $q>1$, we have the recursive relation

$$T[X,q] \;=\; \min\left\{T[Y,q-1]+C[X\setminus Y] \mid Y\subseteq X\right\}.$$

Therefore, for $q>1$, the function $X\mapsto T[X,q]$ is, by definition, the subset convolution of the functions $X\mapsto T[X,q-1]$ and $X\mapsto C[X]$ in the $(\min,+)$ ring. These functions take integer values on $\{0,\ldots,n^4\}$ because n^4 is an upper bound for $\operatorname{cr}_q((A,\sigma_A),B,E)$ for any q. It follows from [8] that one can obtain $T[X,q]$ for all $X\subseteq B$ in $2^n n^{\mathcal{O}(1)}$ time, assuming that $T[\cdot,q-1]$ and $C[\cdot]$ are already available. We compute the entries $T[\cdot,q]$ for $q=2,\ldots,t$, which adds a multiplicative $t\le n$ to the final running time. ◀

Using the theorem for increasing values of t, we obtain the following.

▶ **Corollary 15.** *We can compute the smallest value t such that $\operatorname{cr}_t((A,\sigma_A),B,E)=0$ in $2^n\cdot n^{\mathcal{O}(1)}$ time, where $n=|A|+|B|$.*

6 Open Problems

1. Could we use the concept of the conflict graph for other crossing reduction problems?
2. Is Edge Deletion to 1-Page d-Planar $W[1]$-hard with respect to the natural parameter k if d is part of the input? Can we reduce from Independent Set? Note that Vertex Deletion to Degree-d is $W[1]$-hard with respect to treewidth [6] and that outer-d planar graphs have treewidth $\mathcal{O}(d)$ [10] (which also follows from Lemma 8).
3. What if the vertex order is not given? In other words, what is the parameterized complexity of edge deletion to outer-d planarity?
4. What about exact algorithms for computing the crossing number of an ordered graph? As Masuda et al. [29] showed, the problem is NP-hard for two pages. In their NP-hardness reduction, they use a large number of crossings, and it is easy to get an algorithm that is exponential in the number of edges; see Theorem 1. Can we get a running time of $2^n\cdot n^{\mathcal{O}(1)}$ or perhaps even subexponential in n? Recall that the algorithm of Liu et al. [26] checks in $n\cdot(\operatorname{cr}+2)^{\mathcal{O}(\mathrm{pw}^2)}$ time whether a graph with pathwidth pw can be drawn on a given number of pages with at most cr crossings in total.

References

1 Akanksha Agrawal, Sergio Cabello, Michael Kaufmann, Saket Saurabh, Roohani Sharma, Yushi Uno, and Alexander Wolff. Eliminating crossings in ordered graphs. arXiv report, 2024. `arXiv:2404.09771`.

2 Ravindra K. Ahuja, Andrew V. Goldberg, James B. Orlin, and Robert E. Tarjan. Finding minimum-cost flows by double scaling. *Math. Progr.*, 53:243–266, 1992. `doi:10.1007/BF01585705`.

3 Jawaherul Md. Alam, Michael A. Bekos, Martin Gronemann, Michael Kaufmann, and Sergey Pupyrev. The mixed page number of graphs. *Theoret. Comput. Sci.*, 931:131–141, 2022. `doi:10.1016/j.tcs.2022.07.036`.

4 Patricia Bachmann, Ignaz Rutter, and Peter Stumpf. On the 3-coloring of circle graphs. In Michael Bekos and Markus Chimani, editors, *Proc. Int. Symp. Graph Drawing & Network Vis. (GD)*, volume 14465 of *LNCS*, pages 152–160. Springer, 2023. `doi:10.1007/978-3-031-49272-3_11`.

5 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962. `doi:10.1145/321105.321111`.

6 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Appl. Math.*, 160(1):53–60, 2012. `doi:j.dam.2011.08.013`.

7 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. *J. Graph Algorithms Appl.*, 24(4):603–620, 2020. `doi:10.7155/jgaa.00526`.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proc. 39th Ann. ACM Symp. Theory Comput. (STOC)*, pages 67–74, 2007. `doi:10.1145/1250790.1250801`.

9 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. `doi:10.1137/120872310`.

10 Steven Chaplick, Myroslav Kryven, Giuseppe Liotta, Andre Löffler, and Alexander Wolff. Beyond outerplanarity. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. 25th Int. Symp. Graph Drawing & Network Vis. (GD)*, volume 10692 of *LNCS*, pages 546–559. Springer, 2018. `doi:10.1007/978-3-319-73915-1_42`.

11 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM J. Algebr. Discrete Meth.*, 8(1):33–58, 1987. `doi:10.1137/0608002`.

12 Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. *Algorithms*. McGraw-Hill, 2008.

13 Huib Donkers, Bart M. P. Jansen, and Michał Włodarczyk. Preprocessing for outerplanar vertex deletion: An elementary kernel of quartic size. *Algorithmica*, 84(11):3407–3458, 2022. `doi:10.1007/s00453-022-00984-2`.

14 Zdeněk Dvořák and Sergey Norin. Treewidth of graphs with balanced separations. *J. Comb. Theory, Ser. B*, 137:137–144, 2019. `doi:10.1016/j.jctb.2018.12.007`.

15 Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of Nemhauser and Trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 77(6):1141–1158, 2011. `doi:10.1016/j.jcss.2010.12.001`.

16 Fanica Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3(3):261–273, 1973. `doi:10.1002/net.3230030305`.

17 Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. `doi:10.1016/j.jcss.2003.07.008`.

18 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proc. Ann. ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 1802–1811, 2014. `doi:10.1137/1.9781611973402.130`.

19 Bart M. P. Jansen and Michał Włodarczyk. Lossy planarization: a constant-factor approximate kernelization for planar vertex deletion. In Stefano Leonardi and Anupam Gupta, editors, *Proc. 54th Ann. ACM Symp. Theory Comput. (STOC)*, pages 900–913, 2022. doi:10.1145/3519935.3520021.

20 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proc. Ann. IEEE Symp. Foundat. Comput. Sci. (FOCS)*, pages 639–648, 2009. doi:10.1109/FOCS.2009.45.

21 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In David S. Johnson and Uriel Feige, editors, *Proc. 39th Ann. ACM Symp. Theory Comput. (STOC)*, pages 382–390, 2007. doi:10.1145/1250790.1250848.

22 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72:778–790, 2015. doi:10.1007/s00453-014-9872-x.

23 Yasuaki Kobayashi and Hisao Tamaki. A faster fixed parameter algorithm for two-layer crossing minimization. *Inform. Process. Lett.*, 116(9):547–549, 2016. doi:j.ipl.2016.04.012.

24 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. *CoRR*, abs/2304.14724, 2023. doi:10.48550/arXiv.2304.14724.

25 Yunlong Liu, Jie Chen, and Jingui Huang. Parameterized algorithms for fixed-order book drawing with bounded number of crossings per edge. In Weili Wu and Zhongnan Zhang, editors, *Proc. 14th Int. Conf. Combin. Optim. Appl. (COCOA)*, volume 12577 of *LNCS*, pages 562–576. Springer, 2020. doi:10.1007/978-3-030-64843-5_38.

26 Yunlong Liu, Jie Chen, Jingui Huang, and Jianxin Wang. On parameterized algorithms for fixed-order book thickness with respect to the pathwidth of the vertex ordering. *Theor. Comput. Sci.*, 873:16–24, 2021. doi:10.1016/j.tcs.2021.04.021.

27 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

28 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

29 Sumio Masuda, Kazuo Nakajima, Toshinobu Kashiwabara, and Toshio Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Trans. Computers*, 39(1):124–127, 1990. doi:10.1109/12.46286.

30 Nicholas Nash and David Gregg. An output sensitive algorithm for computing a maximum independent set of a circle graph. *Inf. Process. Lett.*, 110(16):630–634, 2010. doi:10.1016/j.ipl.2010.05.016.

31 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discret. Appl. Math.*, 152(1-3):229–245, 2005. doi:10.1016/j.dam.2005.02.029.

32 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/J.ORL.2003.10.009.

33 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybernetics*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

34 Walter Unger. On the k-colouring of circle-graphs. In Robert Cori and Martin Wirsing, editors, *Proc. 5th Ann. Symp. Theoret. Aspects Comput. Sci. (STACS)*, volume 294 of *LNCS*, pages 61–72. Springer, 1988. doi:10.1007/BFb0035832.

35 Walter Unger. The complexity of colouring circle graphs. In Alain Finkel and Matthias Jantzen, editors, *Proc. 9th Ann. Symp. Theoret. Aspects Comput. Sci. (STACS)*, volume 577 of *LNCS*, pages 389–400. Springer, 1992. doi:10.1007/3-540-55210-3_199.

36 Gabriel Valiente. A new simple algorithm for the maximum-weight independent set problem on circle graphs. In Toshihide Ibaraki, Naoki Katoh, and Hirotaka Ono, editors, *Proc. Int. Symp. Algorithms Comput. (ISAAC)*, volume 2906 of *LNCS*, pages 129–137. Springer, 2003. doi:10.1007/978-3-540-24587-2_15.

37 Mingyu Xiao. On a generalization of Nemhauser and Trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 84:97–106, 2017. doi:10.1016/j.jcss.2016.08.003.

38 Meirav Zehavi. Parameterized analysis and crossing minimization problems. *Comput. Sci. Rev.*, 45:100490, 2022. doi:10.1016/j.cosrev.2022.100490.

Local Spanners Revisited

Stav Ashur
Department of Computer Science, University of Illinois, Urbana, IL, USA

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA

Abstract

For a set $P \subseteq \mathbb{R}^2$ of points and a family $\mathcal{F}$ of regions, a *local t-spanner* of P is a sparse graph G over P, such that for any region $\mathscr{r} \in \mathcal{F}$ the subgraph restricted to $\mathscr{r}$, denoted by $G \cap \mathscr{r}$, is a t-spanner for all the points of $\mathscr{r} \cap P$.

We present algorithms for the construction of local spanners with respect to several families of regions such as homothets of a convex region. Unfortunately, the number of edges in the resulting graph depends logarithmically on the spread of the input point set. We prove that this dependency cannot be removed, thus settling an open problem raised by Abam and Borouny. We also show improved constructions (with no dependency on the spread) of local spanners for fat triangles, and regular k-gons. In particular, this improves over the known construction for axis-parallel squares.

We also study notions of weaker local spanners where one is allowed to shrink the region a "bit". Surprisingly, we show a near linear-size construction of a weak spanner for axis-parallel rectangles, where the shrinkage is *multiplicative*. Any spanner is a weak local spanner if the shrinking is proportional to the diameter of the region.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geometric graphs, Fault-tolerant spanners

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.2

Related Version *Full Version*: `https://arxiv.org/abs/2201.01715`

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF award CCF-1907400.

Acknowledgements The authors thanks Mohammad Abam for providing us with an early version of [2]. The authors also thank the reviewers for the numerous detailed comments.

1 Introduction

For a set P of points in $\mathbb{R}^d$, the *Euclidean graph* $\mathcal{K}_P = \left(P, \binom{P}{2}\right)$ of P is an undirected graph. Here, an edge $pq \in \binom{P}{2}$ is associated with the segment pq, and its weight is the (Euclidean) length of the segment. Let $G = (P, E)$ and $H = (P, E')$ be two graphs over the same set of vertices (usually H is a subgraph of G). Consider two vertices $p, q \in P$, and parameter $t \geq 1$. A path π between p and q in H is a *t-path* if the length of π in H is at most $t \cdot \mathsf{d}_G(p, q)$, where $\mathsf{d}_G(p, q)$ is the length of the shortest path between p and q in G. The graph H is a *t-spanner* of G if there is a t-path in H for every $p, q \in P$. Thus, for a set $P \subseteq \mathbb{R}^d$ of points, a graph G over P is a *t-spanner* if it is a t-spanner of the Euclidean graph $\mathcal{K}_P$. There is a lot of work on building geometric spanners, see [10] and references there in.

Fault-tolerant spanners

An *$\mathcal{F}$-fault-tolerant spanner* for $P \subseteq \mathbb{R}^d$ is a graph $G = (P, E)$ such that for any region $\mathscr{r} \in \mathcal{F}$ (i.e., the "attack"), the graph $G - \mathscr{r}$ is a t-spanner of $\mathcal{K}_P - \mathscr{r}$, where $G - \mathscr{r}$ denotes the graph after one deletes from G all the vertices in $P \cap \mathscr{r}$, and all the edges in G whose corresponding segments intersect $\mathscr{r}$ (See Definition 1 for a formal definition of this notation).

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 2; pp. 2:1–2:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Surprisingly, as shown by Abam et al. [3], such fault-tolerant spanners can be constructed where the attack region is any convex set. Furthermore, these spanners have a near linear number of edges.

Fault-tolerant spanners were first studied with vertex and edge faults, meaning that some arbitrary set of at most k of vertices and edges has failed. Levcopoulos et al. [8] showed the existence of k-vertex/edge fault tolerant spanners for a set P of points in some metric space. Their spanner had $\mathcal{O}(kn \log n)$ edges, and weight, i.e. sum of edge weights, bounded by $f(k) \cdot wt(MST(P))$, where $wt(MST(P))$ is the weight of $MST(P)$, for some function f. Lukovszki [9] later achieved a similar construction, improving the number of edges to $\mathcal{O}(kn)$, and was able to prove that the result is asymptotically tight.

Local spanners

Recently, Abam and Borouny [2] introduced the notion of local spanners, which can be interpreted as having the complement property to being fault-tolerant. For a family $\mathcal{F}$ of regions, a graph $G = (P, E)$ is an *$\mathcal{F}$-local t-spanner* for P if for any $\mathscr{r} \in \mathcal{F}$, the subgraph of G induced on $P \cap \mathscr{r}$ is a t-spanner. Specifically, this induced subgraph $G \cap \mathscr{r}$ contains a t-path between any $p, q \in P \cap \mathscr{r}$ (note that we keep an edge in the subgraph only if both its endpoints are in $\mathscr{r}$, see Definition 1).

Abam and Borouny [2] showed how to construct such spanners for axis-parallel squares and vertical slabs. In this work, we further extend their results. They also showed how to construct such spanners for disks if one is allowed to add Steiner points, but left the question of how to construct local spanners for disks (without Steiner points) as an open problem.

To appreciate the difficulty in constructing local spanners, observe that unlike regular spanners, the construction has to take into account many different scenarios as far as which points are available to be used in the spanner. As a concrete example, a local spanner for axis-parallel rectangles requires a quadratic number of edges, see Figure 1.1.

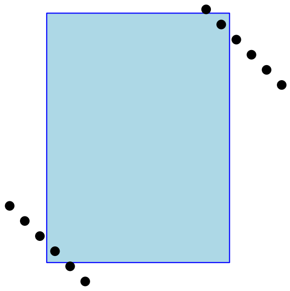


Figure 1.1 For any point in the top diagonal and bottom diagonal, there is a fat axis-parallel rectangle that contains only these two points. Thus, a local spanner requires a quadratic number of edges in this case.

Namely, regular spanners can rely on using midpoints in their path under the assurance that they are always there. For local spanners this is significantly harder as natural midpoints might “disappear”. Intuitively, a local spanner construction needs to use midpoints that are guaranteed to be present judging only from the source and destination points of the path.

A good jump is hard to find

Most constructions for spanners can be viewed as searching for a way to build a path from the source to the destination by finding a “good” jump, either by finding a way to move locally from the source to a nearby point in the right direction, as done in the θ-graph

Table 1.1 Known and new results. The notation $\mathcal{O}_\varepsilon$ hides polynomial dependency on ε which is not specified in the original work.

Region	# edges	Paper	New # edges	Location in paper
Local $(1+\varepsilon)$-spanners				
Halfplanes	$\mathcal{O}(\varepsilon^{-2} n \log n)$	[3]		
Axis-parallel squares	$\mathcal{O}_\varepsilon(n \log^6 n)$	[2]	$\mathcal{O}\big(\varepsilon^{-3} n \log n\big)$	Remark 27
Vertical slabs	$\mathcal{O}(\varepsilon^{-2} n \log n)$	[2]		
Disks+Steiner points	$\mathcal{O}_\varepsilon(n \log^2 n)$	[2]		
Disks			$\mathcal{O}\big(\varepsilon^{-2} n \log \Phi\big)$	Theorem 19
			$\Omega\big(n \log(1 + \frac{\Phi}{n})\big)$	Lemma 20
Homothets of a convex body			$\mathcal{O}\big(\varepsilon^{-2} n \log \Phi\big)$	Theorem 19
Homothets of α-fat triangles			$\mathcal{O}\big((\alpha\varepsilon)^{-1} n\big)$	Theorem 23
Homothets of triangles			$\Omega\big(n \log(1 + \frac{\Phi}{n})\big)$	Lemma 21
δ-weak local $(1+\varepsilon)$-spanners				
Convex body			$\mathcal{O}\big((\varepsilon^{-1} + \delta^{-2}) n\big)$	Lemma 12
$(1-\delta)$-local $(1+\varepsilon)$-spanners				
Axis-parallel rectangles			$\mathcal{O}\big((\varepsilon^{-2} + \delta^{-2}) n \log^2 n\big)$	Theorem 31

construction, or alternatively, by finding an edge in the spanner from the neighborhood of the source to the neighborhood of the destination, as done in the spanner constructions using a well-separated pair decomposition (WSPD). Usually, one argues inductively that the spanner must have (sufficiently short) paths from the source to the start of the jump, and from the end of the jump to the destination, and then, combining these implies that the resulting new path is short. These ideas guide our constructions as well. However, the availability of specific edges depends on the query region, making the search for a good jump significantly more challenging. Intuitively, the constructions have to guarantee that there are many edges available, and that at least one of them is useful as a jump regardless of the chosen region (since slight perturbation in the region might make many of these edges unavailable).

Our results

Almost local spanners

We start by showing that regular geometric spanners are local spanners if one is required to provide the spanner guarantee only to shrunken regions. Namely, if G is a $(1+\varepsilon)$-spanner of P, then for any convex region $\mathcal{C}$, the graph $G \cap \mathcal{C}$ is a spanner for $\mathcal{C}' \cap P$, where $\mathcal{C}'$ is the set of all points in $\mathcal{C}$ that are in distance at least $\delta \cdot \mathrm{diam}(\mathcal{C})$ from its boundary, for $\delta = \Omega(\sqrt{\varepsilon})$ – see Lemma 12.

Homothets

A *homothet* of a convex region $\mathcal{C}$ is a translated and scaled copy of $\mathcal{C}$. In Section 3 we present a construction of spanners which surprisingly is not only fault-tolerant for all smooth convex regions, but is also a local spanner for homothets of a prespecified convex region. This in particular works for disks, and resolves the aforementioned open problem of Abam and Borouny [2]. Our construction is somewhat similar to the original construction of Abam et al. [3]. For a parameter $\varepsilon > 0$ the construction of a local $(1+\varepsilon)$-spanner for homothets takes $\mathcal{O}\bigl(\varepsilon^{-2} n \log \Phi \log n\bigr)$ time, and the resulting spanner is of size $\mathcal{O}\bigl(\varepsilon^{-2} n \log \Phi\bigr)$, where Φ is the spread of the input point set P, and $n = |P|$.

The dependency on the spread Φ in the above construction is somewhat disappointing. However, the lower bound constructions, provided in Section 3.3, show that this is unavoidable for disks or homothets of triangles.

Thus, the natural question is what are the cases where one can avoid the "curse of the spread" – that is, cases where one can construct local spanners of near-linear-size independent of the spread of the input point set.

The basic building block: $\mathcal{C}$-Delaunay triangulation

A key ingredient in the above construction is the concept of Delaunay triangulations induced by homothets of a convex body. Intuitively, one replaces the unit disk (of the standard L_2-norm) by the provided convex region. It is well known [6] that such diagrams exist, have linear complexity in the plane, and can be computed quickly. In Section 3.1 we review these results, and restate the well-known property that the $\mathcal{C}$-Delaunay triangulation is connected when restricted to a homothet of $\mathcal{C}$. By computing these triangulations for carefully chosen subsets of the input point set, we get the results stated above.

Specifically, we use well-separated and semi-separated decompositions to compute these subsets.

Fat triangles

In Section 3.4 we give a construction of local spanners for the family $\mathcal{F}$ of homothets of a given triangle $\triangle$, and get a spanner of size $\mathcal{O}\bigl((\alpha\varepsilon)^{-1} n\bigr)$ in $\mathcal{O}\bigl((\alpha\varepsilon)^{-1} n \log n\bigr)$ time, where α is the smallest angle in $\triangle$. This construction is a careful adaptation of the θ-graph spanner construction to the given triangle, and it is significantly more technically challenging than the original construction.

k-regular polygons

It seems natural that if one can handle fat triangles, then homothets of k-regular polygons should readily follow by a simple decomposition of the polygon into fat triangles. Maybe surprisingly, this is not the case – a critical configuration might involve two points that are on the interior of two non-adjacent edges of a homothet of the input polygon. We overcome this by first showing that sufficiently narrow trapezoids provide us with a good jump somewhere inside the trapezoid, assuming one computes the Delaunay triangulation induced by the trapezoid, and that the source and destination lie on the two legs of the trapezoid. Next, we show that such a polygon can be covered by a small number of narrow trapezoids and fat triangles. By building appropriate graphs for each trapezoid/triangle in the collection, we get a spanner for homothets of the given k-regular polygon, with size that has no dependency on the spread. Of course, the size does depend polynomially on k. See Section 3.5 for details, and Theorem 26 for the precise result.

Multiplicative weak local spanner for rectangles

In the final result we use a less known type of pair-decomposition to construct a weak local spanner for axis-parallel rectangles. Here, the graph G, constructed over P, has the property that for any axis-parallel rectangle R, the graph $G \cap R$ is a $(1+\varepsilon)$-spanner for all the points of $\bigl((1-\delta)R\bigr) \cap P$, where $(1-\delta)R$ is the scaling of the rectangle by a factor of $1-\delta$ around its center. Intuitively, δ is a parameterization of the weakness of the spanner, which guarantees $(1+\varepsilon)$-paths for smaller regions as δ approaches 1. Importantly, this works for narrow rectangles where this form of multiplicative shrinking is still meaningful (unlike the diameter based shrinking mentioned above). Contrast this with the lower bound (illustrated in Figure 1.1) of $\Omega(n^2)$ on the size of local spanner if one does not shrink the rectangles. See Section 4 for details of the precise result.

See Table 1.1 for a summary of known results and comparisons to the results of this paper.

2 Preliminaries

Residual graphs

▶ **Definition 1.** *Let $\mathcal{F}$ be a family of regions in the plane. For a region $\boldsymbol{r} \in \mathcal{F}$ and a geometric graph G on a point set P, let $G - \boldsymbol{r}$ be the residual graph after removing from G all the points of P in $\boldsymbol{r}$ and all the edges whose corresponding segments intersect $\boldsymbol{r}$. Similarly, let $G \cap \boldsymbol{r}$ denote the graph restricted to $\boldsymbol{r}$. Formally, let*

$$G - \boldsymbol{r} = \bigl(P \setminus \boldsymbol{r}, \{uv \in E \mid uv \cap \mathrm{int}(\boldsymbol{r}) = \emptyset\}\bigr) \qquad \textit{and} \qquad G \cap \boldsymbol{r} = \bigl(P \cap \boldsymbol{r}, \{uv \in E \mid uv \subseteq \boldsymbol{r}\}\bigr).$$

where $\mathrm{int}(\boldsymbol{r})$ denotes the interior of $\boldsymbol{r}$,

2.1 On various pair decompositions

For sets X, Y, let $X \otimes Y = \{\{x, y\} \mid x \in X,\ y \in Y, x \neq y\}$ be the set of all the (unordered) pairs of points formed by the sets X and Y.

▶ **Definition 2** (Pair decomposition). *For a point set P, a **pair decomposition** of P is a set of pairs*

$$\mathcal{W} = \bigl\{\{X_1, Y_1\}, \ldots, \{X_s, Y_s\}\bigr\},$$

such that

(I) *$X_i, Y_i \subseteq P$ for every i,*
(II) *$X_i \cap Y_i = \emptyset$ for every i, and*
(III) *$\bigcup_{i=1}^{s} X_i \otimes Y_i = P \otimes P$.*

*Its **weight** is $\omega(\mathcal{W}) = \sum_{i=1}^{s}(|X_i| + |Y_i|)$.*

The **closest pair** distance of a set $P \subseteq \mathbb{R}^d$ of points is $\mathrm{cp}(P) = \min_{p,q \in P, p \neq q} \|pq\|$. The **diameter** of P is $\mathrm{diam}(P) = \max_{p,q \in P} \|pq\|$. The **spread** of P is $\Phi(P) = \mathrm{diam}(P)/\mathrm{cp}(P)$, which is the ratio between the diameter and closest pair distance. While in general the weight of a WSPD (defined below) can be quadratic, if the spread is bounded, the weight is near linear. For $X, Y \subseteq \mathbb{R}^d$, let $\mathsf{d}(X, Y) = \min_{p \in X, q \in Y} \|pq\|$ be the **distance** between the two sets.

▶ **Definition 3.** *Two sets $X, Y \subseteq \mathbb{R}^d$ are*

$$\begin{array}{llll} & 1/\varepsilon\textit{-}\textbf{\textit{well-separated}} & \textit{if} & \mathbf{max}(\mathrm{diam}(X), \mathrm{diam}(Y)) \leq \varepsilon \cdot \mathsf{d}(X, Y), \\ \textit{and} & 1/\varepsilon\textit{-}\textbf{\textit{semi-separated}} & \textit{if} & \mathbf{min}(\mathrm{diam}(X), \mathrm{diam}(Y)) \leq \varepsilon \cdot \mathsf{d}(X, Y). \end{array}$$

For a point set P, a **well-separated pair decomposition** *(***WSPD***) of P with parameter ε is a pair decomposition of P with a set $\mathcal{W} = \bigl\{\{B_1, C_1\}, \ldots, \{B_s, C_s\}\bigr\}$, of pairs such that for all i, the sets B_i and C_i are $(1/\varepsilon)$-well-separated. The notion of $(1/\varepsilon)$-SSPD (a.k.a.* ***semi-separated pair decomposition****) is defined analogously.*

▶ **Lemma 4** ([1])**.** *Let P be a set of n points in $\mathbb{R}^d$, with spread $\Phi = \Phi(P)$, and let $\varepsilon > 0$ be a parameter. Then, one can compute a $(1/\varepsilon)$-WSPD $\mathcal{W}$ for P of total weight $\omega(\mathcal{W}) = \mathcal{O}(n\varepsilon^{-d}\log\Phi)$. Furthermore, any point of P participates in at most $\mathcal{O}\bigl(\varepsilon^{-d}\log\Phi\bigr)$ pairs.*

▶ **Theorem 5** ([1, 7])**.** *Let P be a set of n points in $\mathbb{R}^d$, and let $\varepsilon > 0$ be a parameter. Then, one can compute a $(1/\varepsilon)$-SSPD for P of total weight $\mathcal{O}\bigl(n\varepsilon^{-d}\log n\bigr)$. The number of pairs in the SSPD is $\mathcal{O}\bigl(n\varepsilon^{-d}\bigr)$, and the computation time is $\mathcal{O}\bigl(n\varepsilon^{-d}\log n\bigr)$.*

The following claim is straightforward.

▶ **Lemma 6.** *Given an α-SSPD $\mathcal{W}$ of a set P of n points in $\mathbb{R}^d$ and a parameter $\beta \geq 2$, one can refine $\mathcal{W}$ into an $\alpha\beta$-SSPD $\mathcal{W}'$, such that $|\mathcal{W}'| = \mathcal{O}(|\mathcal{W}|/\beta^d)$ and $\omega(\mathcal{W}') = \mathcal{O}(\omega(\mathcal{W})/\beta^d)$.*

▶ **Definition 7.** *An* ***ε-double-wedge*** *is a region between two lines, where the angle between the two lines is at most ε.*

Two point sets X and Y that each lie in their own face of a shared ε-double-wedge are ***ε-angularly separated****.*

▶ **Theorem 8** (Proof in full version [5])**.** *Given a $(1/\varepsilon)$-SSPD $\mathcal{W}$ of n points in the plane, one can refine $\mathcal{W}$ into a $(1/\varepsilon)$-SSPD $\mathcal{W}'$, such that each pair $\Xi = \{X, Y\} \in \mathcal{W}'$ is contained in an ε-double-wedge $\times_\Xi$, such that X and Y are contained in the two different faces of the double wedge $\times_\Xi$. We have that $|\mathcal{W}'| = \mathcal{O}(|\mathcal{W}|/\varepsilon)$ and $\omega(\mathcal{W}') = \mathcal{O}(\omega(\mathcal{W})/\varepsilon)$. The construction time is proportional to the weight of $\mathcal{W}'$.*

▶ **Corollary 9.** *Let P be a set of n points in the plane, and let $\varepsilon > 0$ be a parameter. Then, one can compute a $(1/\varepsilon)$-SSPD for P such that every pair is ε-angularly separated. The total weight of the SSPD is $\mathcal{O}\bigl(n\varepsilon^{-3}\log n\bigr)$, the number of pairs in the SSPD is $\mathcal{O}\bigl(n\varepsilon^{-3}\bigr)$, and the computation time is $\mathcal{O}\bigl(n\varepsilon^{-3}\log n\bigr)$.*

2.2 Weak local spanners for fat convex regions

▶ **Definition 10.** *Given a convex region C, let*

$$C_{\boxminus\delta} = \bigl\{ p \in C \bigm| \mathsf{d}\bigl(p, \mathbb{R}^2 \setminus C\bigr) \geq \delta \cdot \mathrm{diam}(C) \bigr\}.$$

In other words, $C_{\boxminus\delta}$ is the Minkowski difference of C with a disk of radius $\delta \cdot \mathrm{diam}(C)$.

▶ **Definition 11.** *Consider a (bounded) set C in the plane. Let $r_{\mathrm{in}}(C)$ be the radius of the largest disk contained inside C. Similarly, $R_{\mathrm{out}}(C)$ is the smallest radius of a disk containing C.*

The ***aspect ratio*** *of a region C in the plane is $\mathsf{ar}(C) = R_{\mathrm{out}}(C)/r_{\mathrm{in}}(C)$. Given a family $\mathcal{F}$ of regions in the plane, its aspect ratio is $\mathsf{ar}(\mathcal{F}) = \max_{C\in\mathcal{F}} \mathsf{ar}(C)$.*

Note, that if a convex region C has bounded aspect ratio, then $C_{\boxminus\delta}$ is similar to the result of scaling C by a factor of $1 - \mathcal{O}(\delta)$. On the other hand, if C is long and skinny then this region is much smaller. Specifically, if C has width smaller than $2\delta \cdot \mathrm{diam}(C)$, then $C_{\boxminus\delta}$ is empty.

▶ **Lemma 12** (Proof in full version [5]). *Given a set P of n points in the plane, and parameters $\delta, \varepsilon \in (0,1)$, one can construct a graph G over P, in $\mathcal{O}((\varepsilon^{-1} + \delta^{-2}) n \log n)$ time, and with $\mathcal{O}\big((\varepsilon^{-1} + \delta^{-2}) n\big)$ edges, such that for any (bounded) convex region C in the plane, we have that for any two points $p, q \in P \cap C_{\boxminus\delta}$ the graph $C \cap P$ has a $(1 + \varepsilon)$-path between p and q.*

3 Local spanners of homothets of convex region

Let $\mathcal{C}$ be a bounded convex and closed region in the plane (e.g., a disk). A **homothet** of $\mathcal{C}$ is a scaled and translated copy of $\mathcal{C}$. A point set P is in **general position** with respect to $\mathcal{C}$, if no four points of P lie on the boundary of a homothet of $\mathcal{C}$, and no three points are colinear.

A graph $G = (P, E)$ is a *$\mathcal{C}$-local t-spanner* for P if for any homothet $\mathscr{r}$ of $\mathcal{C}$ we have that $G \cap \mathscr{r}$ is a t-spanner of $\mathcal{K}_P \cap \mathscr{r}$.

3.1 Delaunay triangulation for homothets

▶ **Definition 13** ([6]). *Given $\mathcal{C}$ as above, and a point set P in general position with respect to $\mathcal{C}$, the **$\mathcal{C}$-Delaunay triangulation** of P, denoted by $\mathcal{D}_{\mathcal{C}}(P)$, is the graph formed by edges between any two points $p, q \in P$ such that there is a homothet of $\mathcal{C}$ that contains only p and q and no other point of P.*

▶ **Theorem 14** ([6]). *For a set P of points, $\mathcal{D}_{\mathcal{C}}(P)$ can be computed in $\mathcal{O}(n \log n)$ time for a pre-determined convex body $\mathcal{C}$. Furthermore, the triangulation $\mathcal{D}_{\mathcal{C}}(P)$ has $\mathcal{O}(n)$ edges, vertices, and faces.*

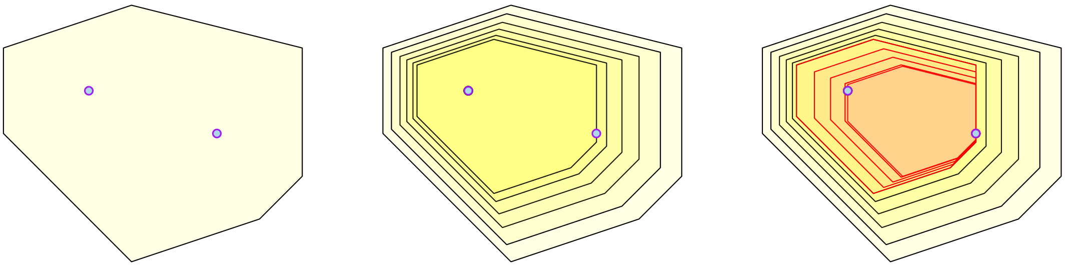


Figure 3.1 Shrinking of a homothet so that two specific points would lie on its boundary.

▶ **Lemma 15** (Proof in full version [5]). *Let $\mathcal{C}$ be a convex body, and let P be a set of points in general position with respect to $\mathcal{C}$. Then, if C is a homothet of $\mathcal{C}$ that contains two points $p, q \in P$, then there exists a homothet $C' \subseteq C$ of $\mathcal{C}$ such that $p, q \in \partial C'$.*

See Figure 3.1 for An illustration of the claim in Lemma 15.

The following standard claim, usually stated for the standard Delaunay triangulations, also holds for homothets.

▷ Claim 16 (Proof in full version [5]). Let $\mathcal{C}$ be a convex body. Given a set $P \subseteq \mathbb{R}^2$ of points in general position with respect to $\mathcal{C}$, let $\mathcal{D} = \mathcal{D}_{\mathcal{C}}(P)$ be the $\mathcal{C}$-Delaunay triangulation of P. For any homothet C of $\mathcal{C}$, we have that $\mathcal{D} \cap C$ is connected.

3.2 The generic construction

The input is a set P of n points in the plane (in general position) with spread $\Phi = \Phi(P)$, a parameter $\varepsilon \in (0,1)$, and a convex body $\mathcal{C}$ that defines the "unit" ball. The task is to construct $\mathcal{C}$-local spanner.

The algorithm computes a $(1/\vartheta)$-WSPD $\mathcal{W}$ of P using the algorithm of Lemma 4, where $\vartheta = \varepsilon/6$. For each pair $\Xi = \{X, Y\} \in \mathcal{W}$, the algorithm computes the $\mathcal{C}$-Delaunay triangulation $\mathcal{D}_\Xi = \mathcal{D}_\mathcal{C}(X \cup Y)$, and adds all the edges in $\mathcal{D}_\Xi \cap (X \otimes Y)$ to the computed graph G.

▶ Remark 17. In the above algorithm, the idea of computing a triangulation for each WSPD pair seems to be new.

3.2.1 Analysis

Size. For each pair $\Xi = \{X, Y\}$ in the WSPD, its $\mathcal{C}$-Delaunay triangulation contains at most $\mathcal{O}(|X| + |Y|)$ edges. As such, the number of edges in the resulting graph is bounded by $\sum_{\{X,Y\}\in\mathcal{W}} O\bigl(|X| + |Y|\bigr) = \mathcal{O}(\omega(\mathcal{W})) = \mathcal{O}\bigl(n\vartheta^{-2}\log\Phi\bigr)$, by Lemma 4.

Construction time. The construction time is bounded by

$$\sum_{\{X,Y\}\in\mathcal{W}} O\bigl((|X| + |Y|)\log(|X| + |Y|)\bigr) = \mathcal{O}(\omega(\mathcal{W})\log n) = \mathcal{O}\bigl(n\vartheta^{-2}\log\Phi\log n\bigr).$$

▶ **Lemma 18** (Local spanner property). *For $P, \mathcal{C}, \varepsilon$ as above, let G be the graph constructed above for the point set P. Then, for any homothet C of $\mathcal{C}$ and any two points $x, y \in P \cap C$, we have that $G \cap C$ has a $(1 + \varepsilon)$-path between x and y. That is, G is a $\mathcal{C}$-local $(1 + \varepsilon)$-spanner.*

Proof. Fix a homothet C of $\mathcal{C}$, and consider two points $p, q \in P \cap C$. The proof is by induction on the distance between p and q (or more precisely, the rank of their distance among the $\binom{n}{2}$ pairwise distances). Consider the pair $\Xi = \{X, Y\}$ such that $x \in X$ and $y \in Y$.

If $xy \in \mathcal{D}_\Xi$ then the claim holds, so assume this is not the case. By the connectivity of $\mathcal{D}_\Xi \cap C$, see Claim 16, there must be points $x' \in X \cap C$, $y' \in Y \cap C$, such that $x'y' \in E(\mathcal{D}_\Xi)$. Indeed, let $x \in X \cap C$, $y \in Y \cap C$, and let π be some (x, y)-path guaranteed to exist by connectivity. π must contain an edge with one endpoint in $X \cap C$ and the other in $Y \cap C$. As such, by construction, we have that $x'y' \in E(G)$. Furthermore, by the separation property, we have that

$$\max(\mathrm{diam}(X), \mathrm{diam}(Y)) \le \vartheta\, \mathsf{d}(X, Y) \le \vartheta\ell,$$

where $\ell = \|xy\|$. In particular, $\|x'x\| \le \vartheta\ell$ and $\|y'y\| \le \vartheta\ell$. As such, by induction, we have $\mathsf{d}_G(x, x') \le (1 + \varepsilon)\|xx'\| \le (1 + \varepsilon)\vartheta\ell$ and $\mathsf{d}_G(y, y') \le (1 + \varepsilon)\|yy'\| \le (1 + \varepsilon)\vartheta\ell$. Furthermore, $\|x'y'\| \le (1 + 2\vartheta)\ell$. As $x'y' \in E(G)$, we have

$$\begin{aligned}\mathsf{d}_G(x, y) &\le \mathsf{d}_G(x, x') + \|x'y'\| + \mathsf{d}_G(y', y) \le (1 + \varepsilon)\vartheta\ell + (1 + 2\vartheta)\ell + (1 + \varepsilon)\vartheta\ell\\ &\le (2\vartheta + 1 + 2\vartheta + 2\vartheta)\ell\\ &= (1 + 6\vartheta)\ell \le (1 + \varepsilon)\|xy\|,\end{aligned}$$

if $\vartheta \le \varepsilon/6$. ◀

The result. We thus get the following.

▶ **Theorem 19.** *Let $\mathcal{C}$ be a convex body in the plane, let P be a given set of n points in the plane (in general position with respect to $\mathcal{C}$), and let $\varepsilon \in (0, 1/2)$ be a parameter. The above algorithm constructs a $\mathcal{C}$-local $(1 + \varepsilon)$-spanner G. The spanner has $\mathcal{O}\bigl(\varepsilon^{-2}n\log\Phi\bigr)$ edges, and the construction time is $\mathcal{O}\bigl(\varepsilon^{-2}n\log\Phi\log n\bigr)$. Formally, for any homothet C of $\mathcal{C}$, and any two points $p, q \in P \cap C$, we have a $(1 + \varepsilon)$-path in $G \cap C$.*

3.3 Lower bounds

3.3.1 A lower bound for local spanner for disks

The result of Theorem 19 is somewhat disappointing as it depends on the spread of the point set (logarithmically, but still). Next, we show a lower bound proving that this dependency is unavoidable, even in the case of disks.

Some intuition. A natural way to attempt a spread-independent construction is to try and emulate the construction of Abam et al. [3] and use an SSPD instead of a WSPD, as the total weight of the SSPD is near linear (with no dependency on the spread). Furthermore, after some post-processing, one can assume every pair $\Xi = \{X, Y\}$ is angularly ε-separated – that is, there is a double wedge with angle $\leq \varepsilon$, such that X and Y are on different sides of the double wedge. The problem is that for a disk $\bigcirc$, it might be that the bridge edge between X and Y that is in $\mathcal{D}_\Xi \cap \bigcirc$ is much longer than the distance between the two points of interest. This somewhat counter-intuitive situation is illustrated in Figure 3.2.

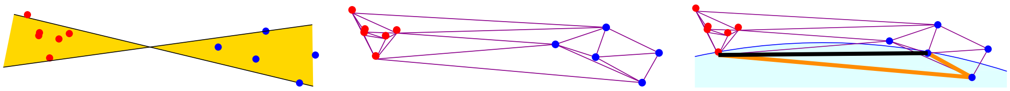


Figure 3.2 A bridge too far – the only surviving bridge between the red and blue points is too far to be useful if the sets of points are not well separated.

▶ **Lemma 20.** *For $\varepsilon = 1/4$, and parameters n and Φ, there is a point set P of $n + \lceil \log_2 \Phi \rceil$ points in the plane, with spread $\mathcal{O}(n\Phi)$, such that any local $(1+\varepsilon)$-spanner of P for disks must have $\Omega\big(n(1 + \log \frac{\Phi}{n})\big)$ edges, as long as $\sqrt{n} \leq \Phi \leq n2^n$.*

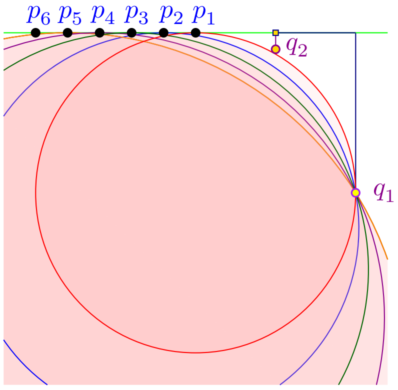


Figure 3.3 The set of disks D_1, and the construction of q_2.

Proof. Let $p_i = (-i, 0)$, for $i = 1, \ldots, n$. Let $M = 1 + \lceil \log_2 \Phi \rceil$, $x_1 = n2^M$ and $q_1 = (x_1, -1)$. For a point p on the x-axis, and a point q below the x-axis and to the right of p, let $\bigcirc^{p}_{\downarrow}(q)$ be the disk whose boundary passes through p and q, and its center has the same x-coordinate as p. In the jth iteration, for $j = 2, \ldots, M-1$, Let $x_j = n2^{M-j+1} = x_{j-1}/2$, and let $y_j < 0$ be the maximum y-coordinate of a point that lies on the intersection of the vertical line $x = x_j$ and the union of disks $D_1 \cup \cdots \cup D_j$ where

$$D_j = \left\{ \bigcirc^{p_i}_{\downarrow}(q_{j-1}) \;\middle|\; i = 1, \ldots, n \right\},$$

see Figure 3.3 for an illustration of D_1.

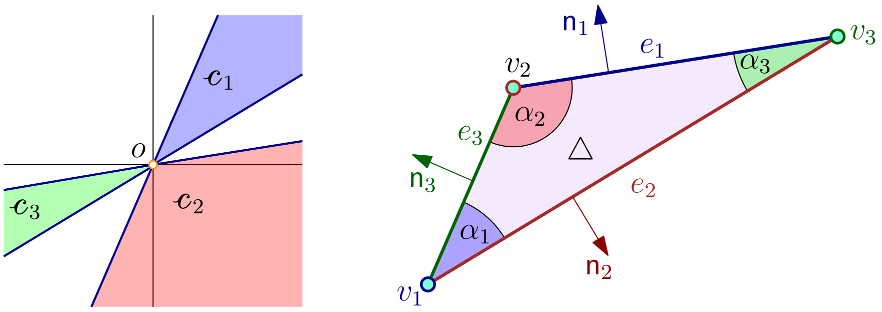


Figure 3.4 For the triangle $\triangle$ with angles α_1, α_2, and α_3 we create the cones $\cancel{c}_1, \cancel{c}_2$, and $\cancel{c}_3$.

Let $q_j = (x_j, 0.99y_j)$.

Clearly, the point q_j lies outside all the disks of $D_1 \cup \ldots \cup D_j$. The construction now continues to the next value of j. Let $P = \{p_1, \ldots, p_n, q_2, \ldots, q_M\}$. We have that $|P| = n + M - 1$.

The minimum distance between any points in the construction is 1 (i.e., $\|p_1p_2\|$). Indeed $x_{M-1} = 4n$ and thus $\|q_{M-1}p_1\| \geq 2n$. The diameter of P is $\|p_1q_1\| = \sqrt{(n + n2^M)^2 + 1} \leq 2n2^M$. As such, the spread of P is bounded by $\leq n2^{M+1} = \mathcal{O}(n\Phi)$.

For any i and j, consider the disk $\bigcirc_{\downarrow}^{p_i}(q_j)$. This disk does not contain any point of $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ since its interior lies below the x-axis. By construction it does not contain any point $q_{j+1}, \ldots, q_{M-1}$. This disk potentially contains the points $q_{j-1}, \ldots, q_1$, but observe that for any index $k \in [\![j-1]\!] = \{1, \ldots, j-1\}$, we have that

$$\|p_iq_k\| = \sqrt{\left(i + n2^{M-k+1}\right)^2 + \left(y(q_j)\right)^2},$$

which implies that $n2^{M-k+1} \leq \|p_iq_k\| < n(2^{M-k+1} + 2)$. We thus have that

$$\frac{\|p_iq_k\|}{\|p_iq_j\|} \geq \frac{n2^{M-k+1}}{n(2^{M-j+1} + 2)} = \frac{2^{M-j} \cdot 2^{j-k}}{2^{M-j} + 1} = \frac{2^{j-k}}{1 + 1/2^{M-j}} \geq \frac{2}{1 + 1/2} = \frac{4}{3} > 1 + \varepsilon,$$

since $j \in [\![M-1]\!]$. Namely, the shortest path in G between p_i and q_j, cannot use any of the points $q_1, \ldots q_{j-1}$. As such, the graph G must contain the edge p_iq_j. This implies that $|E(G)| \geq n(M-1)$, which implies the claim. ◀

3.3.2 A lower bound for triangles

▶ **Lemma 21** (Proof in full version [5]). *For any $n > 0$, and $\Phi = \Omega(n)$, one can compute a set P of $n + \mathcal{O}(\log \Phi)$ points, with spread $\mathcal{O}(\Phi n)$, and a triangle $\triangle$, such that any $\triangle$-local $(3/2)$-spanner of P requires $\Omega\left(n \log(1 + \frac{\Phi}{n})\right)$ edges.*

3.4 Local spanners for fat triangles

While local spanners for homothets of an arbitrary convex body are costly, if we are given a triangle $\triangle$ with the single constraint that $\triangle$ is not too "thin", then one can construct a $\triangle$-local t-spanner with a number of edges that does not depend on the spread of the points.

▶ **Definition 22.** *A triangle $\triangle$ is α-**fat** if the smallest angle in $\triangle$ is at least α.*

3.4.1 Construction

The input is a set P of n points in the plane, an α-fat triangle $\triangle$, and an approximation parameter $\varepsilon \in (0,1)$. Let v_i denote the ith vertex of $\triangle$, α_i be the adjacent angle, and let e_i denote the opposing edge, for $i \in [\![3]\!]$. Let $\mathscr{c}_i = \{(p - v_i)t \mid p \in e_i \text{ and } t \geq 0\}$ denote the cone with an apex at the origin induced by the ith vertex of $\triangle$. Let n_i be the outer normal of $\triangle$ orthogonal to e_i. See Figure 3.4 for an illustration. Let $\mathcal{C}_i$ be a minimum size partition of $\mathscr{c}_i$ into cones each with angle in the range $[\beta/2, \beta]$, where $\beta = \varepsilon\alpha/\gamma$, and $\gamma > 1$ is some constant discussed shortly. For each point $p \in P$, and a cone $\mathscr{c} \in \mathcal{C}_i$, let $\mathsf{nn}_i(p, \mathscr{c})$ be the first point in $(P - p) \cap (p + \mathscr{c})$ ordered by the direction n_i (it is the "nearest-neighbor" to p in $p + \mathscr{c}$ with respect to the direction n_i).

The result

Let G be the graph over P formed by connecting every point $p \in P$ to $\mathsf{nn}_i(p, \mathscr{c})$, for all $i \in [\![3]\!]$ and $\mathscr{c} \in \mathcal{C}_i$. We get the following result (see full version [5] for details).

▶ **Theorem 23.** *Let P be a set of n points in the plane, and let $\varepsilon \in (0,1)$ be an approximation parameter. The above algorithm computes a $\triangle$-local $(1+\varepsilon)$-spanner G for an α-fat triangle $\triangle$. The construction time is $\mathcal{O}\big((\alpha\varepsilon)^{-1} n \log n\big)$, and the spanner G has $\mathcal{O}\big((\alpha\varepsilon)^{-1} n\big)$ edges.*

3.5 A local spanner for nice polygons

3.5.1 A good jump for narrow trapezoids

As a reminder, a trapezoid is a quadrilateral with two parallel edges, known as its *bases*. The other two edges are its *legs*. For $\varepsilon \in (0, 1/4)$, a trapezoid T is **ε-narrow** if the length of each of its legs is at most $\varepsilon \cdot \mathrm{diam}(T)$.

▶ **Lemma 24** (Proof in full version [5]). *Let $\varepsilon \in (0,1)$ be some parameter, and $\vartheta = \varepsilon/16$. Let X, Y be two point sets that are $(1/\vartheta)$-semi separated and ϑ-angularly separated (see Definition 7), and let T be a ϑ-narrow trapezoid, with two points $p \in X$ and $q \in Y$ lying on the two legs of T. Then, one can compute a homothet $T' \subseteq T$ of T such that*

1. *there are two points $p' \in X$ and $q' \in Y$, such that $p'q'$ is an edge of the T-Delaunay triangulation of $X \cup Y$, and*
2. *we have that $(1+\varepsilon)\,\|pp'\| + \|p'q'\| + (1+\varepsilon)\,\|q'q\| \leq (1+\varepsilon)\,\|pq\|$.*

3.5.2 Breaking a nice polygon into narrow trapezoids

For a convex polygon $\mathcal{C}$, its **sensitivity**, denoted by $\mathrm{sen}(\mathcal{C})$, is the minimum distance between any two non-adjacent edges (this quantity is no bigger than the length of the shortest edge in the polygon). A convex polygon $\mathcal{C}$ is **t-nice**, if the outer angle at any vertex of the polygon is at least $2\pi/t$, and the length of the longest edge of $\mathcal{C}$ is $\mathcal{O}(\mathrm{sen}(\mathcal{C}))$. As an example, a k-regular polygon is k-nice.

▶ **Lemma 25** (Proof in full version [5]). *Let t be a positive integer. Given a t-nice polygon $\mathcal{C}$, and a parameter ϑ, one can cover it by a set $\mathcal{T}$ of $\mathcal{O}(t^4/\vartheta^3)$ ϑ-narrow trapezoids, such that for any two points $p, q \in \partial\mathcal{C}$ that belong to two edges of $\mathcal{C}$ that are not adjacent, there exists a narrow trapezoid $T \in \mathcal{T}$, such that p and q are located on two different short legs of T.*

3.5.3 Constructing the local spanner for nice polygons

▶ **Theorem 26** (Proof in full version [5]). *Let $\mathcal{C}$ be a k-nice convex polygon, P be a set of n points in the plane, and let $\varepsilon \in (0,1)$ be a parameter. Then, one can construct a $\mathcal{C}$-local $(1+\varepsilon)$-spanner of P. The construction time is $\mathcal{O}\big((k^4/\varepsilon^6)n \log^2 n\big)$, and the resulting graph has $\mathcal{O}\big((k^4/\varepsilon^6)n \log n\big)$ edges. In particular these bounds hold if $\mathcal{C}$ is a k-regular polygon.*

▶ Remark 27. For axis-parallel squares Theorem 26 implies a local spanner with $\mathcal{O}\big(\varepsilon^{-6} n \log n\big)$ edges. However, for this special case, the decomposition into narrow trapezoid can be skipped. In particular, in this case, the resulting spanner has $\mathcal{O}(\varepsilon^{-3} n \log n)$ edges. We do not provide the details here, as it is only a minor improvement over the above, and requires quite a bit of additional work – essentially, one has to prove a version of Lemma 24 for squares. We leave the question of whether this bound can be further improved as an open problem for further research.

4 Weak local spanners for axis-parallel rectangles

4.1 Orthant separated pair decomposition

For the purpose of building the spanners in this section, we use a variation of a pair decomposition introduced by Agarwal et al. [4]. For two points $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ in $\mathbb{R}^d$, let $p \prec q$ denote that q **dominates** p coordinate-wise. That is $p_i < q_i$, for all i. More generally, let $p <_i q$ denote that $p_i < q_i$. For two point sets $X, Y \subseteq \mathbb{R}^d$, we use $X <_i Y$ to denote that $x <_i y \quad \forall x \in X, y \in Y$. In particular X and Y are *i-coordinate separated* if $X <_i Y$ or $Y <_i X$. A pair $\{X, Y\}$ is **orthant-separated**, if X and Y are i-coordinate separated, for all $i = 1, \ldots, d$.

A **orthant-separated pair decomposition** of a point set $P \subseteq \mathbb{R}^d$, is a pair decomposition (see Definition 2) $\mathcal{W} = \big\{\{X_1, Y_1\}, \ldots, \{X_s, Y_s\}\big\}$ of P such that $\{X_i, Y_i\}$ are orthant-separated for all i.

In the full version of the paper [5], we prove the properties regarding the computational and combinatorial complexity of OSPDs that are used in the proof of Theorem 31

4.2 Weak local spanner for axis-parallel rectangles

For a parameter $\delta \in (0,1)$, and an interval $I = [b, c]$, let $(1-\delta)I = [t - (1-\delta)r, t + (1-\delta)r]$, where $t = (b+c)/2$, and $r = (c-b)/2$, be the shrinking of I by a factor of $1-\delta$.

Let $\mathcal{R}$ be the set of all axis-parallel rectangles in the plane. For a rectangle $R \in \mathcal{R}$ with $R = I \times J$, let $(1-\delta)R = (1-\delta)I \times (1-\delta)J$ denote the rectangle resulting from shrinking R by a factor of $1-\delta$.

▶ **Definition 28.** *Given a set P of n points in the plane, and parameters $\varepsilon, \delta \in (0,1)$, a graph G is a* $(1-\delta)$-local $(1+\varepsilon)$-spanner *for rectangles, if for any axis-parallel rectangle R, we have that $G \cap R$ is a $(1+\varepsilon)$-spanner for all the points in $\big((1-\delta)R\big) \cap P$.*

Observe that rectangles in $\mathcal{R}$ might be quite “skinny”, so the previous notion of shrinkage used before is not useful in this case.

4.2.1 Construction for a single orthant separated pair

Consider a pair $\Xi = \{X, Y\}$ in a OSPD of P. The set X is orthant-separated from Y, that is, there is a point c_Ξ such that X and Y are contained in two opposing orthants in the partition of the plane formed by the vertical and horizontal lines through c_Ξ.

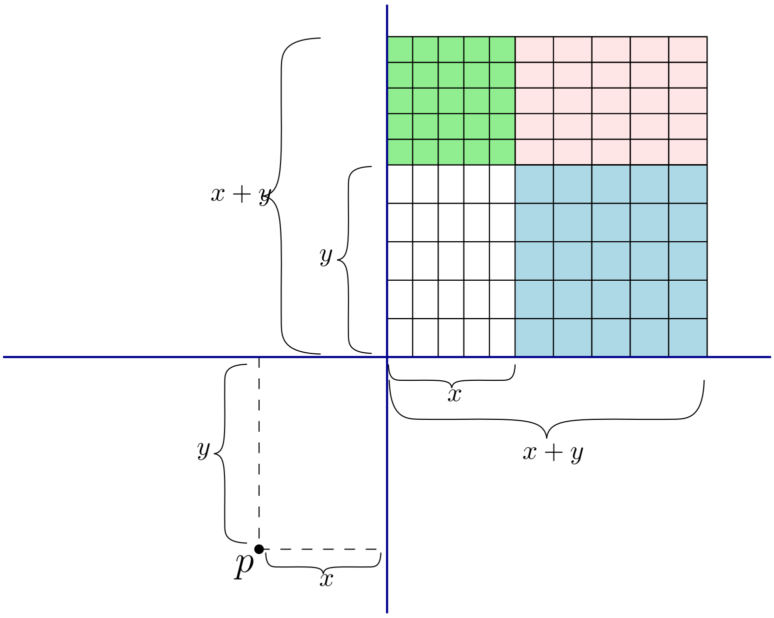


Figure 4.1 The construction of the grid $\mathsf{K}(p, \Xi)$ for a point $p = (-x, -y)$ and a pair Ξ.

For simplicity of exposition, assume that $c_\Xi = (0,0)$, and $X \prec (0,0) \prec Y$. That is, the points of X are in the negative orthant, and the points of Y are in the positive orthant.

For a point $p = (-x,-y) \in X$ we construct a non-uniform grid $\mathsf{K}(p,\Xi)$ in the square $[0, x+y]^2$. To this end, we first partition it into four subrectangles

$B_{\nwarrow} = [0,x] \times [y, x+y]$	$B_{\nearrow} = [x, x+y] \times [y, x+y]$
$B_{\swarrow} = [0,x] \times [0,y]$	$B_{\searrow} = [x, x+y] \times [0,y]$.

Let $\tau \geq 4/\varepsilon + 4/\delta$ be an integer number. We partition each of these rectangles into a $\tau \times \tau$ grid, where each cell is a copy of the rectangle scaled by a factor of $1/\tau$. See Figure 4.1. This grid has $\mathcal{O}(\tau^2)$ cells. For a cell C in this grid, let $Y \cap \mathsf{C}$ be the points of Y contained in it. We connect p to the left-most and bottom-most points in $Y \cap \mathsf{C}$. This process generates two edges in the constructed graph for each grid cell (that contains at least two points), and $\mathcal{O}(\tau^2)$ edges overall.

The algorithm repeats this construction for all the points $p \in X$, and does the symmetric construction for all the points of Y.

4.2.2 The spanner construction algorithm

The algorithm computes a OSPD $\mathcal{W}$ of P. For each pair $\Xi \in \mathcal{W}$, the algorithm generates edges for Ξ using the algorithm of Section 4.2.1 and adds them to the generated spanner G.

4.2.3 Correctness

For a rectangle R, let $\overset{\leftrightarrow}{R} = \left\{(x,y) \in \mathbb{R}^2 \;\middle|\; \exists x' \in \mathbb{R} \text{ such that } (x',y) \in R\right\}$ be its expansion into a horizontal slab. Restricted to a rectangle R', the resulting set is $\overset{\leftrightarrow}{R} \cap R'$, depicted in Figure 4.2. Similarly, we denote

$$\updownarrow R = \left\{(x,y) \in \mathbb{R}^2 \;\middle|\; \exists y' \in \mathbb{R} \text{ such that } (x,y') \in R\right\}.$$

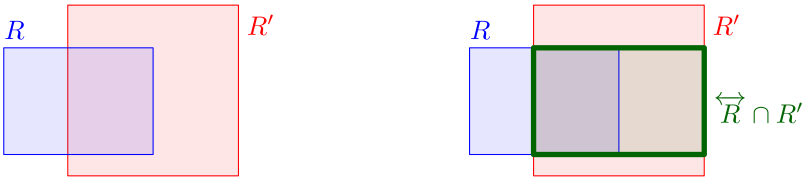


Figure 4.2 Left: The two rectangles R, R'. Right: In green $\overleftrightarrow{R} \cap R'$, the restriction of the slab $\overleftrightarrow{R}$ to the rectangle R'.

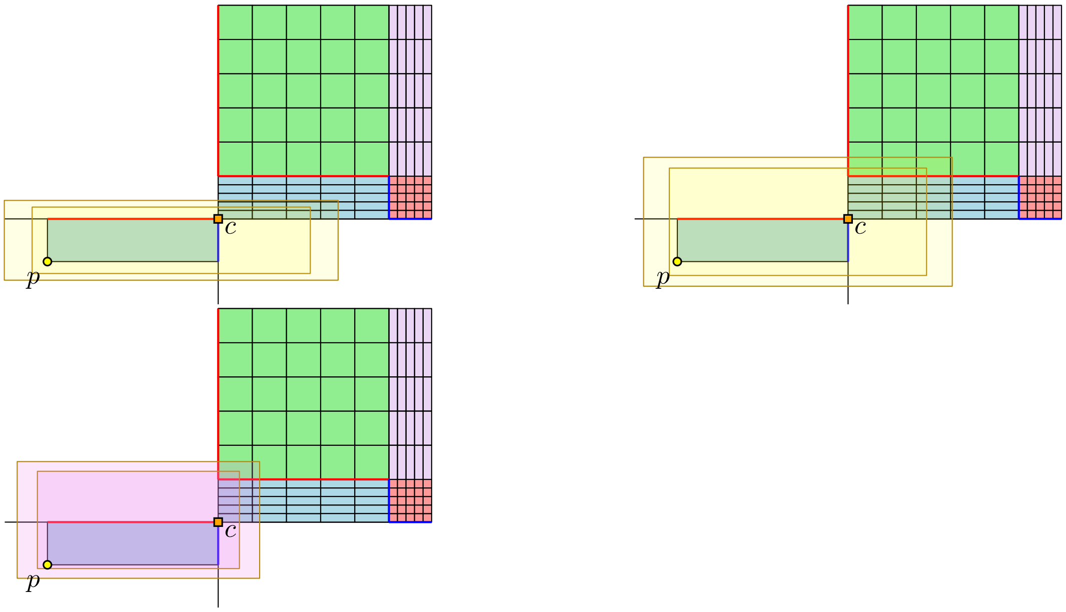


Figure 4.3 An illustration of $\mathsf{K}(p, \Xi)$ with three rectangles and their shrunken version.

▶ **Lemma 29** (Proof in full version [5]). *Assume that $\delta < 1/2$, and $\tau \geq \lceil 20/\varepsilon + 20/\delta \rceil$. Consider a pair $\Xi = \{X, Y\}$ in the above construction, and a point $p = (-x, -y) \in X$ with its associated grid $\mathsf{K} = \mathsf{K}(p, \Xi)$. Consider any axis-parallel rectangle R, such that $p \in (1-\delta)R = I \times J$, and $(1-\delta)R$ intersects a cell $\mathsf{C} \in \mathsf{K}$. We have the following.*

1. *If $\mathsf{C} \subseteq (1-\delta)R$ then $(1-\delta)^{-1}\mathsf{C} \subseteq R$.*
2. $\operatorname{diam}(\mathsf{C}) \leq (\varepsilon/4)\mathsf{d}(p, \mathsf{C})$.
3. *If $x \geq y$ and $\mathsf{C} \subseteq R_{\swarrow} \cup R_{\searrow}$ then $(1-\delta)^{-1}\mathsf{C} \subseteq R$.*
4. *If $x \leq y$ and $\mathsf{C} \subseteq R_{\swarrow} \cup R_{\nwarrow}$ then $(1-\delta)^{-1}\mathsf{C} \subseteq R$.*
5. *If $x \geq y$ and $\mathsf{C} \subseteq R_{\nwarrow}$, then $(1-\delta)^{-1}\big(\overleftrightarrow{(1-\delta)R} \cap \mathsf{C}\big) \subseteq R$.*
6. *If $x \leq y$ and $\mathsf{C} \subseteq R_{\searrow}$, then $(1-\delta)^{-1}\Big(\updownarrow\big((1-\delta)R\big) \cap \mathsf{C}\Big) \subseteq R$.*

▶ **Lemma 30** (Proof in full version [5]). *For any axis-parallel rectangle R, and any two points $p, q \in (1-\delta)R \cap P$, there exists a $(1+\varepsilon)$-path between p and q in G.*

▶ **Theorem 31** (Proof in full version [5]). *Let P be a set of n points in the plane, and let $\varepsilon, \delta \in (0,1)$ be parameters. The above algorithm constructs, in $\mathcal{O}((1/\varepsilon^2 + 1/\delta^2)n \log^2 n)$ time, a graph G with $\mathcal{O}((1/\varepsilon^2 + 1/\delta^2)n \log^2 n)$ edges. The graph G is a $(1-\delta)$-local $(1+\varepsilon)$-spanner for axis-parallel rectangles. Formally, for any axis-parallel rectangle R, we have that $R \cap P$ is an $(1+\varepsilon)$-spanner for all the points of $\big((1-\delta)R\big) \cap P$.*

References

1 M. A. Abam and S. Har-Peled. New constructions of SSPDs and their applications. *Comput. Geom. Theory Appl.*, 45(5–6):200–214, 2012. doi:10.1016/j.comgeo.2011.12.003.

2 Mohammad Ali Abam and Mohammad Sadegh Borouny. Local geometric spanners. *Algorithmica*, 83(12):3629–3648, 2021. doi:10.1007/s00453-021-00860-5.

3 Mohammad Ali Abam, Mark de Berg, Mohammad Farshi, and Joachim Gudmundsson. Region-fault tolerant geometric spanners. *Discret. Comput. Geom.*, 41(4):556–582, 2009. doi:10.1007/s00454-009-9137-7.

4 Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In Raimund Seidel, editor, *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*, pages 203–210. ACM, 1990. doi:10.1145/98524.98567.

5 Stav Ashur and Sariel Har-Peled. Local spanners revisited. *CoRR*, abs/2201.01715, 2022. arXiv:2201.01715.

6 L Paul Chew and Robert L Dyrsdale III. Voronoi diagrams based on convex distance functions. In *Proc. 1st Annu. Sympos. Comput. Geom.* (SoCG), pages 235–244, 1985.

7 S. Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Math. Surveys & Monographs*. Amer. Math. Soc., Boston, MA, USA, 2011. doi:10.1090/surv/173.

8 Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144–156, 2002. doi:10.1007/s00453-001-0075-x.

9 Tamás Lukovszki. New results of fault tolerant geometric spanners. In Frank K. H. A. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 6th International Workshop, WADS '99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, volume 1663 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 1999. doi:10.1007/3-540-48447-7_20.

10 Giri Narasimhan and Michiel H. M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

Pairwise Rearrangement is Fixed-Parameter Tractable in the Single Cut-and-Join Model

Lora Bailey ✉
Department of Mathematics, Grand Valley State University, Allendale, MI, USA

Heather Smith Blake ✉
Department of Mathematics and Computer Science, Davidson College, NC, USA

Garner Cochran ✉
Department of Mathematics and Computer Science, Berry College, Mount Berry, GA, USA

Nathan Fox ✉
Department of Quantitative Sciences, Canisius University, Buffalo, NY, USA

Michael Levet[1] ✉
Department of Computer Science, College of Charleston, SC, USA

Reem Mahmoud ✉
Department of Computer Science, Virginia Commonwealth University, Richmond, VA, USA

Inne Singgih ✉
Department of Mathematical Sciences, University of Cincinnati, OH, USA

Grace Stadnyk ✉
Department of Mathematics, Furman University, Greenville, SC, USA

Alexander Wiedemann ✉
Department of Mathematics, Randolph–Macon College, Ashland, VA, USA

Abstract

Genome rearrangement is a common model for molecular evolution. In this paper, we consider the Pairwise Rearrangement problem, which takes as input two genomes and asks for the number of minimum-length sequences of permissible operations transforming the first genome into the second. In the Single Cut-and-Join model (Bergeron, Medvedev, & Stoye, *J. Comput. Biol.* 2010), Pairwise Rearrangement is #P-complete (Bailey, et. al., COCOON 2023), which implies that exact sampling is intractable. In order to cope with this intractability, we investigate the parameterized complexity of this problem. We exhibit a fixed-parameter tractable algorithm with respect to the number of components in the *adjacency graph* that are not cycles of length 2 or paths of length 1. As a consequence, we obtain that Pairwise Rearrangement in the Single Cut-and-Join model is fixed-parameter tractable by distance. Our results suggest that the number of nontrivial components in the adjacency graph serves as the key obstacle for efficient sampling.

2012 ACM Subject Classification Theory of computation → Complexity classes; Mathematics of computing → Graph theory

Keywords and phrases Genome Rearrangement, Phylogenetics, Single Cut-and-Join, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.3

Related Version *Full Version*: https://arxiv.org/abs/2402.01942

Acknowledgements We wish to thank the American Mathematical Society for organizing the Mathematics Research Community workshop where this work began. This material is based upon work supported by the National Science Foundation under Grant Number DMS 1641020.

[1] Corresponding author

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 3; pp. 3:1–3:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

With the natural occurrence of mutations in genomes and the wide range of effects this can incite, scientists seek to understand the evolutionary relationship between species. Several discrete mathematical models have been proposed to model these mutations based on biological observations. Genome rearrangement models consider situations in which large-scale mutations alter the order of the genes within the genome. Sturtevant [17, 18] observed the biological phenomenon of genome rearrangement in the study of strains of *Drosophila* (fruit flies), only a few years after he produced the first genetic map [16]. Palmer & Herbon [15] observed similar phenomenon in plants. McClintock [10] also found experimental evidence of genes rearranging themselves, or "transposing" themselves, within chromosomes.

Subsequent to his work on *Drosophila*, Sturtevant together with Novitski [19] introduced one of the first genome rearrangement problems, seeking a minimum length sequence of operations, called a *scenario*, that would transform one genome into another. In investigating these questions, it is of key importance to balance biological relevance with computational tractability. One central issue is that of combinatorial explosion: the number of scenarios even between small genomes may be too large to handle, making it difficult to test hypotheses on all possible scenarios. Thus, we desire a polynomial time algorithm to uniformly sample from the rearrangement scenarios. Since uniform sampling is no harder than exact enumeration [8], we investigate the computational complexity of the Pairwise Rearrangement problem (Definition 5) which asks for the number of minimum-length scenarios transforming one genome into another.

The Pairwise Rearrangement problem has received significant attention in several genome rearrangement models. Pairwise Rearrangement is known to be in FP for the Single Cut or Join model [11], but is conjectured to be #P-complete for the Double Cut-and-Join model [14]. The Single Cut-and-Join model sits between these two models. Recently, Bailey, et al. [1], showed that Pairwise Rearrangement is #P-complete in the Single Cut-and-Join model. However, in practice, the key structures that serve as obstacles to efficient sampling may not necessarily appear. In particular, the relevant obstacles for efficient sampling in the Single Cut-and-Join model remain opaque.

Main Results. In this paper, we investigate the Pairwise Rearrangement problem in the Single Cut-and-Join model through the lens of parameterized complexity. Our main result is the following.

▶ **Theorem 1.** *In the Single Cut-and-Join model, Pairwise Rearrangement is fixed-parameter tractable with respect to the number of components in the adjacency graph (see Definition 6) that are not trivial (cycles of length* 2 *or paths of length* 1*).*

Our parameterization in Theorem 1 has biological significance. Indeed, chromoanagenesis is a carcinogenic mechanism that involves massive chromosomal rearrangements, which may lead to fewer components in the adjacency graph [7].

The *adjacency graph* (see Definition 6) is a bipartite multigraph illustrating where two genomes differ. Bergeron, Medvedev, and Stoye [2] established a precise relationship between the adjacency graph and the distance between two genomes in the Single Cut-and-Join model. The operations in this model induce structural changes on the adjacency graph [1, Observation 2.7]. We leverage this crucially to establish Theorem 1. Our precise technique involves developing a dynamic programming algorithm that, when the number of nontrivial components is bounded, the corresponding lookup table only has a polynomial number of entries. This establishes our claim of polynomial-time computation. We stress that arriving at the recurrence relations for the dynamic programming algorithm and proving their correctness is technical and nontrivial. Indeed this is not surprising, as Pairwise Rearrangement is #P-complete [1]. While our work is theoretical in nature, the algorithm underlying Theorem 1 in fact yields an efficient implementation (see GitHub).

We also note that if the distance (Equation (1)) between the two genomes is bounded [2], then so is the number of nontrivial components. As a consequence, we obtain the following corollary:

▶ **Corollary 2.** *In the Single Cut-and-Join model, Pairwise Rearrangement is fixed parameter tractable with respect to the distance between two genomes.*

To the best of our knowledge, parameterized complexity has received minimal attention in the genome rearrangement literature. For instance, a fixed-parameter tractable algorithm (parameterized by the number of components in the adjacency graph) for Pairwise Rearrangement in the Double Cut-and-Join model can easily be deduced from the work of [3], though the authors do not explicitly investigate the parameterized complexity of this problem. Thus, beyond providing a means of coping with the intractability of Pairwise Rearrangement in the Single Cut-and-Join model, Theorem 1 (together with the results of [3]) makes precise that the number of components in the adjacency graph serves as a key obstacle towards efficient sampling, across multiple models of genome rearrangement.

In contrast, there has been significant algorithmic work on approximation and sampling (see, for instance, [11, 12, 4, 5, 13, 9]), to cope with the intractability of enumeration. To the best of our knowledge, such approaches have not been fruitful against the Reversal model, for which the complexity of Pairwise Rearrangement is a longstanding open problem. We are not aware of any work on approximate counting or sampling for Pairwise Rearrangement in the Single Cut-and-Join model.

2 Preliminaries

We recall preliminaries regarding genome rearrangement.

▶ **Definition 3.** *A* genome *is an edge-labeled directed graph in which each label is unique and the total degree of each vertex is 1 or 2 (in-degree and out-degree combined). In particular, a genome consists of disjoint paths and cycles. The weak components of a genome we call* chromosomes*. Each edge begins at its* tail *and ends at its* head*, collectively referred to as its* extremities*. Degree 2 vertices are called* adjacencies*, and degree 1 vertices are called* telomeres*. See Figure 1.*

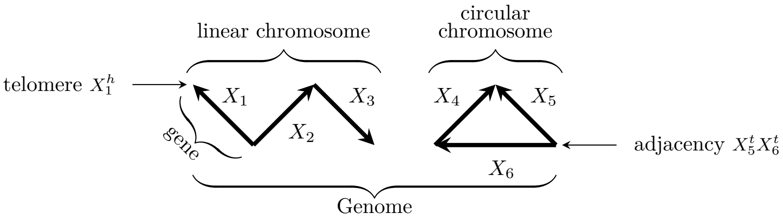


Figure 1 An edge-labeled genome [1, Fig. 1].

Adjacencies can be viewed as sets of two extremities, and telomeres as sets containing exactly one extremity. For simplicity, we write adjacency $\{a, b\}$ as ab or ba and telomere $\{c\}$ as c. For example, the adjacency $X_5^tX_6^t$ in Figure 1 denotes that the tail of the edge X_5 and the tail of the edge X_6 meet, and the telomere X_1^h is where the edge X_1 ends. Each genome is then uniquely defined by its set of adjacencies and telomeres.

Consider the following operations on a given genome:

(i) *Cut*: an adjacency ab is separated into two telomeres, a and b,

(ii) *Join*: two telomeres a and b become one adjacency, ab,

(iii) *Cut-join*: adjacency ab and telomere c are replaced with adjacency ac and telomere b, and

(iv) *Double-cut-join*: adjacencies ab and cd are replaced with adjacencies ac and bd.

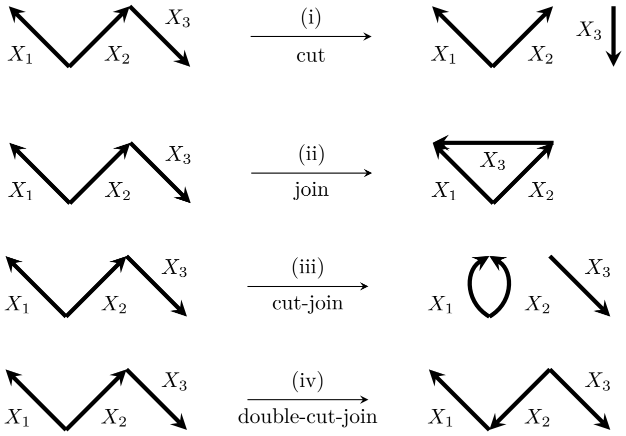


Figure 2 (i) Adjacency $X_2^h X_3^t$ is cut. (ii) Telomeres X_1^h and X_3^h are joined. (iii) Adjacency $X_2^h X_3^t$ is cut, and resulting telomere X_2^h is joined with X_1^h. (iv) Adjacencies $X_1^t X_2^t$ and $X_2^h X_3^t$ are replaced with $X_1^t X_2^h$ and $X_2^t X_3^t$.

Note that a cut-join operation combines one cut and one join into a single operation, and a double-cut-join operation performs two cuts and two joins in one operation. See Figure 2 [1] for an illustration of these operations.

Several key models are based on these operations. The *Double Cut-and-Join (DCJ)* model was initially introduced by Yancopoulos, Attie, & Friedberg [20] and permits all four operations. Later, Feijao & Meidanis [6] introduced the *Single Cut-or-Join (SCoJ)* model, which only allows operations (i) and (ii). Alternatively, the *Single Cut-and-Join (SCaJ)* model [2] allows operations (i)-(iii), but not operation (iv). In this paper, we consider the Single Cut-and-Join model.

▶ **Definition 4.** *For any two genomes G_1 and G_2 with the same set of edge labels, there is a sequences of Single Cut-and-Join operations that transforms G_1 into G_2. Such a sequence is called a* scenario*. The minimum possible length of such a scenario is called the* distance *and is denoted $d(G_1, G_2)$. An operation on a genome G_1 that (strictly) decreases the distance to genome G_2 is called a* sorting operation *for G_1 and G_2. A scenario requiring $d(G_1, G_2)$ operations to transform G_1 into G_2 is called a* most parsimonious scenario *or* sorting scenario*. When G_2 is understood, we refer to the action of transforming G_1 into G_2 using the minimum number of operations as* sorting G_1*. The number of most parsimonious scenarios transforming G_1 into G_2 is denoted $\#MPS(G_1, G_2)$.*

We now turn to defining the key algorithmic problem that we will consider in this paper.

▶ **Definition 5.** *Let G_1 and G_2 be genomes. The* Distance *problem asks to compute $d(G_1, G_2)$. The* Pairwise Rearrangement *problem asks to compute $\#MPS(G_1, G_2)$.*

To investigate Pairwise Rearrangement, we begin by introducing the adjacency graph.

▶ **Definition 6.** *Given two genomes G_1 and G_2 with the same set of edge labels, the* adjacency graph *$A(G_1, G_2)$ is a bipartite undirected multigraph $(V_1 \dot{\cup} V_2, E)$ where the vertices in V_i are the adjacencies and telomeres in G_i and for any $X \in V_1$ and $Y \in V_2$, the number of edges between X and Y is $|X \cap Y|$.*

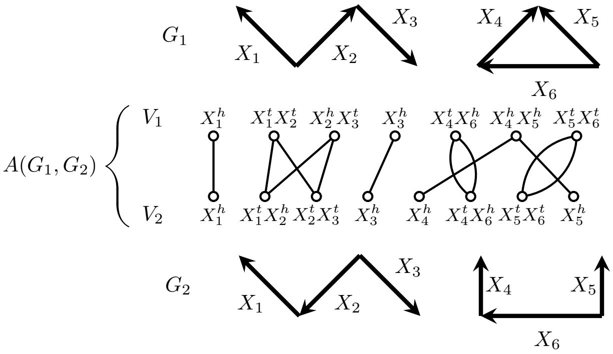


Figure 3 (Taken from [1].) The adjacency graph $A(G_1, G_2)$ is shown in the middle, for genomes G_1 and G_2 shown above and below, respectively.

Note that each vertex in an adjacency graph $A(G_1, G_2)$ must have either degree 1 or 2 (corresponding, respectively, to telomeres and adjacencies in the original genome), and so $A(G_1, G_2)$ is composed entirely of disjoint cycles and paths. Note also that every operation on G_1 corresponds to an operation on V_1 in $A(G_1, G_2)$. For example, in Figure 3 the cut operation on G_1 which separates adjacency $X_4^hX_5^h$ into telomeres X_4^h and X_5^h equates to separating the corresponding vertex $X_4^hX_5^h$ in V_1 into two vertices X_4^h and X_5^h, thus splitting the path of length 2 in $A(G_1, G_2)$ into two disjoint paths of length 1. In a similar fashion, a join operation on G_1 corresponds to combining two vertices a, b in V_1 into a single vertex ab, and a cut-join operation on G_1 corresponds to replacing vertices ab, c in V_1 with vertices ac, b. Whether or not an operation on $A(G_1, G_2)$ corresponds to a sorting operation on G_1 – that is, whether it decreases the distance to G_2 or not – depends highly on the structure of the components acted on. To better describe such sorting operations, we adopt the following classification of components in $A(G_1, G_2)$ and notion of their size:

▶ **Definition 7.** *The possible connected components of $A(G_1, G_2)$ are classified as follows:*

- *A W-*shaped component *is an even path with its two endpoints in V_1.*
- *An M-*shaped component *is an even path with its two endpoints in V_2.*
- *An N-*shaped component *is an odd path.*
- *A* crown *is an even cycle.*

▶ **Definition 8.** *The* size *of a component B in $A(G_1, G_2)$ is defined to be $\lfloor |E(B)|/2 \rfloor$. We refer to an N-shaped component of size 0 (a single edge) as a* trivial path*, and a crown of size 1 (a 2-cycle) as a* trivial crown*.*

The language "trivial" is motivated by the fact that such components indicate where G_1 and G_2 already agree, and hence no sorting operations are required on vertices belonging to trivial components (see, e.g., the trivial components in Figure 3). Indeed, a sorting scenario can be viewed as a minimal length sequence of operations which produces an adjacency graph consisting of only trivial components.

▶ **Observation 9** ([1, Observation 2.7])**.** *In the SCaJ model, a case analysis of all operations yields precisely these as the only sorting operations on $A(G_1, G_2)$:*

(a) *A cut-join operation on a nontrivial N-shaped component, producing an N-shaped component and a trivial crown*

(b) *A cut-join operation on a W-shaped component of size at least 2, producing a trivial crown and a W-shaped component*

(c) *A join operation on a W-shaped component of size 1, producing a trivial crown*

(d) *A cut operation on an M-shaped component, producing two N-shaped components*

(e) *A cut operation on a nontrivial crown, producing a W-shaped component*

(f) *A cut-join operation on an M-shaped component and a W-shaped component, where an adjacency in the M-shaped component is cut and joined to a telomere in the W-shaped component, producing two N-shaped components*

(g) *A cut-join operation on a nontrivial crown and an N-shaped component, where an adjacency in the crown is cut and joined to the telomere in the N-shaped component, producing an N-shaped component*

(h) *A cut-join operation on a nontrivial crown and a W-shaped component, where an adjacency from the crown is cut and joined with a telomere from the W-shaped component, producing a W-shaped component*

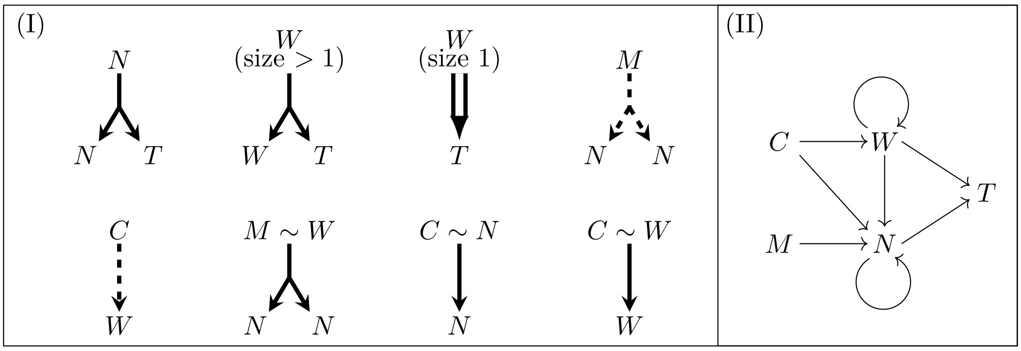


Figure 4 This figure depicts the operations described in Observation 9, where the arrows point from the original component type(s) to the component type(s) produced by the three operations allowed in the SCaJ model: cut, join, and cut-join. The eight diagrams in (I) show each of the sorting operations (a)-(h) where bold single arrows represent cut-join operations, dashed arrows represent cut operations, and the double arrow represents the join operation. Diagram (II) summarizes which components can be produced. Note that all operations will only result in W-shaped components, N-shaped components, and/or trivial crowns. Here, T denotes a trivial crown and C denotes a non-trivial crown.

Note that (a) - (e) are sorting operations on G_1 that operate on only one component in the adjacency graph, though they may produce two different components. On the other hand, (f) - (h) are sorting operations on G_1 that operate on two separate components in the adjacency graph. See Figure 4 for a visualization of each sorting operation.

Using these sorting operations on $A(G_1, G_2)$, the distance between two genomes G_1 and G_2 for the SCaJ model is given by

$$d(G_1, G_2) = n - \frac{\#N}{2} - \#T + \#C \tag{1}$$

where n is the number of edges in G_1 (equivalently, one half of the number of edges in $A(G_1, G_2)$), $\#N$ is the number of N-shaped components, $\#T$ is the number of trivial crowns, and $\#C$ is the number of nontrivial crowns [2].

Let $\mathcal{B}$ be the set of all components of $A(G_1, G_2)$ and let $\mathcal{B}'$ be a subset of $\mathcal{B}$. Define

$$d(\mathcal{B}') := \left(\sum_{B \in \mathcal{B}'} \text{size}(B) \right) - \#T_{\mathcal{B}'} + \#C_{\mathcal{B}'} \tag{2}$$

where $\#T_{\mathcal{B}'}$ and $\#C_{\mathcal{B}'}$ are the number of trivial crowns and nontrivial crowns in $\mathcal{B}'$, respectively. The quantity $d(\mathcal{B}')$ is the minimum number of operations needed to transform all components of $\mathcal{B}'$ into trivial components, with no operation acting on a component not belonging to $\mathcal{B}'$. Note that $d(\mathcal{B}) = d(G_1, G_2)$, as the $\frac{\#N}{2}$ term is absorbed into the summation of the sizes of all components.

▶ **Definition 10** ([1, Definition 2.8]). *Let A and B be components of an adjacency graph, and consider a particular sorting scenario. We say $A \sim B$ if either $A = B$ or there is a cut-join operation in the scenario where an extremity a from A and an extremity b from B are joined into an adjacency. The transitive closure of $\sim$ is an equivalence relation which we call* sort together*. We will be particularly interested in subsets of the equivalence classes of "sort together." We abuse terminology by referring to such a subset as a set that* sorts together*.*

Note that if two components A, B in $A(G_1, G_2)$ satisfy $A \sim B$, the cut-join operation does not need to occur immediately. For example, two nontrivial crowns C_1 and C_2 can satisfy $C_1 \sim C_2$ by first cutting C_1 to produce a W-shaped component, then operation (b) can be applied multiple times before operation (h) sorts C_2 and the remaining W-shaped component together; see Figure 4.

We will now recall additional notation from [1] that we will use in this paper. For a subset $\mathcal{B}'$ of $\mathcal{B}$, define $\#MPS(\mathcal{B}')$ as the number of sequences with $d(\mathcal{B}')$ operations in which the components of $\mathcal{B}'$ are transformed into trivial components with no operation acting on a component not belonging to $\mathcal{B}'$. Note that $\#\text{MPS}(\mathcal{B})$ is the number of most parsimonious scenarios transforming G_1 into G_2. Let $\#\text{ST}(\mathcal{B}')$ denote the number of sequences with $d(\mathcal{B}')$ operations in which the components of $\mathcal{B}'$ sort together and are transformed into trivial components with no operation acting on a component not belonging to $\mathcal{B}'$.

Note that if $\mathcal{B}'$ and $\mathcal{B}''$ are two subsets of $\mathcal{B}$ that have all the same component types with all the same sizes, then $\#MPS(\mathcal{B}') = \#MPS(\mathcal{B}'')$ and $\#\text{ST}(\mathcal{B}') = \#\text{ST}(\mathcal{B}'')$. Going forward, we will often care about values of $\#$MPS and $\#$ST only in the context of their component types and sizes. Suppose we are given multisets $\mathcal{C}$, $\mathcal{M}$, $\mathcal{W}$, and $\mathcal{N}$ of nonnegative integers with every element of $\mathcal{C}$ at least 2 and every element of $\mathcal{M}$ and $\mathcal{W}$ at least 1. We define $\#\text{MPS}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$ to equal $\#\text{MPS}(\mathcal{B}')$ for any set of components $\mathcal{B}'$ with $|\mathcal{C}|$ nontrivial crowns of sizes in $\mathcal{C}$, $|\mathcal{M}|$ M-shaped components of sizes in $\mathcal{M}$, $|\mathcal{W}|$ W-shaped components of sizes in $\mathcal{W}$, and $|\mathcal{N}|$ N-shaped components of sizes in $\mathcal{N}$. We define $\#\text{ST}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$ similarly.

3 Combinatorics of Genome Rearrangement

In this section, we will prove Theorem 1. Our strategy will be to build a lookup table for dynamic programming. In particular, our technique relies crucially on the following lemma.

▶ **Lemma 11** ([1, Lemma A.4]). *Let $\mathcal{B}'$ be a subset of components of an adjacency graph, and let $\Pi(\mathcal{B}')$ denote the set of all partitions of $\mathcal{B}'$. We have*

$$\#MPS(\mathcal{B}') = \sum_{\pi \in \Pi(\mathcal{B}')} \binom{d(\mathcal{B}')}{d(\pi_1), d(\pi_2), \ldots, d(\pi_{p(\pi)})} \prod_{i=1}^{p(\pi)} \#ST(\pi_i),$$

where $\pi = \{\pi_1, \pi_2, \ldots, \pi_{p(\pi)}\}$.

We will utilize Lemma 11 in the following manner. Fix an entry in the lookup table, and let $\mathcal{B}'$ denote the set of components being considered at said entry. Now fix a partition $\pi \in \Pi(\mathcal{B}')$. In order to compute $\#\mathrm{MPS}(\mathcal{B}')$, we will proceed as follows. For each $i \in [p(\pi)]$, we first check if $\#\mathrm{ST}(\pi_i) = 0$. This step is computable in polynomial-time by checking whether there exists a permissible sorting operation (see Observation 9). If $\#\mathrm{ST}(\pi_i) \neq 0$, then we access the entry in the lookup table for $\#\mathrm{ST}(\pi_i)$. This will allow us to compute

$$\binom{d(\mathcal{B}')}{d(\pi_1), d(\pi_2), \ldots, d(\pi_{p(\pi)})} \prod_{i=1}^{p(\pi)} \#\mathrm{ST}(\pi_i).$$

We will show later that as the number of nontrivial crowns in the adjacency graph is bounded (by assumption), there are only a polynomial number of partitions $\pi = (\pi_1, \ldots, \pi_{p(\pi)}) \in \Pi(\mathcal{B}')$ such that $\#\mathrm{ST}(\pi_i) \neq 0$ for all $i \in [p(\pi)]$.

We will now investigate how to compute $\#\mathrm{ST}(\pi_i)$, which requires studying which sets of components can and cannot sort together.

▶ **Proposition 12.** *Given a set $\mathcal{B}'$ of components of some adjacency graph G, $\#\mathrm{ST}(\mathcal{B}') = 0$ if $\mathcal{B}'$ has any of the following properties:*

1. *$\mathcal{B}'$ contains at least two components, at least one of which is a trivial crown.*
2. *$\mathcal{B}'$ contains more than one W-shaped component.*
3. *$\mathcal{B}'$ contains more than one M-shaped component.*
4. *$\mathcal{B}'$ contains more than one path and at least one N-shaped component.*

Proof. We refer to Observation 9 and Figure 4 for the permissible sorting operations, from which the proof essentially follows. We provide full details in the proof of Proposition 3.2 in the full version. ◀

If $\mathcal{B}'$ consists of only one trivial crown, then $\#\mathrm{ST}(\mathcal{B}') = 1$. Otherwise there are five possibilities for when $\#\mathrm{ST}(\mathcal{B}') \neq 0$. Below we list each case and define a function along with each that we will use to simplify $\#\mathrm{ST}(\mathcal{C}, M, W, N)$. In what follows, $\mathcal{Z}_{\geq 2}$ denotes the set of finite multisets of integers in which each integer is at least 2. Also, for this paper we take $\mathbb{N}$ to include 0. We will now list our cases:

- The components are a single N-shaped component and zero or more nontrivial crowns. Define $\#\mathrm{ST}_N : \mathcal{Z}_{\geq 2} \times \mathbb{N} \to \mathbb{N}$ as $\#\mathrm{ST}_N(\mathcal{C}, \eta) = \#\mathrm{ST}(\mathcal{C}, \emptyset, \emptyset, \{\eta\})$.
- The components are a single W-shaped component and zero or more nontrivial crowns. Define $\#\mathrm{ST}_W : \mathcal{Z}_{\geq 2} \times \mathbb{Z}^+ \to \mathbb{N}$ as $\#\mathrm{ST}_W(\mathcal{C}, w) = \#\mathrm{ST}(\mathcal{C}, \emptyset, \{w\}, \emptyset)$.
- The components are one or more nontrivial crowns. Define $\#\mathrm{ST}_C : \mathcal{Z}_{\geq 2} - \{\emptyset\} \to \mathbb{N}$ as $\#\mathrm{ST}_C(\mathcal{C}) = \#\mathrm{ST}(\mathcal{C}, \emptyset, \emptyset, \emptyset)$.
- The components are a single M-shaped component and zero or more nontrivial crowns. Define $\#\mathrm{ST}_M : \mathcal{Z}_{\geq 2} \times \mathbb{Z}^+ \to \mathbb{N}$ as $\#\mathrm{ST}_M(\mathcal{C}, m) = \#\mathrm{ST}(\mathcal{C}, \{m\}, \emptyset, \emptyset)$.
- The components are a single M-shaped component, a single W-shaped component, and zero or more nontrivial crowns. Define $\#\mathrm{ST}_{MW} : \mathcal{Z}_{\geq 2} \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{N}$ as $\#\mathrm{ST}_{MW}(\mathcal{C}, m, w) = \#\mathrm{ST}(\mathcal{C}, \{m\}, \{w\}, \emptyset)$.

We recall the following lemma, which allows us to compute the number of sorting scenarios for a single component.

▶ **Lemma 13** ([1, Lemma 2.9]).

- *For all $\eta \in \mathbb{N}$, $\#\mathrm{ST}_N(\emptyset, \eta) = 1$.*
- *For all $w \in \mathbb{Z}^+$, $\#\mathrm{ST}_W(\emptyset, w) = 2^{w-1}$.*
- *For all $m \in \mathbb{Z}^+$, $\#\mathrm{ST}_M(\emptyset, m) = 2^{m-1}$.*
- *For all $c \in \mathbb{Z}_{\geq 2}$, $\#\mathrm{ST}_C(\{c\}) = c \cdot 2^{c-1}$.*

Our goal now is to enumerate the number of sorting scenarios in each of these cases. We first provide a recurrence relation for $\#\mathrm{ST}_N(\mathcal{C},\eta)$.

▶ **Proposition 14.** *We have the following recurrence relations for* $\#\mathrm{ST}_N$*:*

$$\#\mathrm{ST}_N(\mathcal{C},\eta) = \begin{cases} 1 & : \mathcal{C} = \emptyset, \eta = 0, \\ \sum_{c\in\mathcal{C}} \left(2c \cdot \#\mathrm{ST}_N(\mathcal{C}-\{c\}, c)\right) & : \mathcal{C} \neq \emptyset, \eta = 0, \\ \#\mathrm{ST}_N(\mathcal{C},\eta-1) + \sum_{c\in\mathcal{C}} \left(2c \cdot \#\mathrm{ST}_N(\mathcal{C}-\{c\}, c+\eta)\right) & : \textit{otherwise.} \end{cases}$$

Proof. First, $\#\mathrm{ST}_N(\emptyset,0)$ is the case of a single path of size 0 (a single edge). This represents that the given telomere has already been sorted, so there is only one way to sort this component (do nothing). On the other hand, when sorting a collection of nontrivial crowns together with an N-shaped component, the first operation either consists of applying a cut-join on the N-shaped component alone if it has size greater than 0 (1 way to do this), or applying a cut-join of a nontrivial crown to the N-shaped component. If the crown being operated on has size c, there are c possible places to cut it, and there are 2 possible ways to join it to to the N-shaped component (either telomere of the newly cut crown). The result of this operation is one fewer crown and an increase in the size of the N-shaped component by c. These considerations lead to the recursive formulas for $\#\mathrm{ST}_N(\mathcal{C},\eta)$. ◀

We next provide a recurrence relation for $\#\mathrm{ST}_W(\mathcal{C},w)$.

▶ **Proposition 15.** *We have the following recurrence relations for* $\#\mathrm{ST}_W$*:*

$$\#\mathrm{ST}_W(\mathcal{C},w) = \begin{cases} 1 & : \mathcal{C} = \emptyset, w = 1, \\ \sum_{c\in\mathcal{C}} \left(4c \cdot \#\mathrm{ST}_W(\mathcal{C}-\{c\}, c+1)\right) & : \mathcal{C} \neq \emptyset, w = 1, \\ 2 \cdot \#\mathrm{ST}_W(\mathcal{C},w-1) + \sum_{c\in\mathcal{C}} \left(4c \cdot \#\mathrm{ST}_W(\mathcal{C}-\{c\}, c+w)\right) & : \textit{otherwise.} \end{cases}$$

Proof. First, $\#\mathrm{ST}_W(\emptyset,1)$ is the case of a single W-shaped component of size 1. To sort such a path, join the telomeres in the top genome. So, there is only one way to sort this component. On the other hand, when sorting a collection of nontrivial crowns together with a W-shaped component, the first operation either consists of a cut-join of one end of the W-shaped component if it has size greater than 1 (2 ways to do this) or a cut-join of a nontrivial crown to the W-shaped component. If the crown being operated on has size c, there are c possible places to cut it, and there are 4 possible ways to join it to to the W-shaped component (either telomere of the newly cut crown could join with either telomere of the W-shaped component). The result of this operation is one fewer nontrivial crown and an increase in the size of the W-shaped component by c. These considerations lead to the recursive formulas for $\#\mathrm{ST}_W(\mathcal{C},w)$. ◀

We next provide an expression for $\#\mathrm{ST}_C(\mathcal{C})$.

▶ **Proposition 16.** *We have the following expression for* $\#\mathrm{ST}_C$*:*

$$\#\mathrm{ST}_C(\mathcal{C}) = \sum_{c\in\mathcal{C}} \left(c \cdot \#\mathrm{ST}_W(\mathcal{C} - \{c\}, c)\right).$$

Note that $\mathcal{C} = \emptyset$ is not in the domain of $\#\mathrm{ST}_C$, and so this expression is well-defined.

Proof. When a collection of crowns sorts together, the first operation must be a cut. If the crown being operated on has size c, there are c possible places to cut it. The result of this operation is one fewer crown and a W-shaped component of size c. These considerations lead to the expression for $\#\mathrm{ST}_C(\mathcal{C})$. ◀

▶ Remark 17. A closed-form expression for $\#\mathrm{ST}_C(\mathcal{C})$ was previously established in [1, Corollary 3.2].

We have already discussed $\#\mathrm{ST}_W(\mathcal{C}, w)$. Below we establish tools to show $\#\mathrm{ST}_M(\mathcal{C}, w)$ is the same as $\#\mathrm{ST}_W(\mathcal{C}, w)$. We have thus far considered $A(G_1, G_2)$, where operations act on V_1. We can also consider $A(G_2, G_1)$, which is the same as $A(G_1, G_2)$, but we now operate on V_2. Every component of $A(G_1, G_2)$ corresponds to a component in $A(G_2, G_1)$ (on the same vertices), though component types are not necessarily the same. For example, an M-shaped component A_{12} in $A(G_1, G_2)$ corresponds to a W-shaped component A_{21} in $A(G_2, G_1)$.

▶ **Definition 18.** *For an operation α, define the reverse operation α^{rev} as follows. If α is a cut at adjacency ab, then α^{rev} is a join of a and b. Similarly, the reverse of a join is a cut. Further, if α is a cut-join ab, c to ac, b, then α^{rev} is a cut-join ac, b to ab, c. For a sequence σ of operations $\sigma_1, \ldots, \sigma_k$, let the reverse sequence σ^{rev} be $\sigma_k^{rev}, \ldots, \sigma_1^{rev}$.*

Observe that reversing an operation does not change the extremities operated on. We now show that reversal preserves the sort together relation.

▶ **Proposition 19.** *For a sorting scenario σ and components A_{12} and B_{12} in $A(G_1, G_2)$, suppose we have that $A_{12} \sim B_{12}$. Then, for σ^{rev} and corresponding components A_{21} and B_{21} in $A(G_2, G_1)$, we have $A_{21} \sim B_{21}$.*

Proof. Let σ be a sorting scenario for $A(G_1, G_2)$ that consists of a sequence of operations $\sigma_1, \ldots, \sigma_k$. Suppose further that σ sorts two components A_{12} and B_{12} together such that $A_{12} \sim B_{12}$. Since σ transforms G_1 into G_2, the sequence σ^{rev} transforms G_2 into G_1. Thus, σ^{rev} is a sorting scenario for $A(G_2, G_1)$.

Let A_{21} and B_{21} denote the two components in $A(G_2, G_1)$ that correspond to A_{12} and B_{12}. We will show that $A_{21} \sim B_{21}$ with respect to σ^{rev}. Since $A_{12} \sim B_{12}$ with respect to σ, there is some i such that operation σ_i is a cut-join operation which joins an extremity a of A_{12} with an extremity b of B_{12}. Moreover, σ_i^{rev} cuts the adjacency ab. Since a and b come from different components, they must have been joined through operation σ_j^{rev} for some $j > i$ to create adjacency ab (which is then cut by σ_i^{rev}). Thus, $A_{12} \sim B_{12}$ in $A(G_2, G_1)$ with respect to σ^{rev}, as required. ◀

▶ **Corollary 20.** *Let $\mathcal{B}'_{12}$ be a subset of the components of $A(G_1, G_2)$, and let $\mathcal{B}'_{21}$ be the corresponding subset of the components of $A(G_2, G_1)$. Then $\#\mathrm{ST}(\mathcal{B}'_{12}) = \#\mathrm{ST}(\mathcal{B}'_{21})$. Consequently, $\#\mathrm{ST}_M(\mathcal{C}, m) = \#\mathrm{ST}_W(\mathcal{C}, m)$, and $\#\mathrm{ST}_{MW}(\mathcal{C}, m, w) = \#\mathrm{ST}_{MW}(\mathcal{C}, w, m)$.*

Proof. Given a subset $\mathcal{B}'_{12}$ of components of $A(G_1, G_2)$, let $\mathcal{B}'_{21}$ be the set of corresponding components in $A(G_2, G_1)$. Now, let Σ_{12} be the set of scenarios that sort the components of $\mathcal{B}'_{12}$ together, and let Σ_{21} be the set of scenarios that sort the components of $\mathcal{B}'_{21}$ together. By definition, $|\Sigma_{12}| = \#\mathrm{ST}(\mathcal{B}'_{12})$ and $|\Sigma_{21}| = \#\mathrm{ST}(\mathcal{B}'_{21})$. Next, let Σ_{12}^{rev} and Σ_{21}^{rev} be the sets of reversals of the scenarios in Σ_{12} and Σ_{21} respectively. By Proposition 19, $\Sigma_{12}^{rev} \subseteq \Sigma_{21}$. Since reversal is an involution, we also obtain that $\Sigma_{21}^{rev} \subseteq \Sigma_{12}$. Since $|\Sigma_{12}| = |\Sigma_{12}^{rev}|$ and $|\Sigma_{21}| = |\Sigma_{21}^{rev}|$, this implies that $|\Sigma_{12}| \leq |\Sigma_{21}|$ and $|\Sigma_{21}| \leq |\Sigma_{12}|$. Together, these imply that $|\Sigma_{12}| = |\Sigma_{21}|$, meaning $\#\mathrm{ST}(\mathcal{B}'_{12}) = \#\mathrm{ST}(\mathcal{B}'_{21})$, as required.

We note that an M-shaped component in $A(G_1, G_2)$ is a W-shaped component in $A(G_2, G_1)$. Thus, we obtain that $\#\mathrm{ST}_M(\mathcal{C}, m) = \#\mathrm{ST}_W(\mathcal{C}, m)$ and $\#\mathrm{ST}_{MW}(\mathcal{C}, m, w) = \#\mathrm{ST}_{MW}(\mathcal{C}, w, m)$, as desired. ◀

By Corollary 20, $\#\mathrm{ST}_M(\mathcal{C}, m) = \#\mathrm{ST}_W(\mathcal{C}, m)$. But, here we present a different expression for $\#\mathrm{ST}_M(\mathcal{C}, m)$, which will be useful later when we examine $\#\mathrm{ST}_{MW}$. To simplify notation, we introduce a new definition. Denote:

$$\mathrm{sum}(\mathcal{C}) := \sum_{c \in C} c.$$

▶ **Lemma 21.** *We have the following expression for* $\#\mathrm{ST}_M$*:*

$$\#\mathrm{ST}_M(\mathcal{C},m) = \sum_{\eta=0}^{m-1}\sum_{\mathcal{C}'\subseteq\mathcal{C}}\binom{\mathrm{sum}(\mathcal{C})+|\mathcal{C}|+m-1}{\mathrm{sum}(\mathcal{C}')+|\mathcal{C}'|+\eta}\#\mathrm{ST}_N(\mathcal{C}',\eta)\cdot\#\mathrm{ST}_N(\mathcal{C}-\mathcal{C}',m-1-\eta)$$
$$+\sum_{c\in\mathcal{C}}\left(c\cdot\#\mathrm{ST}_{MW}(\mathcal{C}-\{c\},m,c)\right).$$

Proof. See the proof of Lemma 3.11 in the full version. ◀

It now remains to define a recurrence relation for $\#\mathrm{ST}_{MW}(\mathcal{C},m,w)$. We begin with the following lemma.

▶ **Lemma 22.** *We have the following recurrence relations for* $\#\mathrm{ST}_{MW}$*:*

$$\#\mathrm{ST}_{MW}(\mathcal{C},m,w) = \begin{cases} f(\mathcal{C},m,w) & : w=1, \\ 2\cdot\#\mathrm{ST}_{MW}(\mathcal{C},m,w-1)+f(\mathcal{C},m,w) & : w>1. \end{cases}$$

where

$$f(\mathcal{C},m,w) = \sum_{c\in\mathcal{C}}\left(4c\cdot\#\mathrm{ST}_{MW}(\mathcal{C}-\{c\},m,c+w)\right)$$
$$+\sum_{\eta=0}^{m-1}4\cdot\sum_{\mathcal{C}'\subseteq\mathcal{C}}\binom{\mathrm{sum}(\mathcal{C})+|\mathcal{C}|+m+w-1}{\mathrm{sum}(\mathcal{C}')+|\mathcal{C}'|+\eta}\#\mathrm{ST}_N(\mathcal{C}',\eta)\cdot\#\mathrm{ST}_N(\mathcal{C}-\mathcal{C}',m+w-\eta-1).$$

Proof. See the proof of Lemma 3.12 in the full version. ◀

A priori, to use Lemma 22, we have to track the case when an M-shaped component sorting with a W-shaped component results in two N-shaped components. This requires $O(2^{|\mathcal{C}|})$ steps to compute the double-summation in $f(\mathcal{C},m,w)$ from Lemma 22. The next proposition allows us to both avoid these steps and simplify our algorithm.

▶ **Proposition 23.** *We have the following recurrence relation for* $\#\mathrm{ST}_{MW}$*:*

$$\#\mathrm{ST}_{MW}(\mathcal{C},m,w) = \begin{cases} g(\mathcal{C},m,w) & : m=1,\ w=1, \\ \#\mathrm{ST}_{MW}(\mathcal{C},m-1,w)+g(\mathcal{C},m,w) & : m>1,\ w=1, \\ \#\mathrm{ST}_{MW}(\mathcal{C},m,w-1)+g(\mathcal{C},m,w) & : m=1,\ w>1, \\ \#\mathrm{ST}_{MW}(\mathcal{C},m-1,w) & : \textit{otherwise} \\ \quad+\#\mathrm{ST}_{MW}(\mathcal{C},m,w-1)+g(\mathcal{C},m,w). & \end{cases}$$

where

$$g(\mathcal{C},m,w) = 2\cdot\#\mathrm{ST}_W(\mathcal{C},m+w)+\sum_{c\in\mathcal{C}}2c\Big(\#\mathrm{ST}_{MW}(\mathcal{C}-\{c\},m+c,w)$$
$$+\#\mathrm{ST}_{MW}(\mathcal{C}-\{c\},m,w+c)-\#\mathrm{ST}_{MW}(\mathcal{C}-\{c\},m+w,c)\Big).$$

Proof. See the proof of Proposition 3.13 in the full version. ◀

4 Fixed-Parameter Tractability of Pairwise Rearrangement

In this section, we will use the recurrences we have found to establish our main result.

▶ **Theorem 1.** *In the Single Cut-and-Join model, PAIRWISE REARRANGEMENT is fixed-parameter tractable, with respect to the number of nontrivial components in the adjacency graph.*

Proof. Given two genomes with n edges each, the adjacency graph is easily constructed in polynomial time. Let $\mathcal{B}$ be the set of components for this adjacency graph, and let k denote the number of nontrivial components in $\mathcal{B}$. It suffices to show that the number of scenarios $\#\text{MPS}(\mathcal{B})$ can be determined in time $O(k \cdot 2^k \cdot B_k + k \cdot 2^k \cdot n^2)$, where B_k is the k^{th} Bell number.

Let $\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N}$ be multisets of nonnegative integers, denoting the sizes of the nontrivial crowns, M-shaped components, W-shaped components, and N-shaped components, respectively, of $\mathcal{B}$. Thus, we assume that every element of $\mathcal{C}$ is at least 2, and every element of $\mathcal{M}$ and $\mathcal{W}$ is at least 1.

We will proceed in two stages. In the first stage, we will build up a table of values for $\#\text{ST}_N(\mathcal{A}, \eta)$, $\#\text{ST}_W(\mathcal{A}, w)$, $\#\text{ST}_C(\mathcal{A})$, and $\#\text{ST}_{MW}(\mathcal{A}, m, w)$ with $\mathcal{A}$ any sub-multiset of $\mathcal{C}$ and $\eta \geq 0$, $w \geq 1$, and $m \geq 1$. Then in the second stage, we will use this final table of values along with Lemma 11 to compute $\#\text{MPS}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$.

Stage 1. When $\eta \geq 0$ we have by Lemma 13 that $\#\text{ST}_N(\emptyset, \eta) = 1$. Similarly, for $w \geq 1$, we have that $\#\text{ST}_W(\emptyset, w) = 2^{w-1}$, which is computable in time $O(n)$.

Now fix $m \in [n], w \in [n]$ with $m + w = i$, and suppose that for each $1 \leq m' \leq m$ and each $1 \leq w' \leq w$ with $m' + w' < i$, that we have computed $\#\text{ST}_{MW}(\emptyset, m', w')$. Using these values, together with the fact that $\#\text{ST}_W(\emptyset, w) = 2^{w-1}$, we may apply Proposition 23 to compute $\#\text{ST}_{MW}(\emptyset, m, w)$. There are at most n^2 pairs $(m', w') \in [n] \times [n]$, and so this step takes time $O(n^2)$.

Let u be an integer $0 < u \leq |\mathcal{C}| \leq k$. First, let $0 < x \leq n$. Suppose we have a table of values of $\#\text{ST}_N(\mathcal{A}, \eta)$ and $\#\text{ST}_W(\mathcal{A}, w)$ for all sub-multisets $\mathcal{A}$ of $\mathcal{C}$ with $|\mathcal{A}| < u$ and $\eta, w \leq n$ and also for all sub-multisets $\mathcal{A}$ of $\mathcal{C}$ with $|\mathcal{A}| = u$ and $\eta, w < x$. Let $\mathcal{C}'$ be a sub-multiset of $\mathcal{C}$ with $|\mathcal{C}'| = u$. We now proceed to fill in some of our lookup table from the bottom-up, using the recurrences from the previous section.

- We may compute $\#\text{ST}_N(\mathcal{C}', x)$ using Proposition 14. There are at most kn cells in our lookup table, and so this step takes time $O(kn)$.
- We may compute $\#\text{ST}_W(\mathcal{C}', x)$ using Proposition 15. There are at most kn cells in our lookup table, and so this step takes time $O(kn)$.
- We now turn to computing $\#\text{ST}_C(\mathcal{C}')$. To do so, we apply Proposition 16 using the previously computed values of $\#\text{ST}_W(\mathcal{A}, w)$ for all $\mathcal{A}$ of size less than u and all $1 \leq w \leq n$. This step takes time $O(k)$.

Now, let x and y be nonnegative integers with $x + y \leq n$. Suppose we have a table of values of $\#\text{ST}_{MW}(\mathcal{A}, m, w)$ for all sub-multisets $\mathcal{A}$ of $\mathcal{C}$ satisfying one of the following:

- $|\mathcal{A}| < u$ for any nonnegative integers m and w with $m + w \leq n$, or
- $|\mathcal{A}| = u$ with $m + w < x + y$.

We now turn to computing $\#\text{ST}_{MW}(\mathcal{C}', x, y)$. To do so, we apply Proposition 23 using the previously computed values of $\#\text{ST}_W(\mathcal{A}, w)$ for all $1 \leq w \leq n$ and of $\#\text{ST}_{MW}(\mathcal{A}, m, w)$ for all sub-multisets $\mathcal{A}$ of $\mathcal{C}$ satisfying one of the conditions above. There are at most kn^2 many such cells in our lookup table, and so filling in these cells takes time $O(kn^2)$.

To summarize, our first stage involves building up a table of values for $\#\mathrm{ST}_N(\mathcal{A}, \eta)$, $\#\mathrm{ST}_W(\mathcal{A}, w)$, $\#\mathrm{ST}_C(\mathcal{A})$, and $\#\mathrm{ST}_{MW}(\mathcal{A}, m, w)$ with $\mathcal{A}$ any sub-multiset of $\mathcal{C}$ and η, w, and m having values as above. This first stage is computable in time $O(k \cdot 2^k \cdot n^2)$ by iterating through all sub-multisets of $\mathcal{C}$ in non-decreasing order of cardinality.

Stage 2. We will use the table of values computed in Stage 1, along with Lemma 11, to compute $\#\mathrm{MPS}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$ as follows. First, let $\mathcal{B}' \subseteq \mathcal{B}$ denote the set of all nontrivial components in $\mathcal{B}$, and let t denote the number of trivial N-shaped components in $\mathcal{B}$. As per Lemma 11, we consider each partition π of $\mathcal{B}$, and examine $\#\mathrm{ST}(\pi_i)$ for each part π_i of π. By Proposition 12, we can determine in time $O(k)$ if $\#\mathrm{ST}(\pi_i) = 0$. In such a case, the term of $\#\mathrm{MPS}(\mathcal{B})$ corresponding to π contributes 0, and so we may discard π. Call a partition π *permissible* if $\#\mathrm{ST}(\pi_i) > 0$ for all parts. Recall the parts π_i from Proposition 12 for which $\#\mathrm{ST}(\pi_i) = 0$. Note that the only part π_i with a combination of trivial and nontrivial components for which $\#\mathrm{ST}(\pi_i) > 0$ is the case in which π_i contains precisely nontrivial crowns and a single trivial N-shaped component. So the permissible partitions of $\mathcal{B}$ are precisely the partitions generated by the following procedure:

- Start with a permissible partition π' of $\mathcal{B}'$.
- For at most t parts π'_i of π' that each consist of only nontrivial crowns, add one of the t trivial N-shaped components from $\mathcal{B}$ to π'_i.
- Each of the remaining trivial N-shaped components and each trivial crown from $\mathcal{B}$ become a singleton part.

Let $\pi' = (\pi'_1, \ldots, \pi'_{p(\pi')})$ be a permissible partition of $\mathcal{B}'$. We say that a permissible partition $\pi = (\pi_1, \ldots, \pi_{p(\pi)})$ of $\mathcal{B}$ *extends* π' if there exists an injective function $f : [p(\pi')] \to [p(\pi)]$ such that $\pi'_i \subseteq \pi_{f(i)}$ for all $i \in [p(\pi')]$. Note that by Proposition 12, π'_i and $\pi_{f(i)}$ may only be different if π'_i contains only nontrivial crowns, and $\pi_{f(i)}$ contains a nontrivial N-shaped component.

Note that while $\mathcal{B}'$ has at most k components, $\mathcal{B}$ may in general not have bounded size. Fix a permissible partition π' of $\mathcal{B}'$. Our goal is to compute:

$$\sum_{\substack{\pi \in \Pi(\mathcal{B}) \\ \pi \text{ extends } \pi'}} \binom{d(\mathcal{B})}{d(\pi_1), d(\pi_2), \ldots, d(\pi_{p(\pi)})} \prod_{i=1}^{p(\pi)} \#\mathrm{ST}(\pi_i).$$

As above, let π' be a permissible partition of $\mathcal{B}'$, and let π be an extension of π'. Recall, if π_i consists of only one trivial component, then $\#\mathrm{ST}(\pi_i) = 1$. Let π_i be a part of π that contains only nontrivial crowns and a single trivial N-shaped component. Let π'_i be obtained from π_i by removing the trivial N-shaped component. We have by equation (2) that $d(\pi'_i) = d(\pi_i)$. Thus, if π extends π', we have that:

$$\binom{d(\mathcal{B})}{d(\pi_1), d(\pi_2), \ldots, d(\pi_{p(\pi)})} = \binom{d(\mathcal{B}')}{d(\pi'_1), d(\pi'_2), \ldots, d\left(\pi'_{p(\pi')}\right)}.$$

Proposition 12 and the above procedure for constructing permissible partitions yields precisely the following cases:

- **Case 1:** Suppose that π'_i consists of an N-shaped component and zero or more nontrivial crowns. In this case, we can use a known value of $\#\mathrm{ST}_N$ to count the number of ways to sort this part.
- **Case 2:** Suppose that π'_i consists of a W-shaped component and zero or more nontrivial crowns. In this case, we can use a known value of $\#\mathrm{ST}_W$ to count the number of ways to sort this part.

- **Case 3:** Suppose that π'_i consists of an M-shaped component and zero or more nontrivial crowns. In this case, we can also use a known value of $\#\text{ST}_M$ to count the number of ways to sort this part.
- **Case 4:** Suppose that π'_i consists of an M-shaped component, a W-shaped component, and zero or more nontrivial crowns. In this case, we can use a known value of $\#\text{ST}_{MW}$ to count the number of ways to sort this part.
- **Case 5:** If none of Cases 1-4 hold, then we necessarily have that π'_i consists of only nontrivial crowns. Since the presence of such parts implies that π' may not necessarily extend uniquely to a permissible partition of $\mathcal{B}$, we consider all such parts π'_i simultaneously in order to handle all such permissible extensions of π'. Let $\pi'_{i_1}, \ldots, \pi'_{i_\ell}$ be the parts of π' that contain only nontrivial crowns. By Proposition 12, we have that for $j \in [\ell]$, we can only add at most one trivial N-shaped component to π'_{i_j}. As the trivial N-shaped components and the parts of π' are both distinguishable, there are $P(t, v) = t!/(t-v)!$ ways to assign $v \leq t$ trivial N-shaped components to v parts drawn from $\pi'_{i_1}, \ldots, \pi'_{i_\ell}$. The remaining trivial N-shaped components each form their own singleton component in the corresponding partition π of $\mathcal{B}$ extending π'. If part π'_{i_j} ($j \in [\ell]$) has a trivial N-shaped component added, we can use a known value of $\#\text{ST}_N$ to count the number of ways to sort this part. If not, we can use a known value of $\#\text{ST}_C$ to count the number of ways to sort this part. The number of ways to sort each part is independent of the number of ways to assign the trivial N-shaped components. Thus, multiply the value returned from the lookup table by $P(t, v)$. Finally, using Lemma 11, we multiply together each of the $\#\text{ST}_N$ and $\#\text{ST}_C$ values that we accessed from the lookup table. This takes time $O(k)$ since there are $O(k)$ parts.

  We now iterate over all integers $0 \leq v \leq \min(t, \ell)$, where we consider all possible assignments of v trivial N-shaped components to the parts of π' (where the assignment is as described in the preceding paragraph). As each π'_{i_j} can only receive at most one trivial N-shaped component, we iterate over all v-subsets of $[\ell]$ to fully enumerate each case. As $\ell \leq k$, there are at most 2^k total subsets to consider across all values of v, and so computing the count in this case takes time $O(k \cdot 2^k)$.

In the first four cases, we only look at a single part at a time. Since there are $O(k)$ parts, Lemma 11 gives a complexity of $O(k)$ for each of these cases. In Case 5 we already account for Lemma 11, so we have a complexity of $O(k \cdot 2^k)$. Thus for all cases the runtime is at most $O(k \cdot 2^k)$ using the existing values in the lookup table to compute $\#\text{ST}(\pi_i)$. As we are summing over all possible partitions, we are evaluating at most the number of partitions of a set of cardinality k, which is the k^{th} Bell number B_k. This gives a complexity of $O(k \cdot 2^k \cdot B_k)$ for computing $\#\text{MPS}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$ from the pre-established lookup tables.

So, overall, we have a complexity of $O(k \cdot 2^k \cdot n^2)$ for our first stage and a complexity of $O(k \cdot 2^k \cdot B_k)$ for our second stage. Thus, $\#\text{MPS}(\mathcal{C}, \mathcal{M}, \mathcal{W}, \mathcal{N})$ is computable in time $O(k \cdot 2^k \cdot B_k + k \cdot 2^k \cdot n^2)$, as required. The result now follows. ◀

5 Conclusion

We investigated the computational complexity of the Pairwise Rearrangement problem in the Single Cut-and-Join model. In particular, while Pairwise Rearrangement was previously shown to be #P-complete under polynomial-time Turing reductions [1], we proved it is fixed-parameter tractable when parameterized by the number of components in the adjacency graph that are not trivial crowns (Theorem 1). In particular, our results show that the number of nontrivial components serves as a key barrier towards efficiently enumerating and sampling minimum length sorting scenarios. We conclude with some open questions.

▶ **Question 24.** In the Single Cut-and-Join model, does Pairwise Rearrangement belong to FPRAS?

▶ **Question 25.** In the Double Cut-and-Join model, is Pairwise Rearrangement #P-complete?

The work of Braga and Stoye [3] immediately yields a fixed-parameter tractable algorithm for Pairwise Rearrangement in the Double Cut-and-Join model. Their algorithm is considerably simpler than our work in this paper.

The computational complexity of the Pairwise Rearrangement problem in the Reversal model is a long-standing open question. In particular, it is believed that this problem is #P-complete. Furthermore, Pairwise Rearrangement has been resistant to efficient sampling; membership in FPRAS remains open. Thus, perhaps the lens of parameterized complexity might shed new light on this problem. The following question is natural, though might be hard.

▶ **Question 26.** In the Reversal model, is Pairwise Rearrangement fixed-parameter tractable by the number of components in the adjacency graph?

References

1 Lora Bailey, Heather Smith Blake, Garner Cochran, Nathan Fox, Michael Levet, Reem Mahmoud, Elizabeth Bailey Matson, Inne Singgih, Grace Stadnyk, Xinyi Wang, and Alexander Wiedemann. Complexity and enumeration in models of genome rearrangement. In Weili Wu and Guangmo Tong, editors, *Computing and Combinatorics*, pages 3–14, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-49190-0_1.

2 Anne Bergeron, Paul Medvedev, and Jens Stoye. Rearrangement models and single-cut operations. *Journal of computational biology : a journal of computational molecular cell biology*, 17:1213–25, September 2010. doi:10.1089/cmb.2010.0091.

3 Marília D. V. Braga and Jens Stoye. Counting all DCJ sorting scenarios. In Francesca D. Ciccarelli and István Miklós, editors, *Comparative Genomics*, pages 36–47, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-04744-2_4.

4 Aaron Darling, István Miklós, and Mark Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS genetics*, 4:e1000128, July 2008. doi:10.1371/journal.pgen.1000128.

5 Rick Durrett, Rasmus Nielsen, and Thomas York. Bayesian estimation of genomic distance. *Genetics*, 166:621–9, February 2004. doi:10.1534/genetics.166.1.621.

6 Pedro Feijão and Joao Meidanis. SCJ: A breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM transactions on computational biology and bioinformatics*, 8:1318–29, February 2011. doi:10.1109/TCBB.2011.34.

7 Andrew Holland and Don Cleveland. Chromoanagenesis and cancer: Mechanisms and consequences of localized, complex chromosomal rearrangements. *Nature medicine*, 18:1630–8, November 2012. doi:10.1038/nm.2988.

8 Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.

9 Bret Larget, Donald L. Simon, Joseph B. Kadane, and Deborah Sweet. A Bayesian Analysis of Metazoan Mitochondrial Genome Arrangements. *Molecular Biology and Evolution*, 22(3):486–495, November 2004. doi:10.1093/molbev/msi032.

10 Barbara McClintock. Chromosome organization and genic expression. In *Cold Spring Harbor symposia on quantitative biology*, volume 16, pages 13–47. Cold Spring Harbor Laboratory Press, 1951. doi:10.1101/sqb.1951.016.01.004.

11 István Miklós, Sándor Z. Kiss, and Eric Tannier. Counting and sampling SCJ small parsimony solutions. *Theor. Comput. Sci.*, 552:83–98, 2014. doi:10.1016/j.tcs.2014.07.027.

12 István Miklós and Heather Smith. Sampling and counting genome rearrangement scenarios. *BMC Bioinformatics*, 16:S6, October 2015. doi:10.1186/1471-2105-16-S14-S6.

13 István Miklós and Eric Tannier. Bayesian sampling of genomic rearrangement scenarios via double cut and join. *Bioinformatics*, 26(24):3012–3019, October 2010. doi:10.1093/bioinformatics/btq574.

14 István Miklós and Eric Tannier. Approximating the number of double cut-and-join scenarios. *Theoretical Computer Science*, 439:30–40, 2012. doi:10.1016/j.tcs.2012.03.006.

15 J. D. Palmer and L. A. Herbon. Plant mitochondrial dna evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28:87–97, December 1988. doi:10.1007/BF02143500.

16 A. H. Sturtevant. The linear arrangement of six sex-linked factors in drosophila, as shown by their mode of association. *Journal of Experimental Zoology*, 14(1):43–59, 1913. doi:10.1002/jez.1400140104.

17 A. H. Sturtevant. Genetic factors affecting the strength of linkage in drosophila. *Proceedings of the National Academy of Sciences of the United States of America*, 3(9):555–558, 1917. URL: http://www.jstor.org/stable/83776.

18 A. H. Sturtevant. Known and probably inverted sections of the autosomes of Drosophila melanogaster. *Carnegie Institution of Washington Publisher*, 421:1–27, 1931.

19 A H Sturtevant and E Novitski. The Homologies of the Chromosome Elements in the genus Drosophila. *Genetics*, 26(5):517–541, September 1941. doi:10.1093/genetics/26.5.517.

20 Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics (Oxford, England)*, 21:3340–6, September 2005. doi:10.1093/bioinformatics/bti535.

Succinct Data Structure for Chordal Graphs with Bounded Vertex Leafage

Girish Balakrishnan
Indian Institute of Technology Madras, Chennai, India

Sankardeep Chakraborty
University of Tokyo, Japan

N. S. Narayanaswamy
Indian Institute of Technology Madras, Chennai, India

Kunihiko Sadakane
University of Tokyo, Japan

Abstract

Chordal graphs is a well-studied large graph class that is also a strict super-class of path graphs. Munro and Wu (ISAAC 2018) have given an $(n^2/4+o(n^2))-$bit succinct representation for $n-$vertex unlabeled chordal graphs. A chordal graph $G=(V,E)$ is the intersection graph of sub-trees of a tree T. Based on this characterization, the two parameters of chordal graphs which we consider in this work are *leafage*, introduced by Lin, McKee and West (Discussiones Mathematicae Graph Theory 1998) and *vertex leafage*, introduced by Chaplick and Stacho (Discret. Appl. Math. 2014). Leafage is the minimum number of leaves in any possible tree T characterizing G. Let $L(u)$ denote the number of leaves of the sub-tree in T corresponding to $u \in V$ and $k = \max\limits_{u \in V} L(u)$. The smallest k for which there exists a tree T for G is called its vertex leafage.

In this work, we improve the worst-case information theoretic lower bound of Munro and Wu (ISAAC 2018) for $n-$vertex unlabeled chordal graphs when vertex leafage is bounded and leafage is unbounded. The class of unlabeled $k-$vertex leafage chordal graphs that consists of all chordal graphs with vertex leafage at most k and unbounded leafage, denoted $\mathcal{G}_k$, is introduced for the first time. For $k>0$ in $o(n^c), c>0$, we obtain a lower bound of $((k-1)n\log n - kn\log k - O(\log n))-$bits on the size of any data structure that encodes a graph in $\mathcal{G}_k$. Further, for every $k-$vertex leafage chordal graph G and $k>1$ in $o(n^c), c>0$, we present a $((k-1)n\log n + o(kn\log n))-$bit succinct data structure, constructed using the succinct data structure for path graphs with $(k-1)n$ vertices. Our data structure supports adjacency query in $O(k\log n)$ time and using additional $2n\log n$ bits, an $O(k^2 d_v \log n + \log^2 n)$ time neighbourhood query where d_v is degree of $v \in V$.

2012 ACM Subject Classification Information systems → Data structures

Keywords and phrases succinct data structure, chordal graphs, leafage, vertex leafage, path graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.4

Related Version *Full Version*: https://arxiv.org/abs/2402.03748

Funding *Sankardeep Chakraborty*: supported by MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant Number JPMXS0120319794.

1 Introduction

A data structure for a graph class $\mathcal{G}$ of graphs with $n-$vertices is succinct if it takes $(\log|\mathcal{G}| + o(\log|\mathcal{G}|))-$bits of space; here $|\mathcal{G}|$ denotes the number of $n-$vertex graphs in $\mathcal{G}$. A succinct representation for graph $G=(V,E)$ in $\mathcal{G}$ is expected to support the following queries for each pair of vertices $u, v \in V$:

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 4; pp. 4:1–4:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- `adjacency`(u, v): returns "YES" if and only if vertices u and v are adjacent in G.
- `neighborhood`(u): returns all the vertices in V that are adjacent to vertex u.
- `degree`(u): returns the number of vertices adjacent to vertex u.

The earliest work on space-efficient data structures for graph classes was by Itai and Rodeh [16] for vertex labeled planar graphs and by Jacobson [17] for class of static unlabeled trees and planar graphs. In recent years, succinct data structures for intersection graphs is getting lot of attention. A few of them are, Acan et al. [2] for interval graphs, Acan et al. [1] for families of intersection graphs on circle, Munro and Wu [22] for chordal graphs and Balakrishnan et al. [4] for path graphs. There are also other results on representation of chordal graphs, for instance, the space-efficient representation for chordal graphs by Markenzon et al. [19]. This shows that representation of chordal graphs are of interest to the research community.

A recent paper by Chakraborty and Jo [6] improve the information theoretic lower bound for interval graphs as given by Acan et al. [2] by bounding maximum degree. For bounded maximum degree Δ in $O(n^{\epsilon})$ where $1 < \epsilon < 1$, they give a lower bound of $(\frac{1}{6}n \log \Delta - O(n))-$bits and a $(n \log \Delta + O(n))-$bit space-efficient data structure. They also give a $((\chi - 1)n + o(\chi n))-$bit space-efficient data structure for interval graphs with bounded chromatic number χ for $\chi = o(\log n)$. In Balakrishnan et al. [3], such a parameterization has been applied to a larger graph class, namely, the class of graphs with $d-$dimensional $t-$representation where parameters d and t are bounded. A $((2td-1)n \log n + o(tdn \log n))-$bit succinct data structure for graphs with $d-$dimensional $t-$representation is presented in [3]. Special cases of this graph class are graphs with bounded boxicity with $t = 1$ and bounded interval number with $d = 1$. Chordal graphs is also a large graph class that is a generalization of interval graphs. A graph is a chordal graph if it contains only cycles of length at most three. Lin et al. [18] introduced the parameter leafage and Chaplick and Stacho [8] the parameter vertex leafage for chordal graphs. In this work, we define the $k-$vertex leafage chordal graphs that contains chordal graphs with bounded vertex leafage and unbounded leafage. When vertex leafage and leafage are equal to two we get interval graphs and when vertex leafage is two and leafage is unbounded we get path graphs. We present a data structure for $k-$vertex leafage chordal graphs using succinct data structure for path graphs and prove that it is succinct.

Convention. Throughout the rest of this paper, set of vertices and edges of a graph G will be denoted by $V(G)$ and $E(G)$, respectively.

Our Results. We present the following two theorems. The first theorem proves a lower bound on the class of $k-$vertex leafage graphs, denoted $\mathcal{G}_k$.

▶ **Theorem 1.** *For $k > 0$ in $o(n^c), c > 0, \log |\mathcal{G}_k| \geq (k-1)n \log n - kn \log k - O(\log n)$.*

The next theorem proves the existence of a matching data structure that supports adjacency query efficiently. For neighbour query we use additional $2n \log n$ bits.

▶ **Theorem 2.** *For $k > 1$ and in $o(n^c), c > 0$, a graph $G \in \mathcal{G}_k$ has a $(k-1)n \log n + o(kn \log n)$-bit succinct data structure that supports adjacency query in $O(k \log n)$ time and using additional $2n \log n$ bits the neighbourhood query for vertex v in $O(k^2 d_v \log n + \log^2 n)$ time where d_v is the degree of $v \in V(G)$.*

Our Main Ideas. The main objectives of this work are two fold.

1. Prove a worst-case information theoretic lower bound on the cardinality of chordal graphs with bounded vertex leafage by using a simple and constructive counting technique motivated by the method of partial coloring as used by Acan et al. [1].

2. Present a data structure for chordal graphs with bounded vertex leafage that uses the succinct data structure for path graphs given in Balakrishnan et al. [4] as a black box. Also, using the lower bound show that this data structure is succinct. In order to design the data structure, we use the characterisation of chordal graphs as intersection graph of sub-trees of a tree. The sub-trees of this tree are carefully decomposed into paths and the resulting paths along with the tree are stored using the succinct data structure for path graphs.

Organisation of the Paper. Section 2 gives details of the compact data structures we have used in the construction of the succinct data structure of $k-$vertex leafage chordal graphs along with the relevant characterisations of chordal and path graphs. It also formalizes and explains the method of partial coloring, first used in Acan et al. [1] to obtain the lower bound for $\mathcal{G}_k$. Section 3 defines the $k-$vertex leafage chordal graphs and also gives a lower bound on the cardinality of the class. The succinct data structure design is given in Section 4 and it contains in Section 4.1, the method to convert a $k-$vertex leafage chordal graph with n vertices to a path graph with $(k-1)n$ vertices. Section 5 concludes by giving motivation to extend this work by generalizing the parameters vertex leafage and leafage to general graphs.

2 Preliminaries

From Gavril [12] we have, the following characterisation of chordal graphs; see Golumbic [14] for more details on chordal graphs or otherwise called triangulated graphs.

▶ **Theorem 3.** *The graph G is a chordal graph if and only if there exists a clique tree T, such that for every $v \in V(G)$, the set of maximal cliques containing v form a sub-tree of T such that two vertices are adjacent if the corresponding sub-trees intersect.*

Also, from Gavril [13] we have the following characterisation of path graphs; see Monma and Wu [20] for more details on path graphs.

▶ **Theorem 4.** *The graph G is a path graph if and only if there exists a clique tree T, such that for every $v \in V(G)$, the set of maximal cliques containing v is a path in T such that two vertices are adjacent if the corresponding paths intersect.*

Succinct Data Structure for Ordinal Trees. Tree T is called an *ordinal tree* if for $z > 0$ and any $u \in V(T)$ with children $\{u_1, \ldots, u_z\}$, for $1 \leq i < j \leq z$, u_i is to the left of u_j [24]. By considering ordinal trees as balanced parenthesis Navarro and Sadakane [24] has given a $2n + o(n)$ bit succinct data structure.

▶ **Lemma 5.** *For any ordinal tree T with n nodes, there exists a $2n + o(n)$ bit Balanced Parentheses (BP) based data structure that supports the following functions among others in constant time :*

1. `lca`(i, j)*, returns the lowest common ancestor of two nodes i, j in T, and*
2. `child`(i, v)*, returns the $q-$th child of v in T.*

Rank-Select Data Structure. Bit-vectors are extensively used in the succinct representation given in Section 4. The following data structure due to Golynski et al. [15] and the functions supported by it are useful.

▶ **Lemma 6.** *Let B be an $n-$bit vector and $b \in \{0,1\}$. There exists an $n + o(n)$ bit data structure that supports the following functions in constant time:*

1. `rank`(B, b, i)*: Returns the number of b's up to and including position i in the bit vector B from the left.*
2. `select`(B, b, i)*: Returns the position of the i-th b in the bit vector B from left. For $i \notin [n]$ it returns 0.*

Non-Decreasing Integer Sequence Data Structure. Given a set of positive integers in the non-decreasing order we can store them efficiently using the differential encoding scheme for increasing numbers; see Section 2.8 of [23]. Let S be the data structure that supports differential encoding for increasing numbers then the function `accessNS`(S, i) returns the $i-$th number in the sequence.

▶ **Lemma 7.** *Let S be a sequence of n non-decreasing positive integers $a_1, \ldots, a_n, 1 \leq a_i \leq n$. There exists a $2n + o(n)$ bit data structure that supports* `accessNS`(S, i) *in constant time.*

Proof. We will prove the lemma by giving a construction of such a data structure. a_1 will be represented by a sequence of a_1 1's followed by a 0. Subsequently, a_i's are represented by storing $a_i - a_{i-1}$ many 1's followed by a 0. It will take at most $2n$ bits since there are n 0's and at most n 1's. Let this bit string be stored using the data structure of Lemma 6 and be denoted as B. B takes $2n + o(n)$ bits. `accessNS`(S, i) can be implemented using `rank`$(B, 1,$ `select`$(B, 0, i))$. ◀

Succinct Data Structure for Path Graphs. From [4] we have the following succinct data structure for path graphs that supports adjacency and neighbourhood queries with slight modifications in input. In the following lemma the endpoints of paths are input to the queries whereas in [4] the path indices are given as input.

▶ **Lemma 8.** *A path graph G has an $n \log n + o(n \log n)$-bit succinct representation. For a $u \in V(G)$, let $P_u = (s_u, t_u)$ be the path corresponding u in clique tree T of G. The succinct representation constructed from the clique tree T representation supports for $u \in V(G)$ the following queries:*

1. `adjacencyPG`(s_u, t_u, s_v, t_v)*: Returns true if $P_u = (s_u, t_u)$ intersects $P_v = (s_v, t_v)$ in $O(\log n)$ time else false,*
2. `pathep`(u)*: Returns the endpoints of path P_u, corresponding to u, in T in $O(\log n)$ time, and*
3. `neighbourhoodPG`(s_u, t_u)*: Returns the list of paths intersecting $P_u = (s_u, t_u)$ in $O(d_u \log n)$ time where d_u is the degree of vertex u.*
4. `getHPStartNode`(v)*: Returns the start node of heavy path π that contains $v \in V(T)$ in constant time. If v is not the first child, that is, it is adjacent to its parent by a light edge, then v itself is returned.*

Permutations. The following data structure by Munro et al. [21], gives a succinct representation for storing permutation of $[n]$.

▶ **Lemma 9** ([21])**.** *Given a permutation of $[n]$ there exists an $(n \log n + o(n \log n))$-bit data structure that supports the following queries.*

- $\pi(i)$*: Returns the $i-$th value in the permutation in $O(1)$ time.*
- $\pi^{-1}(j)$*: Returns the position of the $j-$th value in the permutation in $O(f(n))$ time for any increasing function $f(n) = o(\log n)$.*

Method of Partial Coloring. Let $\mathcal{G}$ be a graph class. A *partial coloring* of $G \in \mathcal{G}$ is the triple $\langle G, U, g\rangle$ where,

- $U \subseteq V(G)$ such that for $s > 0, |U| = s$, and
- $g : U \to \{1, \dots, s\}$ is a bijection.

Every vertex $u \in U$ is said to have color $g(u)$ and vertices in $V(G)\backslash U$ are said to be uncolored. Two partially colored graphs $\langle H_1, U_1, g_1\rangle$ and $\langle H_2, U_2, g_2\rangle$ are said to be different when either:

1. $E(H_1) \neq E(H_2)$, or
2. $E(H_1) = E(H_2) = E(H)$ for some $H \in \mathcal{G}$ and
   a. $U_1 \neq U_2$ and there does not exist a bijection $f : V(H_1) \to V(H_2)$ such that for every $u \in V(H_1)\backslash U_1$ there exists a $f(u) \in V(H_2)\backslash U_2$ with same set of colored neighbours, or
   b. for $U_1 = U_2 = U$ there exists $u \in V(H)\backslash U$ such that its colored neighbourhood in $\langle H_1, U_1, g_1\rangle$ and $\langle H_2, U_2, g_2\rangle$ are different.

Else, they are same. The method of counting by partial coloring as given in Theorem 1 of Acan et al. [1] can be defined using the following proposition.

▶ **Proposition 10.** *Let $\mathcal{G}'$ be the class of partially colored graphs obtained from class of graphs $\mathcal{G}$ by selecting m vertices out of n and coloring them using m distinct colors. Then, $|\mathcal{G}'| \leq \binom{n}{m}m!|\mathcal{G}|$. If there exists a graph class $\mathcal{G}^c \subset \mathcal{G}'$ then $|\mathcal{G}'| \geq |\mathcal{G}^c|$ and $|\mathcal{G}| \geq \frac{|\mathcal{G}^c|}{\binom{n}{m}m!}$.*

▶ Remark. While computing $|\mathcal{G}'|$, indistinguishable partially colored graphs can also be counted since we only require an upper bound, however, this is not the case while computing $|\mathcal{G}^c|$.

3 Class of $k-$Vertex Leafage Chordal Graphs and its Lower Bound

In this section, we define the class of $k-$vertex leafage chordal graphs, denoted $\mathcal{G}_k$, followed by the lower bound for $\log|\mathcal{G}_k|$. According to Theorem 3, a graph G is a chordal graph if there exists a tree T such that corresponding to every vertex $v \in V(G)$ there exists a sub-tree T_v of T and $\{u, v\} \in E(G)$ if and only if $V(T_u) \cap V(T_v) \neq \phi$ where T_u is the sub-tree corresponding to u. We call T the tree model of G. For every chordal graph there exists a tree T called the *clique tree* such that $V(T)$ is the set of maximal cliques of G. The *leafage* of a chordal graph G, denoted $l(G)$, is the minimum number of leaves of a tree T out of all possible trees characterising G. Leafage was studied by Lin et al. in [18]. Later, Chaplick and Stacho in [8] studied *vertex leafage*, denoted $vl(G)$. Let $L(u)$ denote the number of leaves of the sub-tree in T corresponding to $u \in V$ and $k = \max\limits_{u \in V} L(u)$. The smallest k for which there exists a tree T for G is called its vertex leafage.

Class of k-Vertex Leafage Chordal Graphs. The class of chordal graphs can be considered as the generalisation of path graphs using vertex leafage as the parameter. Theorem 4 implies that path graphs are chordal graphs with vertex leafage equal to two and unbounded leafage. Generalizing this, $\mathcal{G}_k$ is the set of all chordal graphs with vertex leafage at most k and unbounded leafage. Succinct data structure for path graphs is given by Balakrishnan et al. in [4]. In this paper, we present a succinct data structure for chordal graphs with vertex leafage at most k for $k \in o(n^c), c > 0$ using succinct data structure for path graphs as black-box.

Lower Bound. Counting chordal graphs involves heavy mathematical machinery as can be seen from Wormald [10]. Munro and Wu [22] uses the result from [10] to obtain lower bound for the cardinality of unlabeled chordal graphs. Here we give a much simpler technique for counting unlabeled chordal graphs with bounded vertex leafage. In order to derive the lower bound for $\log|\mathcal{G}_k|$ by implementing Proposition 10, we define two new graph classes, $\mathcal{G}'_k$ and $\mathcal{G}^c_k$, where $\mathcal{G}'_k$ corresponds to $\mathcal{G}'$ and $\mathcal{G}^c_k$ to $\mathcal{G}^c$ of Proposition 10.

Graph Class $\mathcal{G}'_k$. We consider the class of all partially colored $k-$vertex leafage chordal graphs, denoted $\mathcal{G}'_k$, that has for fixed $1 \leq m \leq n$, m out of n vertices colored using colors $\{1,\ldots,m\}$. Graphs in $\mathcal{G}'_k$ are obtained from graphs in $\mathcal{G}_k$ as follows. The input to the procedure is $G \in \mathcal{G}_k$ and a set $\{v_1,\ldots,v_m\}$ of m vertices of G. For each $G \in \mathcal{G}_k$, we get a set of $\binom{n}{m}m!$ graphs of $\mathcal{G}'_k$ where each graph G' is obtained by coloring the selected m vertices of G by a permutation of $\{1,\ldots,m\}$. A partially colored graph in $\mathcal{G}'_k$ is denoted $\langle H,U,g\rangle$ where $U \subset V(H), |U| = m$, and $g : U \to \{1,\ldots,m\}$.

▶ **Lemma 11.** *For each $k > 1$, $\log|\mathcal{G}'_k| \leq \log|\mathcal{G}_k| + n\log n - (n-m)\log(n-m) - 2n + O(\log n)$.*

Graph Class $\mathcal{G}^c_k$. As per Proposition 10, we construct the class $\mathcal{G}^c_k \subset \mathcal{G}'_k$ for which we can obtain exact count or a lower bound. We give a construction mechanism for graphs in $\mathcal{G}^c_k$ such that all graphs in it have the following properties. For $G \in \mathcal{G}^c_k$,

- $U \subseteq V(G)$, with $|U| = m$, is fixed and have a fixed coloring using colors $\{1,\ldots,m\}$,
- let T be the tree model corresponding to G then the sub-trees in the tree model corresponding to vertices in U consist of only one node, and
- sub-trees of T corresponding to vertices in $V(G)\backslash U$ have at most k leaves with at least $\frac{m+1}{2(k-1)}$ sub-trees with exactly k leaves.

In other words, for all the partially colored graphs in $\mathcal{G}^c_k$, U and g are fixed. Based on the tree model, the vertices of a graph $H \in \mathcal{G}^c_k$ with tree model T are of two types:

1. **basis vertices U:** all m vertices in U are represented by single-node sub-trees of T, and
2. **dependent vertices $V(H)\backslash U$:** rest of the $(n-m)$ vertices are represented by sub-trees in T with number of leaves at most k with at least $\frac{m+1}{2(k-1)}$ sub-trees with exactly k leaves. Let the dependent vertices corresponding to these $\frac{m+1}{2(k-1)}$ sub-trees be denoted U'.

We have the following useful proposition:

▶ **Proposition 12.** *For a rooted tree T, the set $V' \subseteq V(T)$ uniquely defines a sub-tree T' of T such that if $u,v \in V'$ then the path connecting it is in T'.*

The sub-trees corresponding to the basis and dependent vertices are called *basis sub-trees* and *dependent sub-trees*, respectively. Let $F = \frac{m+1}{2(k-1)}$. The input to the procedure that constructs H are:

1. n, m, k, and
2. $\mathcal{J} = \{J_1,\ldots,J_{n-m-F}\}$ where for $1 \leq i \leq n-m-F, 1 \leq t \leq k$, $J_i = \{a^i_1,\ldots,a^i_k\}$ and $1 \leq a^i_t \leq m$.

The construction mechanism that constructs H is as follows.

1. Consider a rooted complete binary tree T with $m > 0$ nodes and let them be colored from 1 to m.
2. **Construction of basis sub-trees.** For $1 \leq j \leq m$, let each node $b_j \in V(T)$ be a single node sub-tree, T_j. These sub-trees are the *basis sub-trees.* The basis sub-trees correspond to the m basis vertices of H, denoted by U. Let the basis vertices be colored by the color assigned to the nodes to which they are assigned.
3. **Construction of dependent sub-trees.** First we define the fixed sub-trees with k leaves. Since T is a complete binary tree it has $\frac{m+1}{2}$ leaves. Let the set of leaves of T be denoted by L. For $1 \leq t \leq F$, partition L into blocks $L = \bigcup\limits_t L_t$ such that $|L_t| = k-1$

and for $1 \le z < z' \le t$, there does not exist a node in L_z with color greater the smallest color of nodes in $L_{z'}$. For every $u_t \in U'$, construct a sub-tree T_{u_t} in T by connecting paths from the $k-1$ leaves in L_t to the root of T. This ensures that T_{u_t} has k leaves. So, $|U'| = F$. For $1 \le i \le n-m-F$, construct sub-tree T_i from $J_i = \{a_1^i, \dots, a_k^i\}$, where for $1 \le t \le k, a_t^i \in V(T)$, such that J_i is the set of k nodes of T_i; as per Proposition 12, J_i uniquely defines a sub-tree. These sub-trees are called *dependent sub-trees* and correspond to the dependent vertices in $V(H)\backslash U$. The sub-trees corresponding to nodes of U' ensure that the chordal graph is connected and there are at least F sub-trees with k leaves there by making H a $k-$vertex leafage chordal graph. Also, apart from U and g, U' is also fixed for any partially colored $k-$vertex leafage chordal graph in $\mathcal{G}_k^c$.

Convention. A node $a_j \in V(T)$ will also be used to denote the sub-tree T_j corresponding to the basis vertex of H.

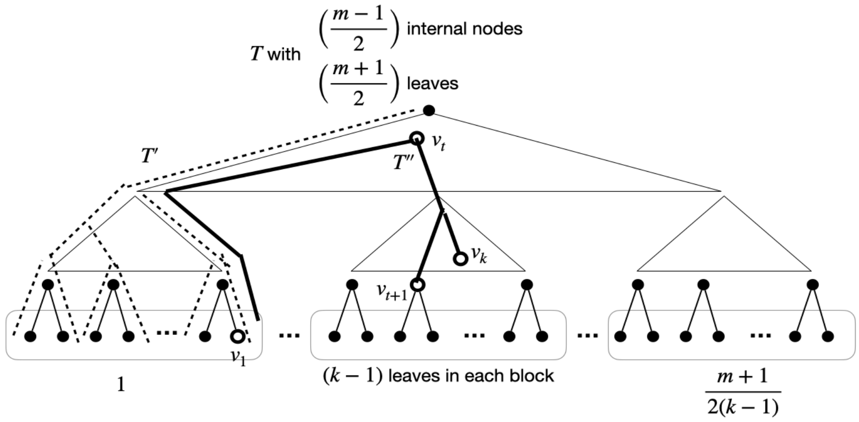

T with $\left(\frac{m-1}{2}\right)$ internal nodes
$\left(\frac{m+1}{2}\right)$ leaves
T'
T''
v_t
v_k
v_{t+1}
v_1
1
$(k-1)$ leaves in each block
$\frac{m+1}{2(k-1)}$

Figure 1 The complete rooted binary tree T with m nodes constructed to produce a graph in $\mathcal{G}_k^c$. T' is one of the sub-trees of T with exactly k leaves corresponding to a dependent vertex in U'. T'' is the sub-tree corresponding to a dependent vertex in $V(H)\backslash(U \cup U')$ constructed from $\{v_1, \dots, v_t, \dots, v_k\}$ which are the k selected nodes of T.

Thus, H is defined by $\mathcal{J} = \{J_1, \dots, J_{n-m-F}\}$ where each J_i defines a sub-tree T_i. For an example construction refer Figure 1. From the construction above we have the following lemma.

▶ **Lemma 13.** $\mathcal{G}_k^c \subseteq \mathcal{G}_k'$.

Computing $|\mathcal{G}_k^c|$. In order to compute $|\mathcal{G}_k^c|$ and use Proposition 10, we first note the following consequence of Lemma 13.

▶ **Proposition 14.** $\log|\mathcal{G}_k'| \ge \log|\mathcal{G}_k^c|$.

Let K denote the set of all possible $\mathcal{J}$'s. We have the following useful proposition and lemma.

▶ **Proposition 15.** *For $1 \le i, i' \le n-m-F, 1 \le j \le m$, if $J_i \ne J_{i'}$ then wlog there exists $a_j \in U$ such that $a_j \in J_i$ and $a_j \notin J_{i'}$.*

▶ **Lemma 16.** *Let $\mathcal{J}, \mathcal{J}' \in K$ where $\mathcal{J} = \{J_1, \dots, J_{n-m-F}\}$ and $\mathcal{J}' = \{J_1', \dots, J_{n-m-F}'\}$ such that for $1 \le s \le n-m-F, J_s \ne J_s'$. Also, let H_1 and H_2 be the graphs generated from $\mathcal{J}$ and $\mathcal{J}'$, respectively, and u be the vertex corresponding to s. Then, $N_{H_1}(u) \cap U \ne N_{H_2}(u) \cap U$ where $N_{H_1}(u)$ and $N_{H_2}(u)$ are the neighbours of u in H_1 and H_2, respectively.*

The following is the central lemma used to obtain the lower bound. For $1 \leq s \leq n - m - F$, let $\mathcal{J}(H)$ denote the $\mathcal{J} \in K$ that produces the graph $H \in \mathcal{G}^c_k$.

▶ **Lemma 17.** *Let $\langle H_1, U, g\rangle, \langle H_2, U, g\rangle$ be constructed from $\mathcal{J}, \mathcal{J}' \in K$. Then $\langle H_1, U, g\rangle$ and $\langle H_2, U, g\rangle$ are same if and only if $\mathcal{J} = \mathcal{J}'$.*

In order to obtain $|\mathcal{G}^c_{t,d}|$ we prove the following lemma first.

▶ **Lemma 18.** *$|\mathcal{G}^c_k| = |K|$.*

The following is an important lemma.

▶ **Lemma 19.** *$\log|\mathcal{G}^c_k| \geq kn \log n - kn \log k$.*

The following theorem gives the lower bound.

▶ **Theorem 1.** *For $k > 0$ in $o(n^c), c > 0, \log|\mathcal{G}_k| \geq (k-1)n \log n - kn \log k - O(\log n)$.*

We have the following proposition.

▶ **Proposition 20.** *For $k > 0$ in $o(n^c), c > 0$ and sufficiently large n, $\log|\mathcal{G}_k| \geq (k-1)n \log n$.*

4 Succinct Data Structure

The high-level procedure used to obtain the succinct data structure for $k-$vertex leafage chordal graph G given as $(T, \{T_1, \ldots, T_n\})$ is as follows:

1. From the tree T, obtain path graph H with $(k-1)n$ vertices by decomposing each sub-tree $T_i, 1 \leq i \leq n$, into at most $(k-1)$ paths such that each path created corresponds to a vertex of H. The input $(T, \{T_1, \ldots, T_n\})$ is decomposed into paths and we get $(T, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ where $\mathcal{P}_i = \mathcal{P}'_i \cup \mathcal{P}''_i, 1 \leq i \leq n$, where $\mathcal{P}'_i = \{P^i_1, \ldots, P^i_{k/2}\}$ and $\mathcal{P}''_i = \{P^i_{k/2+1}, \ldots, P^i_{k-1}\}$ such that if we have T and $\mathcal{P}'_i$ we can compute $\mathcal{P}''_i$.
2. Out of the $(k-1)n$ paths, we store $kn/2$ paths of H and tree T using the data structure for path graphs given in [4]. The rest of the $(k/2-1)n$ paths are computed from the tree T and the stored $kn/2$ paths of H. In other words, for each $T_i, \mathcal{P}'_i$ is stored and $\mathcal{P}''_i$ is computed from $\mathcal{P}'_i$ and T.

The succinct representation for G consists of the following contents:

1. $(T, \mathcal{P}'_1 \cup \mathcal{P}'_2 \cup \ldots \mathcal{P}'_n)$ is stored using the method for storing path graphs given in [4]. This data structure stores the tree T using the data structure of Lemma 5.
2. for $1 \leq j \leq k/2$, the mapping from indices of $P^i_j \in \mathcal{P}'_i$ to the indices of paths in the data structure of [4] that stores $(T, \mathcal{P}'_1 \cup \mathcal{P}'_2 \cup \ldots \cup \mathcal{P}'_n)$ as mentioned in the above step.

The following lemma is important before getting into the details of the data structure.

▶ **Lemma 21.** *Consider clique tree T of $k-$vertex leafage chordal graph G such that k is odd. Then there exists a tree model, denoted T', for G with at most $3n$ nodes such that for $1 \leq i \leq n, T'_i$ is the sub-tree in T' corresponding to $v_i \in V(G)$ and number of leaves of T_i is even.*

In this paper, we only consider even $k \geq 2$ as any T_i with odd number of leaves can be converted to even number of leaves as per Lemma 21. Note that the total increase in number of nodes of T is only constant times n. We first present the method to transform a $k-$vertex leafage chordal graph G to path graph H with $(k-1)n$ vertices followed by the construction of the data structure.

4.1 Transforming $(T, \{T_1, \ldots, T_n\})$ to $(T, \mathcal{P}'_1 \cup \ldots \cup \mathcal{P}'_n)$

The transformation from $(T, \{T_1, \ldots, T_n\})$ to $(T, \mathcal{P}'_1 \cup \ldots \cup \mathcal{P}'_n)$ happens in two steps as below:

a. pre-process the tree T into an ordinal tree, and
b. decompose each $T_i, 1 \le i \le n$, into $\mathcal{P}'_i \cup \mathcal{P}''_i$.

Pre-processing is done using the method as explained in Section 3 of [4]. We describe it here for ease of reading.

Pre-processing the Tree. Fix a root node for T and perform heavy path decomposition on it. For $v \in V(T)$ order its children $(w_1, \ldots, w_c)$ such that $\{v, w_1\}$ is a heavy edge. Let the children adjacent to v by light edges $(w_2, \ldots, w_c)$, be ordered arbitrarily. This ordering of children of a node of the tree makes it an ordinal tree. Label the nodes of this ordinal tree based on the pre-order traversal; see Section 12.1 of [9] for more details of the pre-order traversal of trees. Labels assigned to nodes in this manner are called the *pre-order labels of the nodes*. Throughout the rest of our paper, this ordinal rooted tree labeled with pre-order will be referred to as the tree model; see Section 3 of [4] for more details. After pre-processing, T is an ordinal tree on which heavy-path decomposition is performed such that all heavy edges are left aligned and nodes are numbered based on the order in which they are visited in the pre-order traversal of T. Since G is $k-$vertex leafage chordal graph, the number of leaves of T_i is at most k. We obtain the tree representation of the path graph $H \in \mathcal{G}_k(2, (k-1)n)$, denoted $(T, \mathcal{P}'_1 \cup \mathcal{P}''_1 \cup \ldots \cup \mathcal{P}'_n \cup \mathcal{P}''_n)$, by carefully selecting the $k-1$ paths from sub-tree $T_i, 1 \le i \le n$. The function `getPaths` that does this is explained next.

Function `getPaths`. Given the sub-tree T_i of T with number of leaves at most k, the function returns the set of paths $\mathcal{P}'_i$ such that:

1. $|\mathcal{P}'_i| \le k/2$,
2. $\mathcal{P}'_i$ along with T can uniquely determine $\mathcal{P}''_i$, and
3. $T_i = \mathcal{P}'_i \cup \mathcal{P}''_i$.

The function can be implemented as follows. Let the leaves of T_i labeled based on the order in which nodes are visited in the pre-order traversal of the tree, be $\{l_1, \ldots, l_{k_i}\}, k_i \le k$. For $1 \le j \le k_i/2$, pair the smallest leaf l_j with the largest leaf l_{k_i-j+1} to get path P_j. Let the set of paths obtained from T_i be denoted by $\mathcal{P}'_i$.

Ordering Paths in $\mathcal{P}'_i$. Let $\mathcal{P}'_i = \{P^i_1, \ldots, P^i_{k_i}\}, 1 \le k_i \le k/2$. For $1 \le j < j' \le k_i/2$, $P^i_j = (a_j, b_j)$ and $P^i_{j'} = (a_{j'}, b_{j'})$, $P^i_j \prec_{\mathcal{P}} P^i_{j'}$ if $a_j \le a_{j'} \le b_{j'} \le b_j$. $\prec_{\mathcal{P}}$ is a total order on $\mathcal{P}_i$, since for any two paths $P^i_j, P^i_{j'} \in \mathcal{P}_i$, either $a_j \le a_{j'} \le b_{j'} \le b_j$ or $a_{j'} \le a_j \le b_j \le b_{j'}$.

Relation Between $P^i_j, P^i_{j+1} \in \mathcal{P}'_i$. For $1 \le j \le k/2-1$, the paths connecting P^i_j and P^i_{j+1} are the paths in $\mathcal{P}''_i$. They are computed using the function `connector`. Let $P^i_j = (a_j, b_j)$ and $P^i_{j+1} = (a_{j+1}, b_{j+1})$. We define `connector` based on the following two conditions:

1. P^i_j and P^i_{j+1} are not intersecting: Since $P^i_j \prec_{\mathcal{P}} P^i_{j+1}$ and P^i_{j+1} is contained inside a sub-tree rooted at $\texttt{lca}(a_j, b_j)$, $\texttt{connector}(i, j, j+1) = (\texttt{lca}(a_j, b_j), \texttt{lca}(a_{j+1}, b_{j+1}))$ connects P^i_j and P^i_{j+1}.
2. P^i_j and P^i_{j+1} are intersecting: $\texttt{connector}(i, j, j+1) = \texttt{NULL}$ in this case.

We have the following useful lemmas.

▶ **Lemma 22.** *For $1 \le i \le n$, the following holds:*

1. *$|\mathcal{P}'_i| \leq k/2$ and $|\mathcal{P}''_i| \leq k/2 - 1$,*
2. *$T_i = \mathcal{P}'_i \cup \mathcal{P}''_i$ where $\mathcal{P}''_i = \texttt{connector}(i, 1, 2) \cup \texttt{connector}(i, 2, 3) \cup \ldots \cup \texttt{connector}(i, k_i/2 - 1, k_i/2)$, and*
3. *$T = \bigcup_{i=1}^{n} F_i$ where $F_i = \mathcal{P}'_i \cup \mathcal{P}''_i$.*

▶ **Lemma 23.** *For $1 \leq j \leq k_i/2$, let $P^i_j \in \mathcal{P}'_i$ such that $P^i_j = (a^i_j, b^i_j)$, $e = \texttt{lca}(a^i_j, b^i_j)$ and $Q_1 = (e, b^i_j)$. Also, let $\texttt{connector}(i, j, j+1) \neq \texttt{NULL}$. Then,*

1. *$(e, c) \in E(Q)$ is a light edge, and*
2. *$(e, c') \in E(\texttt{connector}(i, j, j+1))$ is a light edge.*

Finally, we have the following proposition and lemma regarding adjacency and neighbourhood of $v \in V(G)$.

▶ **Proposition 24.** *$\{u, v\} \in E(G)$ if and only if there exists $P \in \mathcal{P}'_u \cup \mathcal{P}''_u, Q \in \mathcal{P}'_v \cup \mathcal{P}''_v$, such that $P \cap Q \neq \phi$.*

For $1 \leq j \leq k_v, 1 \leq j' \leq k_v - 1$, let $\beta(P^v_j)$ and $\beta(\texttt{connector}(v, j', j'+1))$ represent the paths in $\mathcal{P}'_i \cup \mathcal{P}''_i \cup \ldots \cup \mathcal{P}'_n \cup \mathcal{P}''_n$ intersecting P^v_j and $\texttt{connector}(v, j', j'+1)$, respectively.

▶ **Lemma 25.** *For $1 \leq i \leq n$, let $\mathcal{P}_i = \mathcal{P}'_i \cup \mathcal{P}''_i$. For $P \in \mathcal{P}_i$ the following holds:*

1. *$|\beta(P)| \leq (k-1)d_i$, and*
2. *$\sum_{P \in \mathcal{P}_i} |\beta(P)| \leq (k-1)^2 d_i$ where d_i is degree of $v_i \in V(G)$.*

4.2 Construction

Index of Paths in G and H. For $P^i_1, \ldots, P^i_{k_i} \in \mathcal{P}'_i$, we index paths of G and H in the following ways:

1. in G paths are ordered $\{P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n}\}$ where $P^i_1, 1 \leq i \leq n$ is in the increasing order of the starting nodes of P^i_1 and for $1 \leq j \leq k_i, P^i_j$ are ordered based on $\prec_{\mathcal{P}}$, and
2. in H paths are ordered based on their starting nodes; this is as per the storage scheme followed in [4] for path graphs.

We will order the vertices of G based on the order of the starting nodes of the first paths $P^i_1 \in \mathcal{P}_i$ of each T_i. The data structure for $k-$vertex leafage chordal graphs consists of the following components. We distinguish the case when k is odd or even only when the difference alters the higher order term in the space complexity, else we assume k is even. We will show later that k being odd or even does not impact the space complexity of our data structure.

Path Graph H. The path graph H that we store is $(T, \bigcup_{i=1}^{n} \mathcal{P}'_i)$ where $|\bigcup_i \mathcal{P}'_i| \leq nk/2$ if k is even and $|\bigcup_i \mathcal{P}'_i| \leq \left(\frac{k-1}{2} + \frac{1}{2}\right)n$ if k is odd. For $1 \leq i \leq n$, we do not store $\texttt{connector}(i, 1, 2), \texttt{connector}(i, 2, 3), \ldots, \texttt{connector}(i, k_i - 1, k_i)$ but compute it when required. In other words, T_i can be computed from T and $\mathcal{P}'_i$. Thus, path graph H can be stored using $\frac{nk}{2} \log n + o(nk \log n)$ bits as per Lemma 8 if k is even and $\left(\frac{k-1}{2} + \frac{1}{2}\right) n \log n + o(nk \log n)$ bits if k is odd. The data structures used in the succinct representation of path graphs as given in Lemma 8 does not require T to be a clique tree of H as long as the number of nodes in T is constant times $|V(H)|$, so $(T, \bigcup_{i=1}^{n} \mathcal{P}_i)$ is a valid input that represents H.

Array K. K is a one dimensional array of length n. For $1 \leq i \leq n, K[i]$ stores $|\mathcal{P}'_i|$, that is, the number of paths that the sub-tree T_i gets decomposed into. K takes at most $n \log k$ bits of space. The following function is supported.

- `getSize`(i) : Given $1 \leq i \leq n$, the function returns $K[i]$ else 0.

Bit-vector F. F is a bit vector of length at most $kn/2$. Let the index of the first path of each vertex in the order $(P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n})$ be $\{i_1, \ldots, i_n\}$ where $P^i_j \in \mathcal{P}'_i, 1 \leq i \leq n, 1 \leq j \leq k_i$. Since the vertices are numbered based on the start node of their first paths, $\{i_1, \ldots, i_n\}$ is an increasing sequence of numbers with maximum value of $kn/2$ and can be stored using the differential encoding scheme of Lemma 7. This data structure is also denoted by F and takes at most $kn + o(kn)$ bits of space. The following function is supported.

- `getIndex`(i) : Given $1 \leq i \leq n$, the function returns `accessNS`(i, F) else 0. `accessNS`(i, F) returns the value at the $i-$th position in the sequence $\{i_1, \ldots, i_n\}$.

Bit-vector D. Consider the order of paths $O = (P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n})$. For $1 \leq j \leq kn/2, D[j] = i$ if $O[j]$ is a path corresponding to $u_i \in V(G)$. Since D is an increasing sequence it can be stored using the data structure of Lemma 7 taking $O(kn)$ bits. D contains at most n 1's and $kn/2$ 0's.

Permutation π. π store the mapping of indices of paths in H to paths in G by storing the indices of paths $(P^1_2, \ldots, P^1_{k_1}, \ldots, P^n_2, \ldots, P^n_{k_n})$ in G in that order. π is stored using the data structure of Lemma 9 and takes $\left(\frac{k}{2} - 1\right) n \log n + o(kn \log n)$ bits of space if k is even and $\left(\frac{k-1}{2} - \frac{1}{2}\right) n \log n + o(kn \log n)$ bits if k is odd.

Function `alpha`. The function `alpha` takes $1 \leq i \leq n$, as input and returns the indices of paths in $\mathcal{P}'_i$ corresponding to $u_i \in V(G)$ in the tree representation of H. Function returns $\{\pi(p), \ldots, \pi(p+s)\}$ where $p \leftarrow$ `getIndex`$(i) - i + 1$ and $s \leftarrow$ `getSize`(i). `getIndex`$(i) - i + 1$ is the index of P^i_2 in $(P^1_2, \ldots, P^1_{k_1}, \ldots, P^n_2, \ldots, P^n_{k_n})$.

▶ **Lemma 26.** *Given the index $1 \leq i \leq n$, of $u_i \in V(G)$,* `alpha`(i) *returns the indices in H of paths in $\mathcal{P}'_i$ corresponding to u_i in $O(k_i)$ time.*

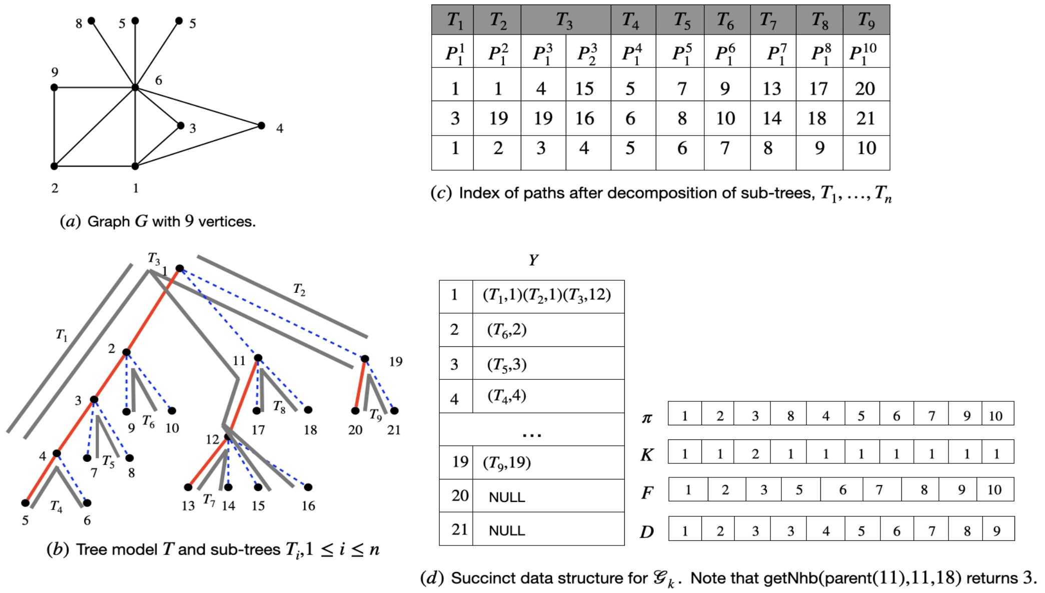
Bit-vector C. C is an array of n bit-vectors each of length $(k/2 - 1)$. For $1 \leq i \leq n, 1 \leq j \leq k/2 - 1$, $C[i][j] = 1$ if `connector`$(i, j, j+1) \neq$ `NULL` else $C[i][j] = 0$. C takes a total space of $n(k/2 - 1)$ bits. The following function is supported.

- `isConnector`(i, j) : Given $1 \leq i \leq n, 1 \leq j \leq k/2 - 1$, the function returns true if $C[i][j] = 1$ else false.

Function `getCPath`. For $1 \leq i \leq n, 1 \leq j \leq k_i/2 - 1$ and $P^i_j = (a_j, b_j), P^i_{j+1} = (a_{j'}, b_{j'})$, the function takes paths P^i_j and P^i_{j+1} as input and returns `connector`$(i, j, j+1)$ if `isConnector`$(i, j))$ is true else `NULL`. Note that H stored as per Lemma 8 contains the tree T and this allows us to perform the `lca` operation.

▶ **Lemma 27.** *For $1 \leq i \leq n, 1 \leq j \leq k_i/2 - 1$, given paths P^i_j, P^i_{j+1} as input, the function* `getCPath`(P^i_j, P^i_{j+1}) *returns* `connector`$(i, j, j+1)$ *in constant time.*

▶ **Lemma 28.** *For $k > 1$ and in $o(n^c), c > 0$, there exists a $(k-1)n \log n + o(kn \log n)$-bit succinct data structure for the class of $k-$vertex leafage chordal graphs.*



(a) Graph G with 9 vertices.

(b) Tree model T and sub-trees $T_i, 1 \leq i \leq n$

T_1	T_2	T_3		T_4	T_5	T_6	T_7	T_8	T_9
P_1^1	P_1^2	P_1^3	P_2^3	P_1^4	P_1^5	P_1^6	P_1^7	P_1^8	P_1^{10}
1	1	4	15	5	7	9	13	17	20
3	19	19	16	6	8	10	14	18	21
1	2	3	4	5	6	7	8	9	10

(c) Index of paths after decomposition of sub-trees, $T_1, \ldots, T_n$

Y

1	$(T_1,1)(T_2,1)(T_3,12)$
2	$(T_6,2)$
3	$(T_5,3)$
4	$(T_4,4)$
…	
19	$(T_9,19)$
20	NULL
21	NULL

π	1	2	3	8	4	5	6	7	9	10
K	1	1	2	1	1	1	1	1	1	1
F	1	2	3	5	6	7	8	9	10	
D	1	2	3	3	4	5	6	7	8	9

(d) Succinct data structure for $\mathscr{G}_k$. Note that getNhb(parent(11),11,18) returns 3.

Figure 2 (a) An example 4-vertex leafage chordal graph G, (b) tree representation of G after pre-processing, (c) the index of the paths generated from the sub-trees along with their start node (second row), end node (third row), and index (forth row), (d) the components of the succinct data structure for G.

4.3 Adjacency and Neighbourhood Queries

The following lemma is useful.

▶ **Lemma 29.** *Let T_i and $T_{i'}$ intersect and $P_1^i = (a_1^i, b_1^i), P_1^{i'} = (a_1^{i'}, b_1^{i'})$. Wlog,*

1. *if $a_1^i \leq a_1^{i'} \leq b_1^{i'} \leq b_1^i$, then*
   a. *there exists $P_j^i, 1 \leq j \leq k_i/2$ such that $P_j^i \cap P_1^{i'} \neq \phi$, or*
   b. *there exists* `connector`$(i, j, j+1) \cap P_j^{i'} \neq \phi, 1 \leq j \leq k_i/2 - 1$.
2. *Else, $a_1^i \leq a_1^{i'} \leq b_1^i \leq b_1^{i'}$ or $a_1^{i'} \leq a_1^i \leq b_1^{i'} \leq b_1^i$.*

Adjacency Query. Given indices of two vertices $u_i, u_j \in V(G)$ and the succinct representation for the $k-$vertex leafage chordal graphs, the adjacency query returns true if $\{u_i, u_j\} \in E(G)$ else false. The characterisation of $\{u_i, u_j\} \in E(G)$ in terms of path intersections in $(T, \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n)$ where $\mathcal{P}_i = \mathcal{P}'_i \cup \mathcal{P}''_i, 1 \leq i \leq n$, is given by Proposition 24. Algorithm 1 gives an implementation of the adjacency query.

▶ **Lemma 30.** *For $k \in o(n^c), c > 0$, the class of $k-$vertex leafage chordal graphs have a $(k-1)n \log n + o(kn \log n)$ bit succinct data structure that supports adjacency query in $O(k \log n)$ time.*

Neighbourhood Query. Given $v \in V(G)$ and the succinct representation of $k-$vertex leafage chordal graph G, the neighbourhood query returns the neighbours of v. We use the following additional data structure.

Algorithm 1 Given indices i, j as input, the function returns true if $\{u_i, u_j\} \in E(G)$ else false.

```
1  Function adjacency($(i, j)$):
2      $\mathcal{P}'_i \leftarrow$ alpha$(i)$, $\mathcal{P}'_j \leftarrow$ alpha$(j)$, $s \leftarrow$ NULL
3      Let $P^i_1 = (a^i_1, b^i_1)$ and $P^j_1 = (a^j_1, b^j_1)$ obtained using pathep(getIndex$(i)$) and
         pathep(getIndex$(j)$), respectively.
4      Check if $P^i_1$ and $P^j_1$ intersect as follows:
5      if adjacencyPG$(H, P^i_1, P^j_1)$ is true then
6          return true
7      Check if endpoints of first path of one falls within the range of the end points of
         the first path of the other as follows:
8      if $a^i_1 \leq a^j_1 \leq b^j_1 \leq b^i_1$ then
9          $s \leftarrow i$
10
11     if $a^j_1 \leq a^i_1 \leq b^i_1 \leq b^j_1$ then
12         $s \leftarrow j$
13
14     if $s \neq$ NULL then
15         if $s = i$ then
16             foreach $1 \leq t \leq$ getSize$(i) - 1$ do
17                 if adjacencyPG$(H$, pathep$(\mathcal{P}'_i[t])$, pathep$(\mathcal{P}'_j[1]))$ is true then
18                     return true
19             foreach $2 \leq t \leq$ getSize$(i) - 1$ do
20                 if $t = 2$ and
                     adjacencyPG$(H$, getCPath(getIndex$(i)$, $\mathcal{P}'_i[t])$, pathep$(\mathcal{P}'_j[1]))$ is
                     true then
21                     return true
22                 else
23                     if adjacencyPG$(H$, getCPath$(\mathcal{P}'_i[t-1], \mathcal{P}'_i[t])$, pathep$(\mathcal{P}'_j[1]))$ is
                         true then
24                         return true
25         if $s = j$ then
26             Perform the same steps as done in Line 15 for the case $s = i$ with $i$ and $j$
                 interchanged.
27     return false
```

Array Y. Y is a one dimensional array of length at most n that stores an array at each of its locations. For $1 \leq i \leq n$, $Y[i]$ stores the array of records where each record is of the form (r, s) such that $1 \leq r \leq n$ is the index of the sub-tree that has node $u_i \in V(T)$ as the lca of endpoints of P^r_1 and for $1 \leq s \leq n$, node u_s as the lca of endpoints of $P^r_{k_r}$. The records at $Y[i]$ are stored in the increasing order of s. The total space taken by Y is $2n \log n$ bits as each tree takes $2 \log n$ bits and there are n trees. The following function is supported.

- `getNhb`(l, l', a): Given $1 \leq l, l' \leq n$, the function returns the list of trees with index $1 \leq j \leq n$, such that lca of endpoints of P^j_1 is equal to l and lca of endpoints of $P^j_{k_j}$ greater than or equal to l' and less than or equal to a in $O(\log n + d)$ time where d is the number of trees returned. The function can be implemented as follows:
  1. Performing binary search on the array A of records at $Y[l]$ to first obtain the range of s values greater than or equal to l'. Let this range be $[p_1, p_2]$.
  2. Performing second binary search on $A[p_1, p_2]$ to obtain the range of s values that are less than or equal to a in $A[p_1, p_2]$. Let this range be $[p'_1, p'_2]$.
  3. Return the indices of trees, that is, the r values stored in records $A[p'_1, p'_2]$.

  The binary search takes $O(\log n)$ time and the tree indices are returned in d time where $d = p'_2 - p'_1 + 1$. Thus, `getNhb` takes a total time of $O(\log n + d)$.

Algorithm 2 gives an implementation of the neighbourhood query.

▶ **Lemma 31.** *For $k \in o(n^c), c > 0$, the class of $k-$vertex leafage chordal graphs have a $(k-1)n \log n + o(kn \log n)$ bit succinct data structure that supports neighbourhood query for vertex v_i in $O(k^2 d_i \log n + \log^2 n)$ time using additional $2n \log n$ bits where d_i is the degree of v_i.*

Thus, we have the following theorem.

▶ **Theorem 2.** *For $k > 1$ and in $o(n^c), c > 0$, a graph $G \in \mathcal{G}_k$ has a $(k-1)n \log n + o(kn \log n)$-bit succinct data structure that supports adjacency query in $O(k \log n)$ time and using additional $2n \log n$ bits the neighbourhood query for vertex v in $O(k^2 d_v \log n + \log^2 n)$ time where d_v is the degree of $v \in V(G)$.*

Proof. From Lemma 28, we know that for $k \in o(n/\log n)$, the $(k-1)n \log n + o(kn \log n)$-bit data structure is succinct. From Lemma 30, we know that the data structure supports adjacency query in time $O(k \log n)$ and from Lemma 31, we know that the neighbourhood query is supported in time $O(k^2 d_i \log n + \log^2 n)$ using additional $2n \log n$ bits. ◀

5 Conclusion

The parameters, leafage and vertex leafage, are defined for chordal graphs which is a special case of intersection graphs. In comparison, boxicity and interval number allow us to model general graphs as intersection graphs. However, they do not give a tree model like in the case of chordal graphs. We ask the following question: "*Can leafage and vertex leafage be generalized for any graph?*" The answer is positive if we consider the nice tree-decomposition. For any graph, the nice tree-decomposition allows us to establish a correspondence between vertices of the graph and sub-trees of the tree obtained. As one can see clearly, the parameters leafage and vertex leafage become applicable to general graphs now. For chordal graphs we know that the clique tree has nodes that correspond to the maximal cliques of the graph. However, we lose such nice properties in the case of nice tree-decomposition. Despite these limitations we think it is worthwhile to generalize these parameters and design a succinct data structure for the more general class thus formed. It is also interesting to note that a unit increase in the leafage parameter increases the number of graphs in the class by $n \log n$ as compared to $2n \log n$ in the case of boxicity or interval number. From Balakrishnan et al. [3] we know that for the succinct data structure designed for graphs with bounded boxicity $d > 0$ using the succinct data structure for interval graphs an efficient but easy implementation for neighbourhood query is not possible. For bounded leafage parameter where the intersection model is a tree, it will be interesting to see if there exists a simple and efficient implementation of the neighbourhood query. Another challenging direction is to consider whether space-efficient graph algorithms can be designed for these specialized graph classes [5, 7, 11].

Algorithm 2 Given index i as input, the function returns the set of vertices adjacent to $u_i \in V(G)$.

```
Function neighbourhood($i$):
    $\mathcal{P}'_i \leftarrow$ alpha($i$), $\mathcal{P} \leftarrow \phi$
    Add pathep(getIndex($i$)) to $\mathcal{P}$
    foreach $j \in \mathcal{P}'_i$ do
        Add pathep($j$) to $\mathcal{P}$
    Add getCPath(getIndex($i$), $\mathcal{P}'_i[1]$) to $\mathcal{P}$ if it is not NULL
    foreach $1 \leq j \leq$ getSize($i$) $- 1$ do
        Add getCPath($\mathcal{P}'_i[j]$, $\mathcal{P}'_i[j+1]$) to $\mathcal{P}$ if it is not NULL
    $N \leftarrow \phi$
    Let $t$ be a bit-vector of length $n$ initialised to 0
    foreach $(a, b) \in \mathcal{P}$ do
        $N' \leftarrow$ neighbourhoodPG($a, b$)
    foreach $j' \in N'$ do
        $p \leftarrow \pi^{-1}(j')$
        if $t[p] \neq 1$ then
            Add $D[p]$ to $N$
            $t[p] \leftarrow 1$
    $P^i_1 \leftarrow$ pathep(getIndex($i$))
    $P^i_2 \leftarrow$ pathep($\mathcal{P}_i[1]$)
    Let $P^i_1 = (a_1, b_1), P^i_2 = (a_2, b_2), l \leftarrow$ lca($a_1, b_1$), $l' \leftarrow l$
    if lmost_child(parent($l$)) $= l$ then
        $v \leftarrow$ parent(getHPStartNode($l$))
    else
        $v \leftarrow$ parent($l$)
    while $v \neq$ NULL do
        Add getNhb($v, l', b_1$) to $N$
        $l \leftarrow v$
        if child(1, parent($l$)) $= l$ then
            $v \leftarrow$ parent(getHPStartNode($l$))
        else
            $v \leftarrow$ parent($l$)
```

References

1 H. Acan, S. Chakraborty, S. Jo, K. Nakashima, K. Sadakane, and S.R. Satti. Succinct navigational oracles for families of intersection graphs on a circle. *Theor. Comput. Sci.*, 928(C):151–166, September 2022. doi:10.1016/j.tcs.2022.06.022.

2 H. Acan, S. Chakraborty, S. Jo, and S. R. Satti. Succinct encodings for families of interval graphs. *Algorithmica*, 83(3):776–794, 2021.

3 G. Balakrishnan, S. Chakraborty, S. Jo, N. S. Narayanaswamy, and K. Sadakane. Succinct data structure for graphs with d-dimensional t-representation, 2023. arXiv:2311.02427.

4 G. Balakrishnan, S. Chakraborty, N.S. Narayanaswamy, and K. Sadakane. Succinct data structure for path graphs. *Information and Computation*, 296:105124, 2024. doi:10.1016/j.ic.2023.105124.

5 N. Banerjee, S. Chakraborty, V. Raman, and S. R. Satti. Space efficient linear time algorithms for BFS, DFS and applications. *Theory Comput. Syst.*, 62(8):1736–1762, 2018. doi:10.1007/S00224-017-9841-2.

6 S. Chakraborty and S. Jo. Compact representation of interval graphs and circular-arc graphs of bounded degree and chromatic number. *Theor. Comput. Sci.*, 941:156–166, 2023.

7 S. Chakraborty, S. Jo, and S. R. Satti. Improved space-efficient linear time algorithms for some classical graph problems. *CoRR*, abs/1712.03349, 2017. arXiv:1712.03349.

8 S. Chaplick and J. Stacho. The vertex leafage of chordal graphs. *Discrete Applied Mathematics*, 168:14–25, 2014. Fifth Workshop on Graph Classes, Optimization, and Width Parameters, Daejeon, Korea, October 2011. doi:10.1016/j.dam.2012.12.006.

9 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

10 N. C.Wormald. Counting labelled chordal graphs. *Graph. Comb.*, 1(1):193–200, December 1985. doi:10.1007/BF02582944.

11 A. Elmasry, T. Hagerup, and F. Kammer. Space-efficient basic graph algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd STACS*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.STACS.2015.288.

12 F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs, 1973.

13 F. Gavril. A recognition algorithm for the intersection graphs of paths in trees, 1978.

14 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co., NLD, 2004.

15 A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: A tool for text indexing. In *SODA*, SODA '06, pages 368–373, USA, 2006. Society for Industrial and Applied Mathematics.

16 A. Itai and M. Rodeh. Representation of graphs. *Acta Inf.*, 17(2):215–219, June 1982. doi:10.1007/BF00288971.

17 G. J. Jacobson. Space-efficient static trees and graphs. *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, 1989.

18 I.J. Lin, T.A. McKee, and D.B. West. The leafage of a chordal graph. *Discussiones Mathematicae Graph Theory*, 18(1):23–48, 1998. URL: http://eudml.org/doc/270535.

19 L. Markenzon, C F. E. M. Waga, P. R. C Pereira, C. V. P. Friedmann, and A. R. G. Lozano. An efficient representation of chordal graphs. *Operations Research Letters*, 41(4):331–335, 2013. doi:10.1016/j.orl.2013.03.008.

20 C. L. Monma and V. K.-W. Wei. Intersection graphs of paths in a tree. *J. Comb. Theory, Ser. B*, 41(2):141–181, 1986.

21 J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012.

22 J. I. Munro and K. Wu. Succinct data structures for chordal graphs. In *ISAAC*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:12, 2018.

23 G. Navarro. *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016.

24 G. Navarro and K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3), May 2014. doi:10.1145/2601073.

Recognition and Proper Coloring of Unit Segment Intersection Graphs

Robert D. Barish[1]
Division of Medical Data Informatics, Human Genome Center,
Institute of Medical Science, University of Tokyo, Japan

Tetsuo Shibuya
Division of Medical Data Informatics, Human Genome Center,
Institute of Medical Science, University of Tokyo, Japan

Abstract

In this work, we concern ourselves with the fine-grained complexity of recognition and proper coloring problems on highly restricted classes of geometric intersection graphs of "thin" objects (i.e., objects with unbounded aspect ratios). As a point of motivation, we remark that there has been significant interest in finding algorithmic lower bounds for classic decision and optimization problems on these types of graphs, as they appear to escape the net of known planar or geometric separator theorems for "fat" objects (i.e., objects with bounded aspect ratios). In particular, letting n be the order of a geometric intersection graph, and assuming a geometric ply bound, per what is known as the "square root phenomenon", these separator theorems often imply the existence of $\mathcal{O}\left(2^{\left(\sqrt{n}\right)}\right)$ algorithms for problems ranging from finding proper colorings to finding Hamiltonian cycles. However, in contrast, it is known for instance that no $2^{o(n)}$ time algorithm can exist under the Exponential Time Hypothesis (ETH) for proper 6-coloring intersection graphs of line segments embedded in the plane (Biró et. al.; *J. Comput. Geom.* **9**(2); pp. 47–80; 2018).

We begin by establishing algorithmic lower bounds for proper k-coloring and recognition problems of intersection graphs of line segments embedded in the plane under the most stringent constraints possible that allow either problem to be non-trivial. In particular, we consider the class UNIT-PURE-k-DIR of unit segment geometric intersection graphs, in which segments are constrained to lie in at most k directions in the plane, and no two parallel segments are permitted to intersect. Here, under the ETH, we show for every $k \geq 3$ that no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for either recognizing or proper k-coloring UNIT-PURE-k-DIR graphs of order n. In addition, for every $k \geq 4$, we establish the same algorithmic lower bound under the ETH for the problem of proper $(k-1)$-coloring UNIT-PURE-k-DIR graphs when provided a list of segment coordinates specified using $\mathcal{O}(n \cdot k)$ bits witnessing graph class membership. As a consequence of our approach, we are also able to show that the problem of properly 3-coloring an arbitrary graph on m edges can be reduced in $\mathcal{O}\left(m^2\right)$ time to the problem of properly $(k-1)$-coloring a UNIT-PURE-k-DIR graph. Finally, we consider a slightly less constrained class of geometric intersection graphs of lines (of unbounded length) in which line-line intersections must occur on any one of $(r = 3)$ parallel planes in $\mathbb{R}^3$. In this context, for every $k \geq 3$, we show that no $2^{o(n/k)}$ time algorithm can exist for proper k-coloring these graphs unless the ETH is false.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Mathematics of computing → Graph coloring

Keywords and phrases graph class recognition, proper coloring, geometric intersection graph, segment intersection graph, fine-grained complexity, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.5

Funding *Tetsuo Shibuya*: This work was supported by JSPS Kakenhi grants {23H03345, 23K18501, 20H05967, 21H05052}.

[1] Corresponding author.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 5; pp. 5:1–5:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The notion of a geometric intersection graph, where vertices correspond to geometric shapes embedded in $\mathbb{R}^n$, for some $n \in \mathbb{N}_{>0}$, and edges encode their intersections, provides a direct bridge between topology and intuitive Euclidean geometry. In particular, fundamental graph and complexity theoretic questions concerning these objects tend to have answers with distinctly "physical" implications.

Nice examples of this phenomena come from the *graph recognition problem* of deciding if a given graph can be realized as a particular type of geometric intersection graph. With regard to positive results, we can note the proof by Koebe that every planar graph is realizable as an intersection graph of disks [32], as well as Chalopin & Gonçalves' proof [12] of Scheinerman's conjecture [61] that every planar graph is likewise realizable as an intersection graph of segments. We can also observe a proof by Pach & Tóth [52] that, of the $2^{\binom{n}{2}}$ graphs on n labeled vertices, at least $2^{\left(\left(\frac{3}{4}\right)\cdot\binom{n}{2}\right)}$ and at most $2^{\left(\left(\frac{3}{4}+o(1)\right)\cdot\binom{n}{2}\right)}$ can be realized as the intersection graphs of n Jordan curves in $\mathbb{R}^2$ (i.e., "string" graphs [19, 35, 62]). However, while it is known to be decidable whether a particular graph is a string graph [51, 59, 60], the problem was also shown to be NP-hard [33] and eventually NP-complete [58]. Similar hardness results exist for recognizing disk graphs [27], unit disk graphs [9], as well as angle-constrained segment [34, 37] and unit segment graphs [47].

Going further, we can ask questions that pertain to the nature of the realizations that are possible for a geometric intersection graph. Here, letting the *ply* or *thickness* of a geometric system correspond to the maximum number of objects intersecting at a common point, a rather remarkable finding has been that results analogous to the Lipton-Tarjan separator theorem [41] for planar graphs likewise exist for (typically bounded ply) geometric intersection graphs of "fat" objects (i.e., objects with bounded aspect ratios) [6, 20, 22, 46, 63], and in a much more limited sense, for "thin" objects (i.e., objects with unbounded aspect ratios) such as string graphs [23, 24, 25, 39, 45]. This, in turn, often leads to a "square root phenomena" for geometric intersection graphs, once again like that observed in planar graphs [43], of subexponential (e.g., $2^{\mathcal{O}\left(\sqrt{n}\right)}$) algorithms for problems ranging from independent set to Hamiltonian cycle where, say, only a $2^{\mathcal{O}(n)}$ algorithm might be known in the general case (see, e.g., [2, 6, 8, 15, 21, 24, 31, 42, 44, 49, 53]).

In this work, we concern ourselves with further exploring the gap between what is known concerning geometric separator theorems for intersection graphs of "fat" and "thin" objects. In particular, we proceed by deriving new algorithmic lower bounds for fundamental graph class recognition and proper coloring problems on geometric intersection graphs of "thin" objects under the most stringent possible constraints. We remark that proper coloring problems are of particular interest in this context, as a proper k-coloring of a given geometric intersection graph can only exist if the graph has ply-at-most-k.

Towards this objective, we first consider the problems of recognizing and proper k-coloring geometric intersection graphs in the class UNIT-PURE-k-DIR (generalizing graph classes discussed in ref. [11, 13, 50]), consisting of all geometric intersection graphs of unit length straight line segments, lying in at most k directions in the plane, where all parallel segments are disjoint. Subsequently, to obtain tighter lower bounds for related geometric intersection graphs of uniform length line segments, we consider the complexity of proper k-coloring geometric intersection graphs of lines (of unbounded length), in which all line-line intersections are required to occur on any one of three parallel planes in $\mathbb{R}^3$.

As a high level summary of our findings, extending a result of Mustaţă & Pergel [47] that recognizing UNIT-PURE-2-DIR graphs is NP-complete, and a result of Barish & Shibuya [4] that finding a proper 3-coloring of a UNIT-PURE-4-DIR is NP-complete, assuming the

Exponential Time Hypothesis (ETH) of Impagliazzo & Paturi [29] we show in part that: for every $k \geq 3$, no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for either recognizing or proper k-coloring order n UNIT-PURE-k-DIR graphs (Theorem 1); and for every $k \geq 4$, no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for finding a proper $(k-1)$-coloring of a UNIT-PURE-k-DIR graph, even when provided a list of segment coordinates, specified using $\mathcal{O}\left(n \cdot k\right)$ bits, witnessing graph class membership (Theorem 2). Here, as a partial consequence of these efforts, we are also able to extend a result of Barish & Shibuya [4] that the problem of proper 3-coloring an arbitrary graph on m edges can be reduced in $\mathcal{O}\left(m^2\right)$ time[2] to the problem of properly 3-coloring a UNIT-PURE-4-DIR graph. In particular, we show for every $k \geq 4$ that the same proper 3-coloring problem can likewise be reduced to a proper $(k-1)$-coloring problem for UNIT-PURE-k-DIR graphs in $\mathcal{O}\left(m^2 \cdot k\right)$ time (Corollary 1).

Finally, assuming the ETH, for every $k \geq 3$, we show that no $2^{o(n/k)}$ time algorithm can exist for proper k-coloring order n geometric intersection graphs of lines in which we require that line-line intersections must occur on any one of three parallel planes in $\mathbb{R}^3$ (Theorem 3).

2 Elaboration concerning motivation

An important point of motivation for the current work comes from a question posed by Miltzow – in the workshop on Graph Classes, Optimization, and Width Parameters (GROW) list of open problems [56] – concerning the lower bound complexity under the ETH of proper k-coloring geometric intersection graphs of unit segments in the plane.

As noted by Miltzow, the difficulty in answering this question has been due, in part, to the failure to extend geometric separator theorems for bounded ply intersection graphs of "fat" objects to bounded ply intersection graphs of "thin" objects. Here, this gap becomes readily apparent when looking at algorithmic lower bounds for proper k-coloring problems. In particular, specifying $k \in \Theta\left(n^{\alpha}\right)$ for some $0 \leq \alpha \leq 1$, Biró et. al. [6] was able to show the existence of a $2^{\mathcal{O}\left(\sqrt{n} \cdot k \cdot \ln n\right)}$ algorithm for finding a proper k-coloring of intersection graphs of disks or other "fat" objects, while also showing that the existence of a $2^{o\left(\sqrt{n/k}\right)}$ algorithm would refute the ETH. On the other hand, Biró et. al. [6] was also able to establish that no $2^{o(n)}$ algorithm can exist for proper 6-coloring 2-DIR graphs (though not PURE-2-DIR graphs) assuming the ETH, even if all segments are constrained to lie at angles of 0 and $\frac{\pi}{2}$ radians in the plane. This latter result was later strengthened and extended by Bonnet & Rzążewski [8], who were able to establish a $2^{o(n)}$ (resp. $2^{o\left(n^{2/3}\right)}$) lower bound for proper k-coloring 2-DIR graphs (resp. 3-DIR graphs with unit length segments) for any constant $k \geq 4$, as well as the existence of a subexponential time $2^{\tilde{\mathcal{O}}\left(n^{2/3}\right)}$ algorithm for finding a proper $(k = 3)$-coloring.

To elaborate on our choice to focus on the class UNIT-PURE-k-DIR of geometric intersection graphs, this first of all represents a limit case for geometric intersection graphs of "thin" objects embedded in the plane with orientation and length constraints. More specifically, let "string" [19, 35, 62] be the earlier defined class of geometric intersection graphs of Jordan curves, let "CONV" [33, 36, 57, 64] be the class of all geometric intersection graphs of convex shapes, let "SEG" [19, 33, 34, 37, 38] be the class of all geometric intersection graphs of straight line segments in the plane, let k-DIR [34, 37, 38] be a subclass of "SEG" where segments must lie in at most k directions, and let PURE-k-DIR [34, 37, 38] be a subclass of k-DIR where all parallel segments are required to be disjoint. We can now observe

[2] This was erroneously reported to be an $\mathcal{O}\left(m\right)$ time reduction in Barish & Shibuya [4].

that, for every $k \geq 1$, PURE-k-DIR $\subsetneq$ k-DIR $\subsetneq$ SEG $\subsetneq$ CONV $\subsetneq$ STRING [19, 37], that k-DIR $\subsetneq$ $(k+1)$-DIR [37], and that PURE-k-DIR $\subsetneq$ PURE-$(k+1)$-DIR [37]. Second of all, as intersections between parallel segments are prohibited for this graph class, an analysis of proper k-coloring problems for this graph class requires fundamentally different techniques that of either Biró et. al. [6] or Bonnet & Rzążewski [8], and we considered this to be an interesting challenge.

To elaborate on our choice to consider geometric intersection graphs of lines, in which we require that line-line intersections must occur on any one of three parallel planes in $\mathbb{R}^3$, this class of graphs can be understood as a weak-as-possible generalization of the type of geometric intersection graphs of unit segments in the plane for which we were able to obtain results. Here, we hope our ability to rule out the existence of $2^{o(n/k)}$ time algorithms for proper k-coloring order n instances of these graphs under the ETH, for all $k \geq 3$, can be extended to establish the same algorithmic lower bound for intersection graphs of unit segments in the plane for some $k \geq 3$.

Finally, we remark that the problem of proper k-coloring UNIT-PURE-k-DIR graphs has practical application to problems ranging from realistic instances of the frequency assignment problem [1, 16, 26], to the design of Very Large Scale Integration (VLSI) circuits [17, 48] (e.g., where overlapping wires (segments) must be assigned to distinct circuit layers abstracted as colors). To briefly elaborate on the former case, the frequency assignment problem asks one to assign a sparse set of frequency bands (colors) to a set of antennas (vertices) in an *interference graph*, where we have that two vertices are adjacent if and only if they correspond to antennas spaced closely enough to interfere when emitting within the same frequency band (e.g., within $\approx 50 - 100$ kHz [16]). Here, in a realistic scenario, any such interference graph will correspond to a geometric intersection graph between radiation emission patterns for antennas, which in many cases (e.g., coastal radio stations) are optimized to be narrow oriented cones approximating bounded length segments.

3 Preliminaries & clarifications

3.1 Graph theoretic terminology

All graphs in this work should be considered to be simple (i.e., loop and multi-edge free), undirected, and unweighted. Concerning basic graph theoretic terminology, we will generally follow Diestel [18], or where appropriate, Bondy & Murty [7]. However, for some brief clarifications, recall that a graph is k-connected if there exist k vertex disjoint simple paths between all pairs of vertices, and recall that a graph is *planar* if it admits an embedding in the plane without edge crossings. In addition, recall that a *proper k-coloring* for a graph is an assignment of $\leq k$ colors to the graph's vertices under the constraint that no two adjacent vertices have an identical coloration, and that the *chromatic number* for a graph is the minimum value of k for which it admits a proper k-coloring.

Concerning less common terminology, when we *identify* a vertex v_a with a vertex v_b, we delete v_a and v_b and create a new vertex v_c adjacent to any vertex formerly adjacent to v_a or v_b. Additionally, when we refer to a vertex v in a drawing or embedding of a graph G as a *metavertex* (e.g., corresponding to a clique of some size), it should be understood that v corresponds to an induced subgraph H of G, where every vertex in G adjacent to v is adjacent to each vertex of H. Here, we can also identify pairs of metavertices by identifying each pair of equivalent vertices in an isomorphism between the subgraphs they correspond to.

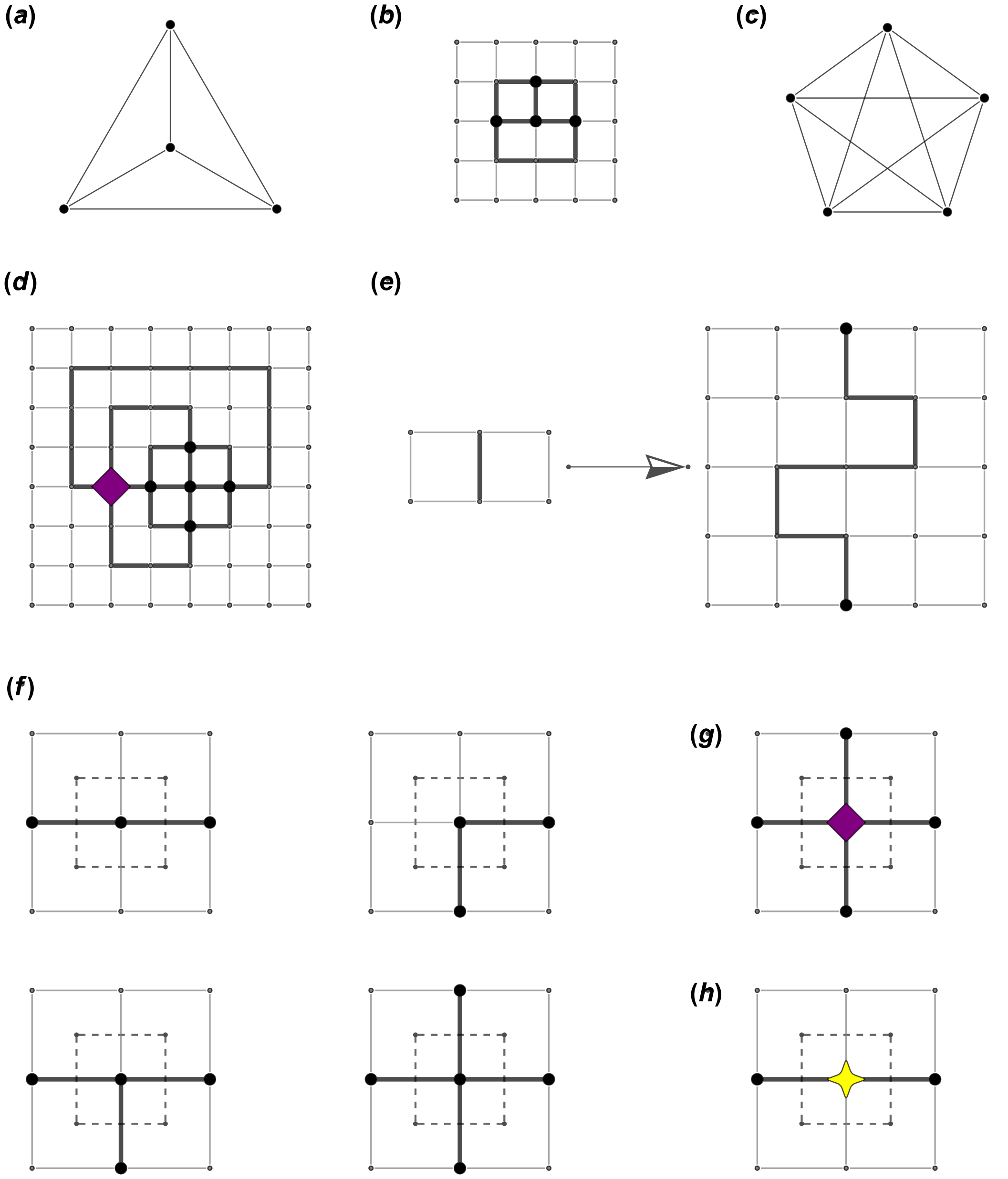


Figure 1 Illustrations of orthogonal integer lattice embeddings of graphs, where larger (black) vertices and (highlighted black) edges indicate the vertices and polylines for the embedding, respectively; **(a)** the complete graph K_4; **(b)** orthogonal integer lattice embedding of K_4; **(c)** the complete graph K_5; **(d)** orthogonal integer lattice embedding of K_5 with a lone polyline crossing indicated by a (purple) diamond polygon; **(e)** scheme for the enlargement and modification of an orthogonal integer lattice embedding to ensure all polylines have horizontal segments; **(f)** all local vertex and polyline configurations (up to rotation and reflection) in an orthogonal integer lattice embedding of a 2-connected graph of maximum vertex degree ≤ 4, where cells around each lattice vertex are indicated by a (dashed) box; **(g)** a cell containing a polyline crossing indicated by a (purple) diamond polygon; **(h)** a cell marked with a (yellow) concave diamond marker designating it to be identified with a "color change" gadget.

3.2 Exponential Time Hypothesis (ETH)

Recalling that k-SAT is the problem of deciding the satisfiability of a Boolean expression in conjunctive normal form where each clause contains at most k literals, the *Exponential Time Hypothesis* (ETH) of Impagliazzo & Paturi [29] can be defined as follows:

▶ **Definition 1.** *Exponential Time Hypothesis (ETH) [29]. Assuming $k \geq 3$, letting n and m be the number of variables and clauses for an instance of k-SAT, and letting $s_k = inf\{\delta\ :\ k$-SAT can be solved in $2^{(\delta \cdot n)} \cdot poly\,(m)$ time$\}$, it holds that $s_k > 0$.*

3.3 Linear time orthogonal integer lattice embeddings of graphs

Let G be an arbitrary not-necessarily-planar graph of maximum vertex degree ≤ 4, with vertex set V_G and edge set E_G. An orthogonal integer lattice embedding (or drawing) $\mathcal{Q}$ of G places each vertex at a distinct integral coordinate, and represents each edge $v_i \leftrightarrow v_j \in E_G$ as a *polyline* consisting of a polygonal chain of axis-parallel unit length horizontal and vertical segments – corresponding to a simple path in the integer lattice into which G is embedded – connecting v_i and v_j in $\mathcal{Q}$. If G is non-planar, then we necessarily must allow for polyline crossings in the embedding $\mathcal{Q}$. Here, a *bend* in $\mathcal{Q}$ corresponds to an instance where a horizontal and a vertical segment meet in a polyline (i.e., an instance where two unit segments meet at an angle of $\frac{\pi}{2}$ radians). We remark that it is possible to find an orthogonal integer lattice embedding for G on a square integer lattice, of total area $\mathcal{O}\left(|V_G|^2\right)$, in $\mathcal{O}\left(|V_G|\right)$ time via either the method of Papakostas & Tollis [54] or Biedl & Kant [5].

For illustrative examples, we refer the reader to Fig. 1(a–d), where in Fig. 1(a) (resp. Fig. 1(c)) we show an instance of the complete graph K_4 (resp. K_5), and in Fig. 1(b) (resp. Fig. 1(d)) we show an orthogonal integer lattice embedding of the graph in a 5×5 (resp. 8×8) integer lattice. For the Fig. 1(d) embedding of the graph K_5, we can observe that one polyline crossing occurs, where we indicate this crossing via a (purple) diamond polygon. Additionally, in Fig. 1(f) we show all possible local polyline configurations in a cell, up to rotation and reflection, under the assumption that the embedded graph is at least 2-connected.

4 Recognition and proper coloring of UNIT-PURE-k-DIR graphs

In this section, we establish Theorem 1 through Theorem 3, as well as Corollary 1. Concerning Theorem 1, we briefly remark that previous reductions for proving the NP-hardness of recognizing "string" graphs [33], k-DIR and PURE-k-DIR graphs [34, 37], and UNIT-PURE-2-DIR graphs [47], have generally followed a strategy of giving an $\mathcal{O}\left(n^2\right)$ reduction from a variant of planar 3-SAT already admitting a subexponential time algorithm. Accordingly, we required a different approach for establishing our claims.

▶ **Theorem 1.** *Unless the ETH is false, for any $k \geq 3$, no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for either recognizing or proper k-coloring order n UNIT-PURE-k-DIR graphs.*

Proof. We proceed by giving an $\mathcal{O}\left(n^2 \cdot k\right)$ reduction from the problem of finding a proper 3-coloring of a 2-connected graph with n vertices and maximum vertex degree ≤ 4. Here, it is known that no $2^{o(n)}$ time algorithm can exist for this problem unless the ETH fails (see, e.g., "Lemma 2.1" of [14]). In particular, let G and H be a pair of graphs with vertex sets V_G and V_H, respectively, where $n = |V_G|$, G is an arbitrary 4-regular graph, and H is a graph constructable from G in $\mathcal{O}\left(n^2 \cdot k\right)$ time with $|V_H| = \mathcal{O}\left(n^2 \cdot k\right)$ vertices. We will

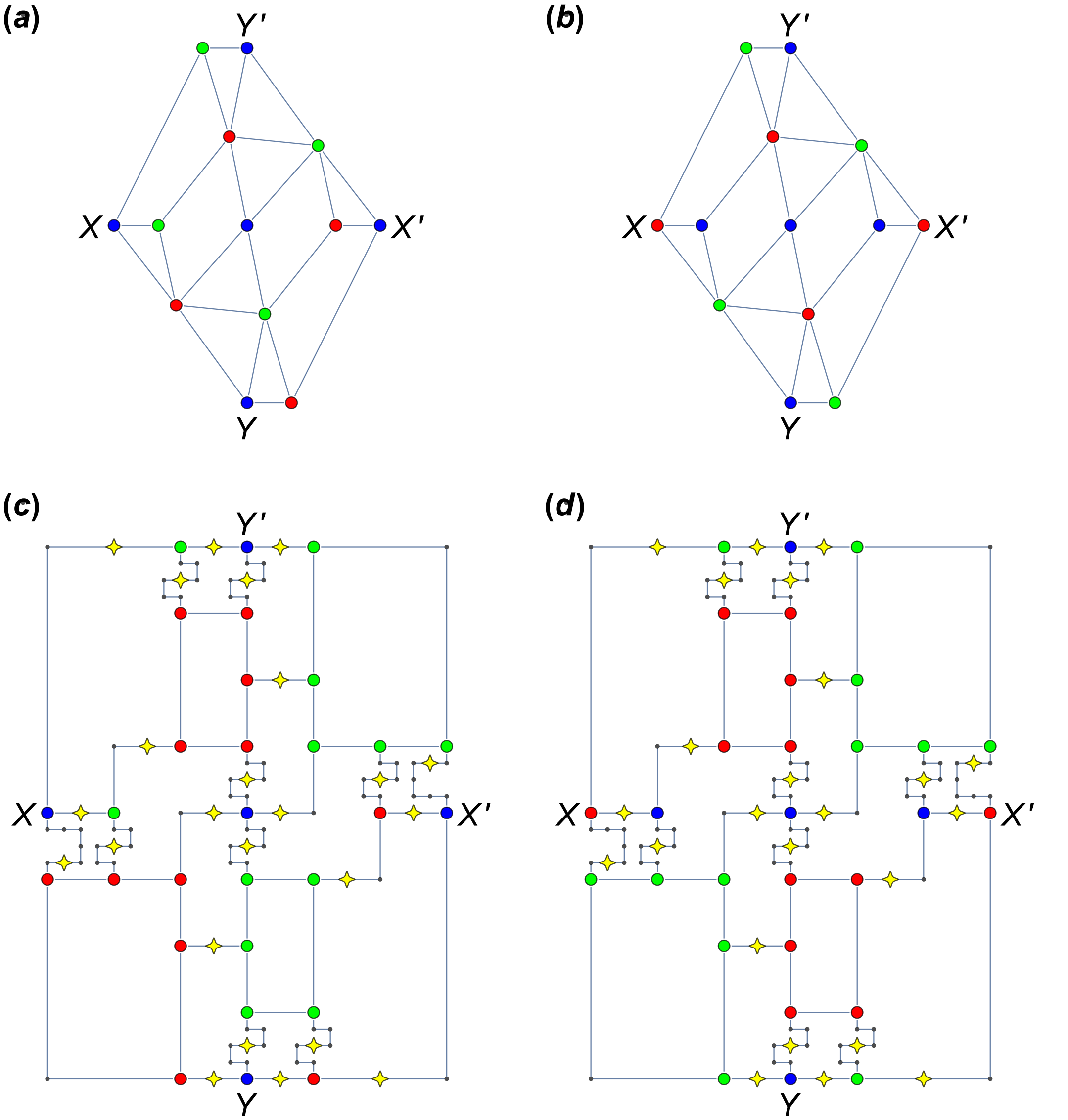


Figure 2 Illustration and example proper colorings of a novel proper 3-coloring planarization gadget, and scheme for its adaption as the gadget Υ in the context of an orthogonal integer lattice embedding; **(a)** the order 13 and size 24 planarization gadget, where vertices labeled X and X' (resp. Y and Y') are required to have the same coloration, with an example proper 3-coloring where vertices labeled X and Y are assigned the same color; **(b)** an alternative 3-coloration of the gadget, where the vertices labeled X and Y are assigned distinct colors; **(c)** orthogonal 25×33 integer lattice embedding of a modification of the gadget shown in (a,b) – denoted Υ – where some vertices in the planarization gadget from (a,b) are replaced with connected subgraphs, where such subgraphs are colored in accordance with the proper 3-coloring from the illustration of the gadget in (a), and where exactly one cell along each polyline between distinct subgraphs is marked for identification with the “color change” gadget; **(d)** another coloring of Υ in accordance with the proper 3-coloring of the planarization gadget from (b).

show for every $k \geq 3$ that G admits a proper 3-coloring if and only if H is proper k-colorable, or equivalently in this specific context, realizable as a UNIT-PURE-k-DIR graph. From this we will deduce that a $2^{o\left(\sqrt{n/k}\right)}$ algorithm for either recognizing or proper k-coloring a UNIT-PURE-k-DIR graph would refute the ETH.

Provided the instance of the aforementioned graph G, we begin by computing an orthogonal integer lattice embedding $\mathcal{Q}_1$ for G, of total area $\mathcal{O}\left(n^2\right)$, in $\mathcal{O}\left(n\right)$ time via either the method of Papakostas & Tollis [54] or Biedl & Kant [5] (see, e.g., Section 3.3 for an elaboration on these embeddings). In this context, we define a *cell* in an orthogonal integer lattice embedding $\mathcal{Q}_1$ to be a square area of volume 1 centered on each lattice point in $\mathcal{Q}_1$. For instance, the boundaries of this square area for a lattice point at coordinates $(x, y) \in \mathbb{Z}^2$ would be given by the coordinates $\left(\left(x - \frac{1}{2}, y - \frac{1}{2}\right), \left(x + \frac{1}{2}, y - \frac{1}{2}\right), \left(x + \frac{1}{2}, y + \frac{1}{2}\right), \left(x - \frac{1}{2}, y + \frac{1}{2}\right)\right)$. Letting the Manhattan distance between a pair of cells correspond to the Manhattan distance between their respective lattice points, we also define the *von Neumann neighborhood* of a cell as the set containing both the cell itself as well as its four neighbors at distance 1. For a cell with a centerpoint at some coordinate $(i, j) \in \mathbb{Z}^2$, we refer to its distance 1 neighbors at coordinates $(i, j + 1)$, $(i - 1, j)$, $(i + 1, j)$, and $(i, j - 1)$ as being to the North, West, East, and South, respectively.

We next construct an orthogonal integer lattice embedding $\mathcal{Q}_3$ from $\mathcal{Q}_1$ – via an intermediate graph $\mathcal{Q}_2$ potentially containing polyline crossings – satisfying the dual constraints that: (constraint 1) each polyline originally in $\mathcal{Q}_1$ has at least one horizontal segment marking the position of a "color change" gadget with a (yellow) concave diamond marker (e.g, as shown in Fig. 1(h)); and (constraint 2) each polyline crossing is replaced with an orthogonal integer lattice embedding of a gadget such that $\mathcal{Q}_3$ has no polyline crossings, and in addition, its corresponding graph is proper 3-colorable if and only if the graph corresponding to the embedding $\mathcal{Q}_1$ is proper 3-colorable.

To address (constraint 1), we begin by checking if every polyline contains at least one horizontal segment, and subsequently marking this horizontal segment if it exists. For any remaining polylines consisting of only vertical segments, we can perform the operation shown in Fig. 1(e) to introduce and subsequently mark a horizontal segment. We note that this may require expanding the embedding $\mathcal{Q}_1$ by moving each point at a position (x, y) to a position $(2x, 4y)$, treating expanded polyline edges as chains of unit length segments. Here, we call the resulting embedding $\mathcal{Q}_2$.

To address (constraint 2), let Ψ be the subset of cells in $\mathcal{Q}_2$ corresponding to the type of polyline crossing indicated by the (purple) diamond polygon in Fig. 1(g). If $\Psi = \emptyset$, we can specify $\mathcal{Q}_3 = \mathcal{Q}_2$. If $\Psi \neq \emptyset$, we proceed by substituting each polyline crossing with an orthogonal integer lattice embedding of a gadget, denoted Υ, having the same properties as the novel proper 3-coloring planarization gadget shown in Fig. 2(a,b) (note that Fig. 2(a) and Fig. 2(b) are identical aside from having distinct vertex colorations), where these properties are given by the following lemma:

▶ **Lemma 1.** *The proper* 3*-coloring planarization gadget shown in Fig. 2(a,b) has the following properties: (1) its chromatic number is* 3*; (2) any proper* 3*-coloring will assign identical colors to the vertices labeled X and X' as well as the vertices labeled Y and Y'.*

Proof. Let W be a graph isomorphic to the 13 vertex and 24 edge gadget shown in Fig. 2(a,b). To establish properties (1) and (2), it suffices to first evaluate the chromatic polynomial $P\left(W, k\right)$ of the Fig. 2(a,b) gadget, check that $P\left(W, 2\right) = 0$, observe that $P\left(W, 3\right) = 12$, and then inspect each of the 12 possible proper 3-colorings to confirm property (2). Here, noting that there are at most $3^{13} = 1594323$ possible vertex colorings to check, we used brute force methods to enumerate all 12 possible proper 3-colorings. We briefly remark that, while a more insightful proof is possible, we were unable to find one of a short enough length to include in the current context. ◀

In particular, we specify Υ as the 25×33 cell construction shown in Fig. 2(c,d) (note that Fig. 2(c) and Fig. 2(d) are identical aside from having distinct vertex colorations). As in the case of the Fig. 2(a,b) graph, and as we will see in Lemma 2, the vertices labeled X and X' (respectively, Y and Y') in the graph corresponding to the Fig. 2(c,d) construction are likewise forced to be the same in any proper 3-coloring. Here, to replace polyline crossings in $\mathcal{Q}_2$ with Υ, we can expand the embedding $\mathcal{Q}_2$ by moving each point at a position (x, y) to a position $(25x, 33y)$, then replace each 25×33 block of cells centered on a polyline crossing with Υ. Observe that this will serve to ensure that the outgoing polylines to the North, West, East, and South cells in the von Neumann neighborhood of each cell in Ψ are connected to the Υ gadget vertices labeled Y', X, X', and Y, respectively. In this context, we let Φ correspond to all cells marking the position of a "color change" gadget with a (yellow) concave diamond marker in $\mathcal{Q}_3$, where Φ includes the markers shown in Fig. 2(c,d).

Next, for each cell $c \notin \Phi$ containing an endpoint of a polyline, we replace a $\left(\frac{103}{50}\right) \times 3$ distortion of the cell with an appropriate version of the embedded "color copying" gadget shown in Fig. 3. We note that (purple) nodes in this gadget represent metavertices corresponding to cliques of size $(k - 3)$. More specifically, for a given cell $c \notin \Phi$, letting ζ_c be an instance of the Fig. 3 graph, we generate a graph ζ'_c by: (case N) deleting the vertices $\{N_1, N_2\}$ and all vertices embedded above (i.e., with a larger y coordinate) if and only if c *does not* have an outgoing polyline segment to the North in its von Neumann neighborhood; (case W) deleting the vertices $\{W_1, W_2\}$ and all vertices embedded to the left (i.e., with a smaller x coordinate) if and only if c does not have an outgoing polyline segment to the West in its von Neumann neighborhood; (case E) deleting the vertices $\{E_1, E_2\}$ and all vertices embedded to the right (i.e., with a larger x coordinate) if and only if c does not have an outgoing polyline segment to the East in its von Neumann neighborhood; and (case S) deleting the vertices $\{S_1, S_2\}$ and all vertices embedded below (i.e., with a smaller y coordinate) if and only if c does not have an outgoing polyline segment to the South in its von Neumann neighborhood. To complete the construction of H, we then embed ζ' on the aforementioned $\left(\frac{103}{50}\right) \times 3$ distortion of the cell $c \notin \Phi$ in exactly the manner shown in Fig. 3, embed an instance η_c of the "color change" gadget shown in Fig. 4 on a $\left(\frac{103}{50}\right) \times 3$ distortion of each cell $c \in \Phi$ – with (purple) metavertices corresponding to cliques of size $(k - 3)$ as in Fig. 3 – and identify any vertices or metavertices from the borders of adjacent cells mapped to the same coordinates (see Section 3.1 for an elaboration on metavertex identification).

We can now observe the following lemma:

▶ **Lemma 2.** *The graph H admits a proper k-coloring if and only if the graph G admits a proper 3-coloring.*

Proof. Assume the definitions previously given in the proof argument for Theorem 1.

First consider the case where G is planar, and no copies of the gadget Υ were embedded during the process of generating $\mathcal{Q}_3$ from $\mathcal{Q}_1$. Here, we have that H will be constructed from the embedding $\mathcal{Q}_3$ by: (1) replacing exactly one cell falling along each polyline at a position where exactly two polyline segment endpoints coincide with the "color change" gadget; and (2) replacing all remaining cells hosting polyline segments with a "color copying" gadget η in such a manner that the vertices labeled X_1 and metavertices labeled X_2 in Fig. 3 will be present on the North, West, East, and South boundaries of each cell's von Neumann neighborhood if and only if the cell has a polyline segment egressing from its North,

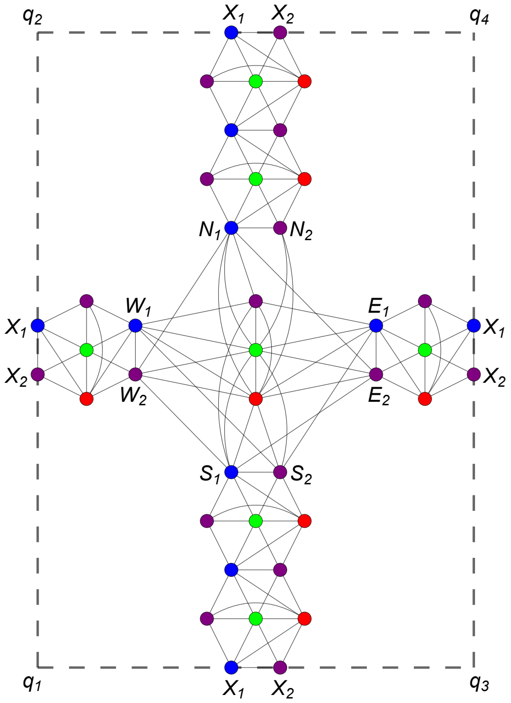


Figure 3 Illustration of the "color copying" gadget – and scheme for the gadget's placement on a "distorted" $\left(\frac{103}{50}\right) \times 3$ cell (dashed box) – where (purple) vertices (e.g., labeled X_2) are metavertices corresponding to cliques of size $(k-3)$.

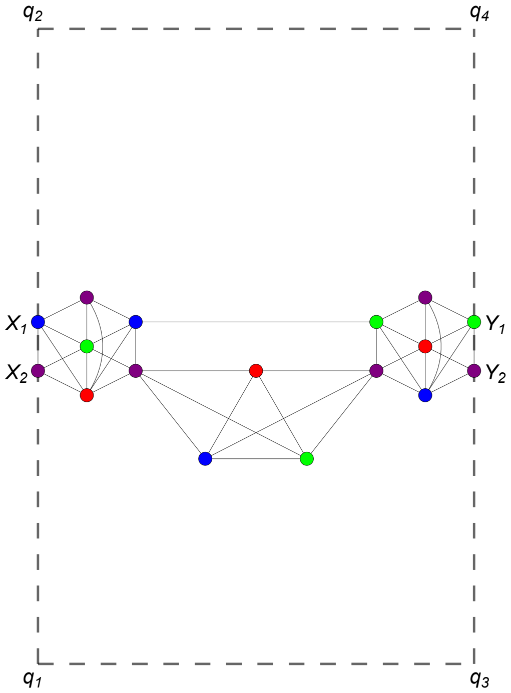


Figure 4 Illustration of the "color change" gadget – and scheme for the gadget's placement on a "distorted" $\left(\frac{103}{50}\right) \times 3$ cell (dashed box) – where (purple) vertices (e.g., labeled X_2 or Y_2) are metavertices corresponding to cliques of size $(k-3)$.

West, East, and South boundaries, respectively. Accordingly, each cell in $\mathcal{Q}_3$ corresponding to a vertex in G of degree d will, in turn, correspond to a specific "color copying" gadget with d copies of the vertices labeled X_1 and X_2. Furthermore, all but one of the cells corresponding to part of a polyline connecting vertices in G will, on some pair of boundaries, have exactly two copies of vertices labeled X_1 and two copies of metavertices labeled X_2, with the remaining cell corresponding to a "color change" gadget.

To now show that H admits a proper k-coloring if and only if the planar instance of G admits a proper 3-coloring, it suffices to observe for any proper k-coloring of H that: (requirement 1) the vertices labeled X_1 in every "color copying" gadget must have the same coloration; and (requirement 2) the vertices labeled X_1 and Y_1 in every "color change" gadget must have a distinct coloration.

Concerning (requirement 1), consider first the case where $k = 3$. Here, for every possible instance of the gadget ζ_c', we can check the chromatic polynomial $P\left(\zeta_c', k\right)$ with parameter k, or enumerate all possible proper 3-colorings via brute force, observe that $P\left(\zeta_c', 2\right) = 0$ and $P\left(\zeta_c', 3\right) \neq 0$ (e.g., $\zeta_c = \zeta_c' \implies P\left(\zeta_c', 3\right) = 384$), and furthermore check that the constraint is satisfied in each of these cases. To address cases where $k \geq 4$, it suffices to observe that, for any instance of ζ_c' (recalling that the metavertices shown in Fig. 3 correspond to cliques of size $(k-3)$), every vertex will necessarily belong to a clique of size k. Accordingly, if (1) is satisfied in the case where $k = 3$, it will likewise be satisfied in the case where $k = 4$ and we are forced to assign the additional color to the single vertex corresponding to each metavertex, satisfied in the case where $k = 5$ and we are required to assign the two additional colors to the two vertices corresponding to each metavertex, and by induction, satisfied for every $k \geq 3$.

Concerning (requirement 2), we proceed in a similar manner. In particular, letting η_n be an instance of the "color change" gadget in the case where $k = 3$, we observe that $P\left(\eta_n, 2\right) = 0$ and $P\left(\eta_n, 3\right) = 144$. With exactly the same inductive argument used to address (requirement 1), we can then show that (requirement 2) will hold for every $k \geq 3$.

As we have now seen that (requirement 1) and (requirement 2) will be satisfied for every $k \geq 3$, this yields the lemma in the case where G is planar.

In the case where G is non-planar, it suffices to observe that the Υ construction simply replaces certain vertices in the Fig. 2(a,b) proper 3-coloring planarization gadget with connected subgraphs $s_1, s_2, \ldots$, then places the "color change" gadget on polylines if and only if they connect distinct subgraphs. Accordingly, following our earlier argument, the "color change" gadget will conceptually force each of the embedded vertices in the same subgraph s_i in Υ to have an identical coloration, and each pair of adjacent subgraphs s_i and s_j in Υ to have distinct colorations. In observation of Lemma 1, this yields the current lemma in the case where G is non-planar. ◀

We can also observe that the construction of H from an initial order n graph G takes at most $\mathcal{O}\left(n^2\right)$ time as a consequence of the orthogonal integer lattice embedding $\mathcal{Q}_1$ having at most $\mathcal{O}\left(n^2\right)$ cells hosting at least one endpoint of a polyline, where the embedding of the "color copying" and "color change" gadgets on each cell hosting a polyline then increases the time complexity of the construction to $\mathcal{O}\left(n^2 \cdot k\right)$. This together with Lemma 2 implies that, as a consequence of the fact that no $2^{o(\sqrt{n_G})}$ time algorithm can exist for proper 3-coloring an order n_G instance of the graph G under the ETH, we likewise have that no $2^{o(\sqrt{n_H \cdot k})}$ time algorithm can exist for proper k-coloring an order n_H instance of the constructed graph H unless the ETH is false.

The subsequent step of this proof argument, which is simultaneously the most technically difficult and easiest to describe, is to show that H can be realized as a UNIT-PURE-k-DIR graph if and only if H admits a proper k-coloring. Here, we can begin by observing that, due to the requirement no parallel segments may intersect, any proper k-coloring for a UNIT-PURE-k-DIR can be understood as an assignment of embedding angles for the segments of a geometric system of intersecting unit segments witnessing graph class membership.

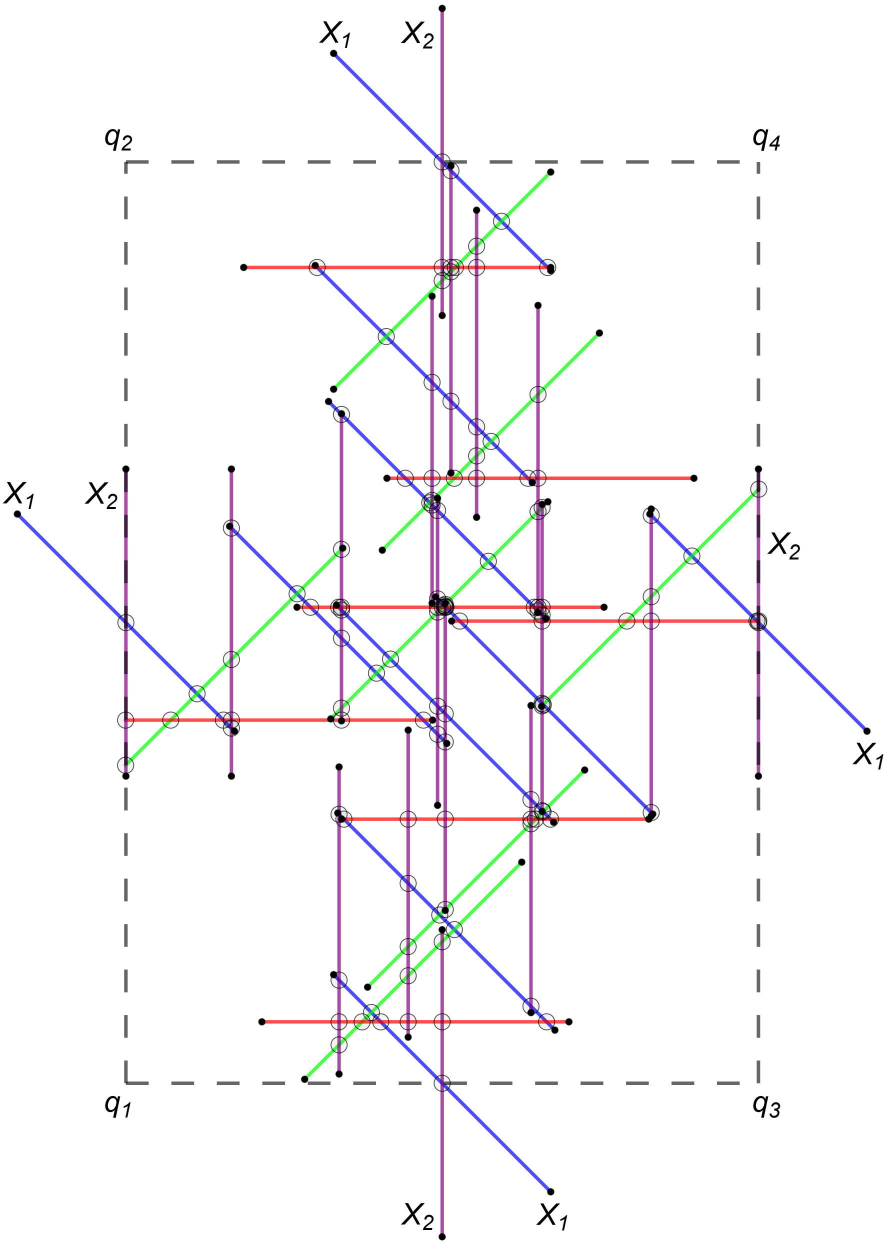


Figure 5 UNIT-PURE-k-DIR realization (for every $k \geq 3$) of the "color copying" gadget – and scheme for the gadget's placement on a "distorted" $\left(\frac{103}{50}\right) \times 3$ cell (dashed box) – where (purple) segments (e.g., labeled X_2) correspond to $(k-3)$ overlapping segments embedded in the plane at distinct infinitesimal perturbations of $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_{(k-3)}$ from $\frac{\pi}{2}$ radians, and where segments corresponding to vertices labeled X_1 in Fig. 3 are embedded in the plane at an angle of $-\frac{\pi}{4}$ radians (respectively, $\frac{\pi}{4}$ radians in a reflection of the embedding across the y-axis); segment-segment intersections are denoted with a (hollow) circle; note that no endpoint of a line segment is ever embedded along another line segment.

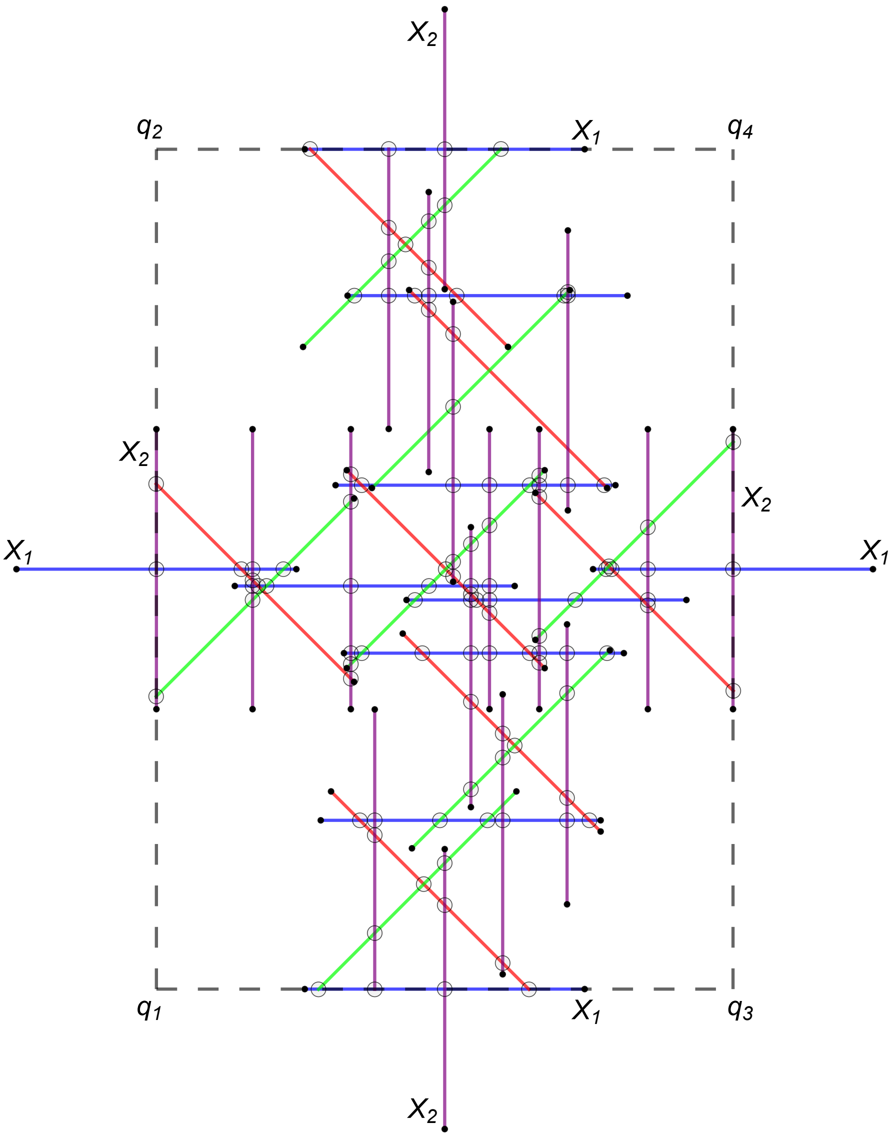


Figure 6 Alternative UNIT-PURE-k-DIR realization (for every $k \geq 3$) of the "color copying" gadget from the Theorem 1 proof argument – and scheme for the gadget's placement on a "distorted" $\left(\frac{103}{50}\right) \times 3$ cell (dashed box) – where (purple) segments (e.g., labeled X_2) correspond to $(k-3)$ overlapping segments embedded in the plane at distinct infinitesimal perturbations of $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_{(k-3)}$ from $\frac{\pi}{2}$ radians, and segments corresponding to vertices labeled X_1 in Fig. 3 are embedded in the plane at an angle of 0 radians; segment-segment intersections are denoted with a (hollow) circle; note that no endpoint of a line segment is ever embedded along another line segment.

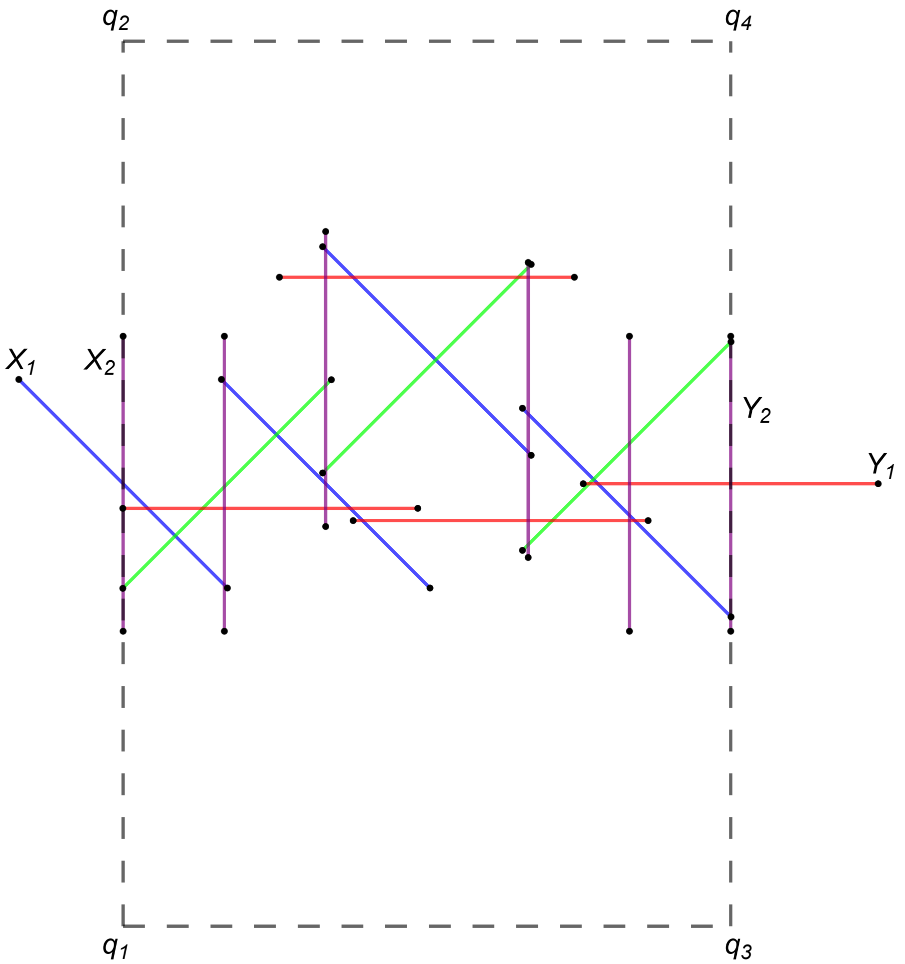


Figure 7 UNIT-PURE-k-DIR realization (for every $k \geq 3$) of the "color change" gadget from the Theorem 1 proof argument – and scheme for the gadget's placement on a "distorted" $\left(\frac{103}{50}\right) \times 3$ cell (dashed box) – where (purple) segments (e.g., labeled X_2) correspond to $(k-3)$ overlapping segments embedded in the plane at distinct infinitesimal perturbations of $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_{(k-3)}$ from $\frac{\pi}{2}$ radians, and where segments corresponding to vertices labeled X_1 and Y_1 in Fig. 4 are, respectively, embedded in the plane at angles of $-\frac{\pi}{4}$ and 0 radians, $\frac{\pi}{4}$ and 0 radians after reflection across x-axis, 0 and $\frac{\pi}{4}$ radians after reflection across y-axis, and 0 and $-\frac{\pi}{4}$ radians after reflection across both the x-axis and y-axis; note that no endpoint of a line segment is ever embedded along another line segment.

We now observe that the Fig. 3 "color copying" gadget can be realized as a UNIT-PURE-k-DIR graph with three distinct segment angles for the vertices labeled X_1 on each of the four borders of the "distorted" (i.e., non-square) $\left(\frac{103}{50}\right) \times 3$ cell embedding the gadget. In particular, we refer the reader to the UNIT-PURE-k-DIR realizations of this gadget shown in Fig. 5 and Fig. 6, where in both cases (purple) segments (e.g., labeled X_2) correspond to $(k-3)$ overlapping segments embedded in the plane at distinct infinitesimal perturbations of $\epsilon_1 < \epsilon_2 < \ldots < \epsilon_{(k-3)}$ from $\frac{\pi}{2}$ radians. Note that, as no endpoint of a segment is ever embedded along another line segment, these infinitesimal perturbations can always be chosen to be small enough to not change the set of segment-segment intersections. For Fig. 5, we can observe that segments corresponding to vertices labeled X_1 in Fig. 3 are embedded in the plane at an angle of $-\frac{\pi}{4}$ radians (respectively, $\frac{\pi}{4}$ radians in a reflection of the embedding across the y-axis). For Fig. 6, we can observe that segments corresponding to vertices labeled X_1 in Fig. 3 are embedded in the plane at an angle of 0 radians. This covers each of the three necessary cases.

Similarly, the Fig. 4 "color change" gadget can be realized as a UNIT-PURE-k-DIR graph with all possible combinations of three distinct segment angles for the vertices labeled X_1 and Y_1 on West and East border, respectively, of the "distorted" (i.e., non-square) $\left(\frac{103}{50}\right) \times 3$ cell embedding the gadget, with the segments labeled X_2 and Y_2 embedded in the same manner as the (purple) segments in Fig. 5 and Fig. 6. While we are unable to show all possible UNIT-PURE-k-DIR realizations of this gadget due to space constraints, we refer the reader to Fig. 7 for an illustration of the case where segments corresponding to the vertices labeled X_1 and Y_1 in Fig. 4 are embedded in the plane at angles of $-\frac{\pi}{4}$ and 0 radians, respectively.

Putting everything together yields that, assuming the ETH, no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for recognizing order n UNIT-PURE-k-DIR graphs. It remains to deduce that this directly implies no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for proper k-coloring an order n UNIT-PURE-k-DIR graph under the ETH. Here, simply observe that if such a $2^{o\left(\sqrt{n/k}\right)}$ time coloring algorithm $\mathcal{A}$ exists, there will likewise exist some upperbound for its run time of the form $2^{\left(\sqrt{n}\cdot k-\epsilon\right)}$ for some constant $\epsilon \in \mathbb{R}_{>0}$ and n sufficiently large. Accordingly, we could simply run $\mathcal{A}$ for $2^{\left(\sqrt{n}\cdot k-\epsilon\right)}$ time steps on the graph H, and if no proper k-coloring is found, determine that no such proper k-coloring exists. However, as H has a proper k-coloring if and only if it is a UNIT-PURE-k-DIR graph, this implies the existence of a $2^{o\left(\sqrt{n/k}\right)}$ time algorithm for recognizing H as a UNIT-PURE-k-DIR graph. Therefore, by our earlier arguments, no such algorithm $\mathcal{A}$ can exist under the ETH. ◀

▶ **Theorem 2.** *For any $k \geq 4$, provided a UNIT-PURE-k-DIR graph and a list of segment coordinates specified using $\mathcal{O}(n \cdot k)$ bits witnessing graph class membership, no $2^{o\left(\sqrt{n/k}\right)}$ time algorithm can exist for finding a proper $(k-1)$-coloring of the graph unless the ETH is false.*

Proof Sketch. Recall that the Theorem 1 proof argument realized the Fig. 4 "color change" gadget as a UNIT-PURE-k-DIR graph. Here, we can instead realize the Fig. 4 "color change" gadget as a UNIT-PURE-$(k+1)$-DIR graph in which we allow for one additional segment embedded in the plane at an angle of $\frac{\pi}{20}$ radians. While we omit the details due to space constraints, briefly, this can be shown to allow for the vertices labeled X_1 and Y_1 in Fig. 4 to correspond to unit segments having the same $\frac{\pi}{4}$, $-\frac{\pi}{4}$, or 0 radian angle. Accordingly, there will no longer be a correspondence between segment angles and color assignments in a proper $(k-1)$-coloring. This then allows one to show that the proper $(k-1)$-coloring problem remains hard even when provided a list of segment coordinates. Finally, as the reduction given in Theorem 1 can be performed in $\mathcal{O}\left(n^2 \cdot k\right)$ time, this yields the current theorem. ◀

▶ **Corollary 1.** *For each $k \geq 4$, the problem of properly 3-coloring an arbitrary m edge graph can be reduced in $\mathcal{O}\left(m^2\right)$ time to properly $(k-1)$-coloring a UNIT-PURE-k-DIR graph.*

Proof. Let Q be an arbitrary graph with vertex set V_Q and edge set E_Q, where $|E_Q| = m$. Generate a graph G with $2m$ edges by replacing every vertex $v_i \in V_Q$ of degree d with a cycle of length d, doing so in such a manner that exactly one vertex in the cycle is adjacent to each distinct neighbor of $v_i \in V_Q$. Assign a unique "cycle label" to the vertices in each generated cycle. We can now generally proceed along the lines of the Theorem 2 proof argument to reduce the problem of properly 3-coloring G to properly $(k-1)$-coloring a

UNIT-PURE-k-DIR graph, with the exception that we do not place the "color change" gadget on polylines corresponding to edges connecting vertices with the same "cycle label" in G. It now suffices to observe that all vertices with the same "cycle label" will be forced to have an identical coloration. ◀

▶ **Theorem 3.** *Unless the ETH is false, for any $k \geq 3$, no $2^{o(n/k)}$ time algorithm can exist for proper k-coloring order n geometric intersection graphs of lines where line-line intersections are constrained to occur on any one of three parallel planes in $\mathbb{R}^3$.*

Proof Sketch. For every $k \geq 3$, we proceed via reduction from the problem of proper edge k-coloring a proper 3-colorable simple undirected k-regular graph G. Briefly, by a reduction of Holyer [28] in the case where $k = 3$ (where Brooks' theorem [10] implies proper 3-colorability), a reduction of Leven & Galil [40] in cases where $k \geq 4$, and by invoking the sparsification lemma of Impagliazzo et. al. [30], it is straightforward to rule out the existence of a $2^{o(n/k)}$ algorithm for finding edge proper k-colorings of these graphs under the ETH.

To begin, generate an embedding $\mathcal{Q}$ of G on three parallel planes in $\mathbb{R}^3$ by embedding each vertex v at a coordinate $\{x, y, z\}$ in $\mathcal{Q}$, where $z = 1$, 2, or 3, if the proper 3-coloring of G places v in the first, second, or third (arbitrarily ordered) color classes, respectively. Next, generate a new embedding $\mathcal{Q}'$ from $\mathcal{Q}$ by perturbing only the x and y coordinates of vertices to place them in general position, guaranteeing that no four points are concyclic in $\mathbb{R}^3$, and thus, that no two edges in G will fall along the same hyperplane in the embedding $\mathcal{Q}'$ unless they intersect at a common vertex point. Here, treating edges in $\mathcal{Q}'$ as line segments with endpoints at vertices, we can replace each line segment with a line containing the segment while ensuring that line-line intersections must occur at vertex positions on any one of three parallel planes in $\mathbb{R}^3$. In this context, letting $\mathcal{L}$ be the set of all lines generated in this manner from the edges in the embedding $\mathcal{Q}'$, it will accordingly be the case that the geometric intersection graph of these lines will correspond to a line graph for G.

Putting everything together, as finding a proper k-coloring of a line graph for G is equivalent to finding an edge proper k-coloring of G, and as we have shown that no $2^{o(n/k)}$ time algorithm can exist for edge proper k-coloring G, this yields the current theorem. ◀

5 Concluding remarks

Should it happen to be the case that no $2^{o(n/k)}$ algorithm exists under the ETH for recognizing or proper k-coloring a UNIT-PURE-k-DIR graph for some $k \in \mathbb{N}_{>2}$ – which would answer the open question of Miltzow discussed in Section 2 [56] – it seems unlikely that it will be possible to prove this via a straightforward modification of our approach in Theorem 1. In particular, recall that in our proof argument for Theorem 1, we make use of an orthogonal integer lattice embedding algorithm for a graph of maximum degree ≤ 4, then proceed by substituting cells in at most a constant factor expansion of this embedding with different UNIT-PURE-k-DIR subgraphs. Here, while there again exist linear time algorithms for computing these embeddings [5, 54], for an order n graph there can be $\mathcal{O}\left(n^2\right)$ cells hosting polylines (or polyline intersections). Furthermore, we remark that finding an embedding of a degree ≤ 4 order n graph minimizing the total length of all polyline segments is NP-hard [55] and, unless $P = NP$, inapproximable within a factor of $\mathcal{O}\left(n^{1/2-\epsilon}\right)$ [3].

A possible path forward would be to: (1) find a problem on a class of graphs that does not admit a $2^{o(n/k)}$ algorithm under the ETH, and (2) show that graphs in this class admit linear time computable orthogonal integer lattice embeddings where the total length of all polylines is asymptotically $\mathcal{O}\left(n\right)$. However, we know of no such problem.

References

1 K. I. Aardal, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for frequency assignment problems. *Ann. Oper. Res.*, 153(1):79–129, 2007.

2 J. Alber and J. Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004.

3 M. J. Bannister, D. Eppstein, and J. A. Simons. Inapproximability of orthogonal compaction. *J. Graph Algorithms Appl.*, 16(3):651–673, 2012.

4 R. D. Barish and T. Shibuya. Proper colorability of segment intersection graphs. *Proc. 28th COCOON*, pages 573–584, 2022.

5 T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998.

6 C. Biró, É Bonnet, D. Marx, T. Miltzow, and P. Rzążewski. Fine-grained complexity of coloring unit disks and balls. *J. Comput. Geom.*, 9(2):47–80, 2018.

7 J. A. Bondy and U. S. R. Murty. *Graph theory with applications.* Macmillan Press: New York, NY, 1st edition, 1976.

8 É Bonnet and P. Rzążewski. Optimality program in segment and string graphs. *Algorithmica*, 81:3047–3073, 2019.

9 H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1–2):3–24, 1998.

10 R. L. Brooks. On colouring the nodes of a network. *Math. Proc. Camb. Philos. Soc.*, 37(2):194–197, 1941.

11 S. Cabello and M. Jejčič. Refining the hierarchies of classes of geometric intersection graphs. *Electron. J. Comb.*, 24(1)(P1.33):1–19, 2017.

12 J. Chalopin and D. Gonçalves. Every planar graph is the intersection graph of segments in the plane: extended abstract. *Proc. 41st STOC*, pages 631–638, 2009.

13 S. Chaplick, P. Hell, Y. Otachi, T. Saitoh, and R. Uehara. Intersection dimension of bipartite graphs. *Proc. TAMC*, pages 323–340, 2014.

14 M. Cygan, F. V. Fomin, A. Golovnev, A. S. Kulikov, I. Mihajlin, J. Pachocki, and A. Socała. Tight bounds for graph homomorphism and subgraph isomorphism. *Proc. SODA*, pages 1643–1649, 2016.

15 M. de Berg, H. L. Bodlaender, S. Kisfaludi-Bak, D. Marx, and T. C. van der Zanden. A framework for exponential-time-hypothesis–tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49(6):1291–1331, 2020.

16 D. de Werra and Y. Gay. Chromatic scheduling and frequency assignment. *Discrete Appl. Math.*, 49(1–3):165–174, 1994.

17 J. Deguchi, T. Sugimura, Y. Nakatani, T. Fukushima, and M. Koyanagi. Quantitative derivation and evaluation of wire length distribution in three-dimensional integrated circuits using simulated quenching. *Jpn. J. Appl. Phys.*, 45(4B):3260–3265, 2006.

18 R. Diestel. *Graph theory.* Springer-Verlag: Heidelberg, 5th edition, 2017.

19 G. Ehrlich, S. Even, and R. E. Tarjan. Intersection graphs of curves in the plane. *J. Comb. Theory. Ser. B*, 21(1):8–20, 1976.

20 P. Carmi et. al. Balanced line separators of unit disk graphs. *Comput. Geom.*, 86:101575, 2020.

21 F. V. Fomin, D. Lokshtanov, F. Panolan, S. Saurabh, and M. Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete Comput. Geom.*, 62(4):879–911, 2019.

22 J. Fox and J. Pach. Separator theorems and Turán-type results for planar intersection graphs. *Adv. Math.*, 219(3):1070–1080, 2008.

23 J. Fox and J. Pach. A separator theorem for string graphs and its applications. *Comb. Probab. Comput.*, 19(3):371–390, 2010.

24 J. Fox and J. Pach. Computing the independence number of intersection graphs. *Proc. 22nd SODA*, pages 1161–1165, 2011.

25 J. Fox, J. Pach, and C. D. Tóth. A bipartite strengthening of the crossing lemma. *J. Comb. Theory. Ser. B*, 100(1):23–35, 2010.

26 W. K. Hale. Frequency assignment: theory and applications. *Proc. IEEE*, 68(12):1497–1514, 1980.

27 P. Hliněný and J. Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discrete Math.*, 229(1–3):101–124, 2001.

28 I. Holyer. The NP-completeness of edge coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.

29 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

30 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

31 S. Kisfaludi-Bak and T. C. van der Zanden. On the exact complexity of Hamiltonian cycle and q-colouring in disk graphs. *Proc. 10th CIAC*, pages 369–380, 2017.

32 P. Koebe. Kontaktprobleme der konformen abbildung. *Berichte Verhande. Sächs. Akad. Wiss. Leipzig, Math. -Phys. Klasse*, 88:141–164, 1936.

33 J. Kratochvíl. String graphs. II. Recognizing string graphs is NP-hard. *J. Comb. Theory. Ser. B*, 52(1):67–78, 1991.

34 J. Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Appl. Math.*, 52(3):233–252, 1994.

35 J. Kratochvíl, M. Goljan, and P. Kučera. String graphs. *Rozpr. Česk. Akad. Věd, Řada Mat. Přír. Věd*, 96(3):1–96, 1986.

36 J. Kratochvíl and A. Kuběna. On intersection representations of co-planar graphs. *Discrete Math.*, 178(1–3):251–255, 1998.

37 J. Kratochvíl and J. Matoušek. Intersection graphs of segments. *J. Comb. Theory. Ser. B*, 62(2):289–315, 1994.

38 J. Kratochvíl and J. Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Comment. Math. Univ. Carolinae*, 31(1):85–93, 1990.

39 J. R. Lee. Separators in region intersection graphs. *arXiv:1608.01612*, pages 1–29, 2016.

40 D. Leven and Z. Galil. NP completeness of finding the chromatic index of regular graphs. *J. Algorithms*, 4(1):35–44, 1983.

41 R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

42 D. Lokshtanov, F. Panolan, S. Saurabh, J. Xue, and M. Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). *Proc. 33rd SODA*, pages 2005–2031, 2022.

43 D. Marx. The square root phenomenon in planar graphs. *Proc. 40th ICALP*, pages 1–28, 2013.

44 D. Marx and M. Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. *ACM Trans. Algorithms*, 18(2):1–64, 2022.

45 J. Matoušek. Near-optimal separators in string graphs. *Comb. Probab. Comput.*, 23(1):135–139, 2014.

46 G. L. Miller, S. H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997.

47 I. Mustaţă and M. Pergel. Unit grid intersection graphs: recognition and properties. *arXiv:1306.1855*, pages 1–19, 2013.

48 A. Nahman, A. Fan, J. Chung, and R. Reif. Wire-length distribution of three-dimensional integrated circuits. *Proc. IITC*, pages 233–235, 1999.

49 K. Okrasa and P. Rzążewski. Subexponential algorithms for variants of homomorphism problem in string graphs. *Proc. 45th WG*, pages 1–13, 2019.

50 Y. Otachi, Y. Okamoto, and K. Yamazaki. Relationships between the class of unit grid intersection graphs and other classes of bipartite graphs. *Discrete Appl. Math.*, 155(17):2383–2390, 2007.

51 J. Pach and G. Tóth. Recognizing string graphs is decidable. *Discrete Comput. Geom.*, 28(4):593–606, 2002.

52 J. Pach and G. Tóth. How many ways can one draw a graph? *Combinatorica*, 26(5):559–576, 2006.

53 F. Panolan, S. Saurabh, and M. Zehavi. Contraction decomposition in unit disk graphs and algorithmic applications in parameterized complexity. *Proc. 30th SODA*, pages 1035–1054, 2019.

54 A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom.*, 9(1–2):83–110, 1998.

55 M. Patrignani. On the complexity of orthogonal compaction. *Comput. Geom.*, 19(1):47–67, 2001.

56 I. Penev, S. Oum, T. Miltzow, and L. Feuilloley. GROW 2017: open problems. URL: `http://www.fields.utoronto.ca/sites/default/files/GROW_Open_Problems%202017.pdf`.

57 F. S. Roberts. *On the boxicity and cubicity of a graph.* In: Recent progress in combinatorics (W. T. Tutte, ed.). Academic Press: New York, NY, 1969.

58 M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003.

59 M. Schaefer and D. Štefankovič. Decidability of string graphs. *Proc. 33rd STOC*, pages 241–246, 2001.

60 M. Schaefer and D. Štefankovič. Decidability of string graphs. *J. Comput. Syst. Sci.*, 68(2):319–334, 2004.

61 E. R. Scheinerman. Intersection classes and multiple intersection parameters of graphs. *Ph.D. thesis, Princeton University*, 1984.

62 F. W. Sinden. Topology of thin film RC circuits. *Bell Syst. Tech. J.*, 45(9):1639–1662, 1966.

63 W. D. Smith and N. C. Wormald. Geometric separator theorems and applications. *Proc. 39th FOCS*, pages 232–243, 1998.

64 J. E. Steif. The frame dimension and the complete overlap dimension of a graph. *J. Graph Theory*, 9(2):285–299, 1985.

Destroying Densest Subgraphs Is Hard

Cristina Bazgan
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France

André Nichterlein
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Sofia Vazquez Alferez
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris, France

Abstract

We analyze the computational complexity of the following computational problems called Bounded-Density Edge Deletion and Bounded-Density Vertex Deletion: Given a graph G, a budget k and a target density τ_ρ, are there k edges (k vertices) whose removal from G results in a graph where the densest subgraph has density at most τ_ρ? Here, the density of a graph is the number of its edges divided by the number of its vertices. We prove that both problems are polynomial-time solvable on trees and cliques but are NP-complete on planar bipartite graphs and split graphs. From a parameterized point of view, we show that both problems are fixed-parameter tractable with respect to the vertex cover number but W[1]-hard with respect to the solution size. Furthermore, we prove that Bounded-Density Edge Deletion is W[1]-hard with respect to the feedback edge number, demonstrating that the problem remains hard on very sparse graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph modification problems, NP-hardness, fixed-parameter tractability, W-hardness, special graph classes

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.6

Related Version *Full Version*: http://arxiv.org/abs/2404.08599

Acknowledgements We thank anonymous reviewers of SWAT 2024 for their detailed comments improving the presentation.

1 Introduction

Finding a densest subgraph is a central problem with applications ranging from social network analysis to bioinformatics to finance [21]. There is a rich literature on this topic with the first polynomial-time algorithms given more than 40 years ago [15, 29]. In this work, we study the robustness of densest subgraphs under perturbations of the input graph. More precisely, we study the (parameterized) complexity of the following two computational problems called Bounded-Density Edge Deletion and Bounded-Density Vertex Deletion: Given a graph G, a budget k and a target density τ_ρ, the questions are whether there are k edges, respectively, k vertices, whose removal from G results in a graph where the densest subgraph has density at most τ_ρ? Here, the density $\rho(G)$ of a graph G is defined as the ratio between its number m of edges and number n of vertices, that is, $\rho(G) = m/n$, which is equal to half the average degree of G. Thus, we contribute to the literature on graph modification problems with degree constraints [13, 23, 27]. More broadly, our work fits into parameterized algorithmics on graph modification problems – a line of research with a plethora of results. See Crespelle et al. [6] for a recent survey focusing on edge modification problems.

Denote with $\rho^*(G)$ the density of a densest subgraph of G. Note that cycles have density exactly one and forests a density below one. Thus, it is easy to see that a graph G is a forest if and only if $\rho^*(G) < 1$. Hence, our problems contain the NP-hard Feedback

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 6; pp. 6:1–6:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Our results for Bounded-Density Edge Deletion / Vertex Deletion.

	Edge Deletion	Vertex Deletion
Trees	$O(n^3)$ (Theorem 7)	$O(n)$ (Theorem 16)
Cliques	$O(n^2)$ (Theorem 9)	$O(n^2)$ (trivial)
Split	NP-complete (Theorem 13)	NP-complete (Theorem 20)
Planar Bipartite	NP-complete (Theorem 12)	NP-complete (Theorem 17)
Solution Size k	W[1]-hard (Theorem 15)	W[2]-hard (Theorem 23)
Vertex Cover Number	FPT (Theorem 14)	FPT (Theorem 21)
Feedback Edge Number	W[1]-hard (Theorem 15)	?

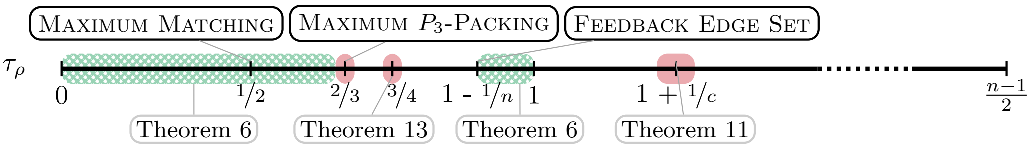


Figure 1 The computational complexity and special cases of Bounded-Density Edge Deletion for specific values of the target density τ_ρ, see Sections 3.1 and 3.2 for the details. Green (hatched) boxes indicate polynomial-time solvable cases while red (solid) boxes denote NP-hard cases. The c in $1 + 1/c$ can be any constant larger than 24. The complexity for larger values of τ_ρ remains open.

Vertex Set and the polynomial-time solvable Feedback Edge Set[1], respectively, as special cases. For target densities smaller than one not only are cycles to be destroyed, but also a bound on the size of the remaining connected components is implied. For example, for $\tau_\rho = 2/3$ ($= 1/2$ or $= 0$) each connected component in the resulting graph can have at most 2 edges (1 edge for $\tau_\rho = 1/2$ or 0 edges for $\tau_\rho = 0$). Consequently, Bounded-Density Vertex Deletion generalizes Dissociation Set ($\tau_\rho = 1/2$) and Vertex Cover ($\tau_\rho = 0$). Bounded-Density Edge Deletion generalizes Maximum Cardinality Matching ($\tau_\rho = 1/2$) and the NP-hard Maximum P_3-packing ($\tau_\rho = 2/3$) where the non-deleted edges form the matching and P_3-packing, respectively.

Our contributions. We refer to Table 1 for an overview of our results. Given the above connections to known computational problems, we start with the seemingly easier of the two problems: Bounded-Density Edge Deletion. We provide polynomial-time algorithms for specific target densities below one (see Figure 1) or when the input is a clique or tree (see Section 3.1). However, beyond these cases, the problem turns out to be surprisingly hard. There are target densities above or below one for which it is NP-hard, see Figure 1 for an overview. We show that Bounded-Density Edge Deletion remains NP-hard on claw-free cubic planar, planar bipartite, and split graphs (see Section 3.2). Moreover, we prove W[1]-hardness with respect to the combined parameter k and feedback edge number. This implies that the problem remains hard even on very sparse graphs as the feedback edge number in a connected graph is $m - n + 1$, despite being polynomial-time solvable on trees. Note that this also implies W[1]-hardness with respect to prominent parameters like treewidth. Moreover, our employed reduction shows W[1]-hardness for T_{h+1}-Free Edge

[1] Given a graph G and an integer k, Feedback Vertex Set (Feedback Edge Set) asks if there is a set of k vertices (k edges) whose removal makes G acyclic.

Deletion[2] with respect to the treewidth, thus answering an open question by Enright and Meeks [10]. On the positive side, using integer linear programming, we classify the problem as fixed-parameter tractable with respect to the vertex cover number (see Section 3.3).

Turning to Bounded-Density Vertex Deletion, we derive NP-hardness for all $\tau_\rho \in [0, n^{1-1/c}]$ for any constant c. Note that the density of a graph is between 0 and $(n-1)/2$ and the case $\tau_\rho = (n-1)/2$ is trivial. Moreover, we show NP-hardness on planar bipartite graphs of maximum degree four, line graphs of planar bipartite graphs, and split graphs (see Section 4.2) as well as a polynomial-time algorithm for trees (see Section 4.1). Furthermore, we prove W[2]-hardness with respect to k and fixed-parameter tractability with respect to the vertex cover number (see Section 4.3). Notably, the latter algorithm is easier than in the edge deletion setting; in particular it does not rely on integer linear programming.

Due to space restrictions the proofs of some statements (marked by $\star$) are omitted.

Further related work. The density as defined above is related to a variety of useful concepts. It belongs to a family of functions of the form $f(G, a, b, c) = a|E(G)|/(b|V(G)| - c)$ with $a, b, c \in \mathbb{Q}$. Depending on the values of a, b and c, the function has been used to study a variety of network properties [17]. For instance $\rho(G) = f(G, 1, 1, 0)$ is used in the study of random graphs [1], whilst $f(G, 1, 1, 1)$ comes up in the study of vulnerability of networks [7], and $f(G, 1, 3, 6)$ is used to study rigid frameworks [20]. A more general class of functions can be studied, where a, b, c are not rational numbers but functions. For instance, Hobbs [16] studies the vulnerability of a graph G by finding the subgraph H of G that maximizes $|E(H)|/(|V(H)| - \omega(H))$ with $\omega(H)$ being the number of connected components in H. The interpretation being that attacking such a subgraph would lead to the maximum number of connected components being created per unit of effort spent on the attack, where the effort is proportional to the number of edges being targeted by the attacker.

The maximum average degree $\mathrm{mad}(G)$ of a graph G is the maximum of the average degrees of all subgraphs of G. Note that $\mathrm{mad}(G) = 2\rho^*(G)$. Recently, Nadara and Smulewicz [25] showed that for every graph G and positive integer k such that $\mathrm{mad}(G) > k$, there exists a polynomial-time algorithm to compute a subset of vertices $S \subseteq V(G)$ such that $\mathrm{mad}(G - S) \leq \mathrm{mad}(G) - k$ and every subgraph of $G[S]$ has minimum degree at most $k - 1$. Though no guarantees are given that S has minimum size for subsets S that achieve $\mathrm{mad}(G - S) \leq \mathrm{mad}(G) - k$.

Modifying a graph to bound its maximum average degree can be of use because of a variety of results on the colorability of graphs with bounded mad. In general, mad can be used to give a bound on the chromatic number $\chi(G)$ of G, as $\chi(G) \leq \lfloor \mathrm{mad}(G) \rfloor + 1$ [26]. It is well-known that for any planar graph G, the maximum average degree is related to the girth $g(G)$ of G in the following way: $(\mathrm{mad}(G) - 2)(g(G) - 2) < 4$ [25]. Several results are known for variations of coloring problems and mad [2, 3, 4, 19].

2 Preliminaries

Notation. For $n \in \mathbb{N}$ we set $[n] = \{1, 2, \ldots, n\}$. Let G be a simple, undirected, and unweighted graph. We denote the set of vertices of G by $V(G)$ and the set of edges of G by $E(G)$. We set $n_G = |V(G)|$ and $m_G = |E(G)|$. We denote the degree of a vertex $v \in V(G)$ by $\deg_G(v)$. If the graph is clear from context, then we drop the subscript. The minimum

[2] Given a graph and an integer k, the question is whether k edges can be removed so that no connected component has more than h vertices?

degree of G is denoted by $\delta(G)$, and the maximum degree of G is denoted by $\Delta(G)$. We denote with $H \subseteq G$ that H is a subgraph of G. The *density of* G is $\rho(G) = m/n$. We define the density of the empty graph as zero. We denote by $\rho^*(G)$ the density of the densest subgraph of G, that is, $\rho^*(G) = \max_{H \subseteq G} \rho(H)$. For a subset of vertices $W \subseteq V(G)$, we denote with $G[W]$ the subgraph *induced* by W. For two subsets of vertices $W, U \subseteq V(G)$ we set $E(W, U)$ to be the set of edges with one endpoint in W and another in U.

We denote by P_n the path on n vertices, by K_n the complete graph on n vertices (also called a clique of size n), and by $K_{a,b}$ the complete bipartite graph with a and b the size of its two vertex sets. A graph G is *r-regular* if $\deg(v) = r$ for every vertex $v \in G$. A *perfect P_3-packing of* G is a partition of $V(G)$ into sets $V_1, V_2, \ldots, V_{n/3}$ such that for all $i \in [n/3]$ the graph $G[V_i]$ is isomorphic to P_3.

A graph G is *balanced* if $\rho(G') \leq \rho(G)$ for every subgraph $G' \subseteq G$. Let $\rho'(G) = \frac{m}{n-1}$ for $1 \leq n-1$ and define ρ' to be 0 for the empty graph and one-vertex graph. A graph G is *strongly balanced* if $\rho'(G') \leq \rho'(G)$ for every subgraph $G' \subseteq G$. Ruciński and Vince [31, page 252] point out that every strongly balanced graph is also balanced, though the converse is not true.

Problem Definitions. The problem definition for the edge deletion variant is as follows (the definition for vertex deletion is analogous):

Bounded-Density Edge Deletion
Input: A graph G, an integer $k \geq 0$ and a rational number $\tau_\rho \geq 0$.
Question: Is there a subset $F \subseteq E(G)$ with $|F| \leq k$ such that $\rho^*(G - F) \leq \tau_\rho$?

There are two natural optimization problems associated to Bounded-Density Edge Deletion which we call Min Density Edge Deletion (given k minimize τ_ρ) and Min Edge Deletion Bounded-Density (given τ_ρ minimize the number of edge deletions k).

We emphasize that all problems for vertex deletion are defined and named analogously.

Useful Observations. We often compare the ratio of vertices to edges in different induced subgraphs. To this end, the following basic result is useful.

▶ **Lemma 1** ($\star$). $\frac{a}{b} \leq \frac{a+c}{b+d} \iff \frac{a}{b} \leq \frac{c}{d}$ *and* $\frac{a}{b} = \frac{a+c}{b+d} \iff \frac{a}{b} = \frac{c}{d}$.

The following is a collection of easy observations that can be obtained with Lemma 1.

▶ **Lemma 2** ($\star$). *Let G^* be a densest subgraph of G with $\rho(G^*) = \rho^*(G)$. Then:*
1. *If G^* is not connected, then each connected component C of G^* has density $\rho^*(G)$.*
2. *If $\rho(G^*) = a/b$ for $a, b \in \mathbb{N}$ and $a < b$, then $a = b - 1$ and G^* is a tree on b vertices or a forest where each tree is on b vertices.*
3. *Any vertex $v \notin V(G^*)$ has at most $\lfloor \rho(G^*) \rfloor$ neighbors in $V(G^*)$.*
4. *The minimum degree in G^* is at least $\lceil \rho(G^*) \rceil$. This is tight for trees.*

Note that Lemma 2 (4.) implies that we can remove vertices with degree less than our desired target density τ_ρ.

▶ **Data Reduction Rule 3.** *Let v be a vertex with degree* $\deg(v) < \tau_\rho$. *Then delete v.*

The following two observations imply that there are only a polynomial number of "interesting" values for the target density τ_ρ. Thus, if we have an algorithm for the decision problem, then, using binary search, one can solve the optimization problems with little overhead in running time.

▶ **Observation 4.** *The density of a graph G on n vertices can have values between 0 and $(n-1)/2$ in intervals of $1/n$: $\rho(G) \in \{0, 1/n, 2/n, \dots, \binom{n}{2}/n = (n-1)/2\}$.*

▶ **Observation 5** ([15])**.** *The maximum density of a subgraph of G can take only a finite number of values: $\rho^*(G) \in \{m'/n' \mid 0 \leq m' \leq m, 1 \leq n' \leq n\}$. Moreover, the minimum distance between two different possible values of $\rho^*(G)$ is at least $1/(n(n-1))$.*

3 Bounded-Density Edge Deletion

In this section we provide our results for Bounded-Density Edge Deletion, starting with the polynomial-time algorithms, continuing with the NP-hard cases, and finishing with our parameterized results.

3.1 Polynomial-time solvable cases

Specific Density Intervals. We show that if the density τ_ρ falls within one of two intervals, then Min Edge Deletion Bounded-Density boils down to computing maximum matchings or spanning trees.

▶ **Theorem 6.** *Min Edge Deletion Bounded-Density can be solved in time $O(m\sqrt{n})$ if $0 \leq \tau_\rho < 2/3$ and in time $O(n+m)$ if $1 - 1/n \leq \tau_\rho \leq 1$.*

Proof. The proof is by case distinction on τ_ρ.

Note that if $\tau_\rho < 1/2$, then no edge can remain in the graph as a K_2 has density $1/2$. Similarly, if $1/2 \leq \tau_\rho < 2/3$, then no connected component can have more than one edge: Otherwise, the component would contain a P_3 which has density $2/3$. Hence, computing a maximum cardinality matching in time $O(m\sqrt{n})$ [24] and removing all edges not in the matching solves the given instance of Min Edge Deletion Bounded-Density.

The second interval is similar. If $1 - 1/n \leq \tau_\rho < 1$, then the resulting graph cannot have any cycle as a cycle has density 1. Moreover, any tree on at most n vertices has density at most $1 - 1/n$. Thus, in this case Min Edge Deletion Bounded-Density is equivalent to computing a minimum feedback edge set, which can be done in time $O(n+m)$ by e. g. deleting all edges not in a spanning tree.

Lastly, if $\tau_\rho = 1$, then each connected component can have at most one cycle, that is, the resulting graph must be a pseudoforest: Consider a connected component C with ℓ vertices and at least two cycles. Any spanning tree of C contains $\ell - 1$ edges and misses at least one edge per cycle. Hence, C contains at least $\ell + 1$ edges and has, thus, density larger than one. Thus, each connected component in the remaining graph can have at most as many edges as vertices. Hence, a solution to the Min Edge Deletion Bounded-Density instance is to do the following for each connected component: delete all edges not in a spanning tree and reinsert an arbitrary edge. This can be done in $O(n+m)$ time. ◀

Trees. If the input is a tree, then any target threshold $\tau_\rho \geq 1$ makes the problem trivial. Hence, the case $\tau_\rho < 1$ is left. Thus, each tree in the remaining graph can have at most $h = \lfloor 1/(1-\tau_\rho) \rfloor$ many vertices: a tree with $h' > h$ vertices has density $(h'-1)/h' = 1 - 1/h' > 1 - 1/h \geq 1 - (1-\tau_\rho) = \tau_\rho$. Hence, the task is to remove as few edges as possible so that each connected component in the remaining graph is of order at most h. This problem is known as T_{h+1}-Free Edge Deletion and can be solved in $O((wh)^{2w}n)$ time [10], where w is the treewidth. As trees have treewidth one and $h \leq n$ (otherwise the problem is trivial), we get the following.

▶ **Theorem 7.** *On the trees,* Min Edge Deletion Bounded-Density *can be solved in time $O(n^3)$.*

Cliques. The problem is not completely trivial on cliques: While a target threshold τ_ρ indicates an upper bound on the remaining edges (as $\rho(G-F) \leq \tau_\rho$ must hold), the question is whether for all m and n there is a balanced graph G with m edges and n vertices; recall that a graph is balanced if the whole graph is a densest subgraph. Ruciński and Vince [31] showed a slightly stronger statement about strongly balanced graphs. Recall that every strongly balanced graph is also balanced; refer to Section 2 for formal definitions.

▶ **Theorem 8** ([31, Theorem 1])**.** *Let n and m be two integers. If $1 \leq n-1 \leq m \leq \binom{n}{2}$, then there exists a strongly balanced graph with n vertices and m edges.*

The proof of Ruciński and Vince [31] is constructive: A strongly balanced graph (that is also a balanced graph) with m edges and n vertices can be constructed in $O(m)$ time.

▶ **Theorem 9.** *On the complete graph K_n,* Min Edge Deletion Bounded-Density *and* Min Density Edge Deletion *can be solved in time $O(n^2)$.*

Proof. We provide the proof for Min Edge Deletion Bounded-Density. The proof for Min Density Edge Deletion is analogous.

Consider an instance $(G = K_n, \tau_\rho)$ of Min Edge Deletion Bounded-Density. Let $F \subseteq E(G)$ be a solution that our algorithm wants to find. Throughout the proof we assume $\tau_\rho \leq (n-1)/2$ and $n \geq 1$, as otherwise $F = \emptyset$. We consider two cases: $\tau_\rho < 1$ and $\tau_\rho \geq 1$.

Case 1. ($\tau_\rho < 1$): Let $t \leq n$ be the largest integer satisfying $(t-1)/t \leq \tau_\rho$. By Lemma 2 (2.), the resulting graph $K_n - F$ must be a collection of trees on at most t vertices each. Thus, partition the vertices in $\lceil n/t \rceil$ parts of size at most t. For each part keep an arbitrary spanning tree. Then F consists of all non-kept edges, i. e., all edges between the parts and all edges not in the selected spanning trees. Clearly, this can be done in $O(n^2)$ time.

Case 2. ($\tau_\rho \geq 1$): We will construct a strongly balanced graph G' on n vertices with density as close to τ_ρ as possible, as allowed by Observation 4. To this end, let t be the largest number in $\{0, 1/n, \dots, (n-1)/2\}$ so that $t \leq \tau_\rho$, thus $t = \ell/n$ for some integer $\ell \in \{n, n+1, \dots, \binom{n}{2}\}$ (as $\tau_\rho \geq 1$).

Now we use Theorem 8 to construct a balanced graph G' with ℓ edges and n vertices, thus $\rho^*(G') \leq \tau_\rho$. We will select $F \subseteq E(G)$ so that $G' = G - F$, that is, F contains all edges not in G' and $|F| = \binom{n}{2} - \ell$. By choice of ℓ we know that $(\ell+1)/n > \tau_\rho$. Hence, removing less than $\binom{n}{2} - \ell$ edges from G means the whole graph has density more than τ_ρ. As building the graph G' takes time $O(\ell)$ the overall running time is $O(n^2)$.

To construct the analogous proof for Min Density Edge Deletion use two cases: $k > \binom{n}{2} - n$ (equivalent of Case 1 above) and $k \leq \binom{n}{2} - n$ (equivalent of Case 2). ◀

3.2 NP-Hardness for special graph classes

Claw-free cubic planar graphs. For our first hardness proof of Bounded-Density Edge Deletion we provide a reduction from Perfect P_3-packing which was proven NP-complete even in claw-free cubic planar graphs [33]. Denote with P_k a path on k vertices. A *perfect P_3-packing* of a given graph G is a partition of G into subgraphs in which each subgraph is isomorphic to P_3. The Perfect P_3-packing problem is defined as follows:

Perfect P_3-packing
Input: A graph G.
Question: Is there a perfect P_3-packing of G?

▶ **Theorem 10.** *Bounded-Density Edge Deletion is* NP*-complete for $\tau_\rho = 2/3$ even on claw-free cubic planar graphs.*

Proof. Given an instance G of perfect P_3-packing where G is a claw-free cubic planar graph, let $\mathcal{I} = (G, k, \tau_\rho)$ be an instance of Bounded-Density Edge Deletion where $k = m - 2n/3$ and $\tau_\rho = 2/3$. We claim that G has a perfect P_3-packing if and only if there is a set $F \subseteq E(G)$ with $\rho^*(G - F) = 2/3$ and $|F| = m - 2n/3$.

"$\Rightarrow$:" Given a perfect P_3-packing of G, we set F to be the set of all edges in G that are not in the P_3-packing. Clearly $\rho^*(G - F) = 2/3$. Additionally, $|F| = m - 2n/3$ since there are $n/3$ paths in a perfect P_3-packing, and each path has two edges.

"$\Leftarrow$:" Consider $F \subseteq E(G)$ of size at most $m - 2n/3$ such that $\rho^*(G - F) \leq 2/3$. Then $|E(G) \setminus F| \geq 2n/3$. Note that $\tau_\rho = 2/3$ implies, by Lemma 2 (2.), that all connected components of $G - F$ must be trees of size at most 3. In other words, all connected components in $G - F$ are singletons, P_2's, or P_3's. Denote by t the number of connected components that are P_3's in $G - F$. Then there are $3t$ vertices and $2t$ edges of $G - F$ which belong to a P_3. Consequently, there are $n - 3t$ vertices and $|E(G) \setminus F| - 2t \geq 2(n - 3t)/3$ edges of $G - F$ which are either singletons or belong to a P_2. This is possible only if $n - 3t = 0$, that is $G - F$ is a perfect P_3-packing of G. ◀

Planar graphs with $\Delta = 3$, target density above 1. We next show that Bounded-Density Edge Deletion remains NP-complete even for $\tau_\rho > 1$. To prove this, we reduce from Vertex Cover on cubic planar graphs, which is known to be NP-complete [14]. Vertex Cover is defined as follows:

Vertex Cover
Input: A graph G and a positive integer k.
Question: Is there a vertex cover $C \subseteq V(G)$ of size at most k, that is, for each $\{u, v\} \in E$ at least one of u or v is in C?

▶ **Theorem 11** (⋆)**.** *Bounded-Density Edge Deletion is* NP*-complete for $\tau_\rho > 1$ even on planar graphs with maximum degree 3.*

Bipartite graphs and split graphs. Finally, we show that Bounded-Density Edge Deletion is NP-hard even on planar bipartite graphs and on split graphs.

▶ **Theorem 12** (⋆)**.** *Bounded-Density Edge Deletion remains* NP*-complete on planar bipartite graphs.*

▶ **Theorem 13** (⋆)**.** *Bounded-Density Edge Deletion is* NP*-complete on split graphs with $\tau_\rho = 3/4$.*

3.3 Parameterized Complexity Results

FPT wrt. Vertex Cover Number. The vertex cover number of a graph denotes the size of a smallest vertex cover, i. e., the size of a set of vertices whose removal results in an edge-less graph. Although this parameter is relatively large, we still need to rely on integer linear programming in our next algorithm.

▶ **Theorem 14.** *Bounded-Density Edge Deletion can be solved in $2^{O(\ell 2^{2\ell})} + O(m + n)$ time where ℓ is the vertex cover number.*

Proof. Consider an instance (G, k, τ_ρ) of Bounded-Density Edge Deletion.

We provide an algorithm that first computes a minimum vertex cover $C \subseteq V(G)$, $|C| = \ell$, in $O(2^\ell + n + m)$ time [8]. Then it computes edge deletions within $G[C]$ and incident to $S = V(G) \setminus C$ with an ILP, i. e., computes F. To this end, we divide the vertices in S into at most 2^ℓ classes $I_1, \ldots, I_{2^\ell}$, where two vertices are in the same class if and only if they have the same neighbors. Hence, we can define for a class I_i the neighborhood $N(I_i) = N(v)$ for $v \in I_i$. We denote with $|I_i|$ the number of vertices in the class I_i. The usefulness of these classes hinges on the fact that at least one densest subgraph G' of G is such that it either contains all the vertices of a class I_i or none. This is a consequence of Lemma 1: if removing (adding) a vertex of I_i from a subgraph of G increases its density, then removing (adding) any other vertices from I_i must do the same, as they are twins and non-adjacent.

We say a class I_j is *obtainable* from a class I_i if $N(I_j) \subseteq N(I_i)$, that is, by deleting some edges a vertex from I_i can get into class I_j. Note that I_j is obtainable from I_j. Denote with $\text{ob}(I_i)$ all classes obtainable from I_i, formally, $\text{ob}(I_i) = \{I_j \mid N(I_j) \subseteq N(I_i)\}$. Similarly, we denote with $\text{ob}^{-1}(I_j)$ all classes I_i so that I_j is obtainable from I_i, formally, $\text{ob}^{-1}(I_j) = \{I_i \mid N(I_j) \subseteq N(I_i)\}$. If I_j is obtainable from I_i, then we set $\text{cost}(i \to j) = |N(I_i) \setminus N(I_j)|$ to be the number of edges that need to be deleted from a vertex $v \in I_i$ to make it a vertex in I_j.

We now give our ILP. To handle edges with one endpoint in the independent set S we do the following: For each pair of classes I_i, I_j, we add a variable $x_{i\to j}$ whose purpose is to denote how many vertices from class I_i will end up in class I_j by deleting edges. For convenience, we further have a variable $y_j = \sum_{I_i \in \text{ob}^{-1}(I_j)} x_{i\to j}$ denoting the total number of vertices that end up in a class I_j after deleting edges. To ensure correctness we require that no vertex from class I_i gets transformed into a vertex from a class not obtainable from I_i which we represent with the constraint $\sum_{I_j \in \text{ob}(I_i)} x_{i\to j} = |I_i|$ for all $i \in [2^\ell]$. Moreover, both $x_{i\to j}$ and y_j must be integers for all $i, j \in [2^\ell]$.

To handle edges within C we do the following: For each edge $e \in E(C)$ we create a binary variable z_e, where $z_e = 1$ if e survives and 0 if it gets deleted. Again for convenience, for a subset $C' \subseteq C$ we denote with $m_{C'} = \sum_{e \in E(C')} z_e$ the number of edges within C' that survive. The ILP is as follows:

$$\text{Minimize} \quad \sum_{j=1}^{2^\ell} \sum_{I_i \in \text{ob}(I_j)} \text{cost}(i \to j) \cdot x_{i\to j} + \sum_{e \in E(C)} (1 - z_e) \tag{1}$$

$$\text{such that} \quad \sum_{I_j \in \text{ob}(I_i)} x_{i\to j} = |I_i| \qquad \forall i \in [2^\ell] \tag{2}$$

$$\sum_{I_i \in \text{ob}^{-1}(I_j)} x_{i\to j} = y_j \qquad \forall j \in [2^\ell] \tag{3}$$

$$\sum_{e \in E(C')} z_e = m_{C'} \qquad \forall C' \subseteq C \tag{4}$$

$$m_{C'} + \sum_{i \in \mathcal{I}} y_i |N(I_i) \cap C'| \le \tau_\rho(|C'| + \sum_{i \in \mathcal{I}} y_i) \qquad \forall C' \subseteq C, \forall \mathcal{I} \subseteq [2^\ell] \tag{5}$$

$$x_{i\to j}, y_i \in \{0, 1, 2, \ldots\} \qquad \forall i, j \in [2^\ell] \tag{6}$$

$$z_e \in \{0, 1\} \qquad \forall e \in E(C) \tag{7}$$

To prove correctness it remains to show that the ILP admits a solution such that the objective value is at most k if and only if there exist an edge deletion set F of size at most k such that $\rho^*(G - F) \le \tau_\rho$.

Firstly, suppose there exists a feasible solution to the ILP that achieves an objective value of at most k. Because the vertices in I_i are indistinguishable, the set of variables $\{x_{i\to j} \mid j \in [2^\ell]\}$ uniquely determines the set of edges incident to I_i that need to go into F for each class $I_i, i \in [2^\ell]$. The other edges in F can be directly read off the solution $E(C) \cap F = \{z_e \mid z_e = 0\}$. A densest subgraph G' of G contains some vertices from C and from S. If G' does not contain all vertices from some class I_i, then either adding or removing all vertices from I_i will give a graph G'' that is as least as dense as G' (cf. Lemma 1). Inequality (5) ensures that all such possible subgraphs G'' have density at most τ_ρ. Therefore, this constraint guarantees that all subgraphs of $G - F$ have density at most τ_ρ. In the objective function, $\sum_{e\in E(C)}(1 - z_e)$ counts the number of edges that are deleted from C, whilst $\sum_{j=1}^{2^\ell} \sum_{I_i\in \mathrm{ob}(I_j)} \mathrm{cost}(i \to j) \cdot x_{i\to j}$ counts the number of edges that are deleted from $E(C, S)$. Since the feasible solution achieves an objective value of at most k, we have that $|F| \leq k$. Thus, F as built above is the desired edge deletion set of size at most k.

Secondly, consider an edge subset $F \subseteq E(G)$ of size at most k such that $\rho^*(G - F) \leq \tau_\rho$. We can assign values to $x_{i\to j}$ by reading the number of edges in F that are incident to each vertex v in class I_i. The first and second constraint will be satisfied by construction. The third constraint will be satisfied because $\rho^*(G - F) \leq \tau_\rho$. And since $|F| \leq k$ the minimum of the objective function will have value at most k.

The ILP has at most $2^{2\ell}$ variables of type $x_{i\to j}$ and at most $\binom{\ell}{2}$ of type z_e. The linear program can be expressed without variables of types y_j and $m_{C'}$ simply by substituting their expressions into the other constraints. This yields a total of $O(2^{2\ell+1})$ variables. In addition, we have $O(2^\ell \cdot 2^{2^\ell})$ constraints. Since an ILP instance $\mathcal{I}$ on p variables can be solved in time $O(p^{2.5p+o(p)} \cdot |\mathcal{I}|)$ [22, 18, 12] we obtain a total (FPT) running time of $2^{O(\ell 2^{2\ell})} + O(m + n)$ for our algorithm. We note that one can achieve a running time of $\ell^{O(2^{2\ell})} + O(n + m)$ by using a randomized algorithm that solves ILPs on p variables in time $\log(2p)^{O(p)}$[30]. ◀

W[1]-Hardness wrt. feedback edge number and solution size. We next prove that Bounded-Density Edge Deletion is W[1]-hard with respect to the combined parameter solution size and feedback edge number. To this end, we use the construction of Enciso et al. [9], who reduced Multicolored Clique to Equitable Connected Partition. These problems are defined as follows.

Multicolored Clique
Input: An undirected graph G properly colored with ℓ colors.
Question: Does G contain a clique on ℓ vertices?

Equitable Connected Partition
Input: An undirected graph G and a positive integer c.
Question: Is there a partition of G into c equally sized connected components, that is, a partition of $V(G)$ into $V_1, \dots, V_c$ so that each $G[V_i]$ is connected and $||V_i| - |V_j|| \leq 1$ for each $i, j \in [c]$?

Note that since G is properly colored any clique in G is multicolored, that is, contains at most one vertex per color.

As a byproduct, we also obtain W[1]-hardness for T_{h+1}-Free Edge Deletion parameterized by the combined parameter feedback edge number and solution size. This confirms a conjecture by Enright and Meeks [10]: T_{h+1}-Free Edge Deletion is indeed W[1]-hard with respect to treewidth.

Connection between problems. Let us briefly discuss the connections between the three problems to see why the construction of Enciso et al. [9] works for all three: Consider a cycle on $3n$ vertices. Requiring $c = 3$ for Equitable Connected Partition guarantees one solution (up to symmetry): delete 3 edges so that 3 paths on n vertices remain. Similarly, requiring $k = 3$ and $h = n$ for T_{h+1}-Free Edge Deletion guarantees the same solution. The same applies for Bounded-Density Edge Deletion with $\tau_\rho = (n-1)/n$ and $k = 3$.

The construction of Enciso et al. [9] will enforce the following: Any connected component in a solution will be a tree. Hence, we can control its maximum size via the density τ_ρ or directly via h. Combining the budget k with the size constraint ensures that the minimum number of vertices per component is equal to the maximum number of vertices per component. Thus, for the constructed graph all three problems will ask essentially for the same solution.

▶ **Theorem 15.** *Bounded-Density Edge Deletion and T_{h+1}-Free Edge Deletion are W[1]-hard with respect to the combined parameter solution size and feedback edge number.*

Proof. We use the same reduction as Enciso et al. [9]. Let (G, ℓ) be an instance of Multicolored Clique, which is W[1]-hard with respect to ℓ [11]. Assume without loss of generality that G contains exactly n/ℓ vertices of each color $i \in [\ell]$. (We can add isolated vertices of the appropriate color to G to meet this demand.) Denote with $\lambda\colon E(G) \to [m]$ a bijection that assigns each edge in G a unique integer from $[m]$. Further, denote with $v_1^i, \dots, v_{n/\ell}^i$ the vertices of color i in G.

Construct instances (H, k, τ_ρ) of Bounded-Density Edge Deletion and (H, k, h) of T_{h+1}-Free Edge Deletion as follows. Let α and β be the smallest integers satisfying $\beta > 2m + 10$ and $\alpha > \beta \cdot n/\ell + 2m + 10$. Then set $h = \alpha + \beta \cdot n/\ell + m + 1$ which will be the maximum number of vertices any subtree in the resulting graph $H - F$ is allowed to have. To ensure this, set $\tau_\rho = 1 - 1/h$.

The basic building block for H is the so-called *anchor*. For $q \geq 1$, a q-anchor is a vertex adjacent to $q - 1$ many vertices of degree one. The idea is that we cannot afford to cut off single vertices, thus all anchors in the constructed graph stay intact. Hence, a q-anchor acts as a single vertex with weight q (the vertex contributes q to the size of its connected component). We use anchors in the *choice* gadget (see Figure 2 for an illustration): Let $A = \{a_1, \dots, a_q\}$ be a set of integers so that $1 \leq a_i < a_j$ for all $1 \leq i < j$. An A-*choice* is a path on p vertices $u_1, \dots, u_p$ where vertex u_i, $i \in [p]$, is the center of an $(a_i - a_{i-1})$-anchor (u_1 is the center of an a_1-anchor). Note that an A-choice has exactly a_p many vertices (a_p is the maximum of A), and that removing the edge $\{u_i, u_{i+1}\}$ splits the A-choice gadget into two connected components with a_i and $a_q - a_i$ vertices respectively. We say we *cut* the A-choice at a_i, $i \in [|A| - 1]$, to indicate the removal of the edge $\{u_i, u_{i+1}\}$. To *connect* two vertices u and v by an A-choice means to merge the first and last anchor of the A-choice gadget with u and v, respectively. Merging a vertex v with an q-anchor means to remove the anchor, add q degree-one vertices adjacent only to v and make v adjacent to all vertices the center of the anchor was adjacent to. In other words, identify v with the center of the anchor and add one vertex only adjacent to v (to ensure the overall number of vertices stays the same). Note that connecting two vertices u and v by an A-choice still leaves $|A| - 1$ many possible cuts of the A-choice. We only connect center vertices of α-anchors by choice gadgets, thus enforcing a cut in each choice gadget.

For each of the ℓ colors in G add $2(\ell - 1)$ many α-anchors to the initially empty graph H. Note that α is sufficiently large to ensure that no two anchors can be in the same connected component, that is, $2\alpha > h$. Denote with N_j^i, P_j^i, $j \in [\ell] \setminus \{i\}$, the center vertices of the

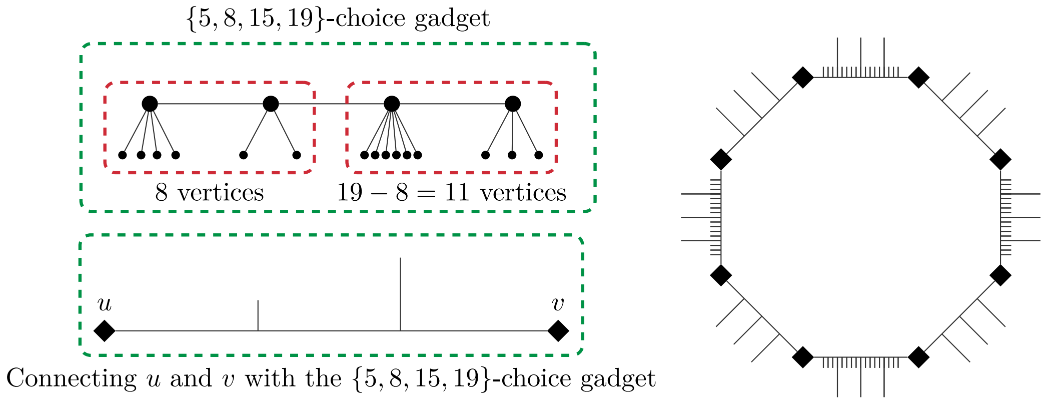

{5, 8, 15, 19}-choice gadget
8 vertices
19 − 8 = 11 vertices
u
v
Connecting u and v with the {5, 8, 15, 19}-choice gadget

Figure 2 *Left above:* A $\{5, 8, 15, 19\}$-choice gadget consisting of four anchors. The two red dashed boxed indicate a possible split into 8 and 11 vertices. *Left below:* A more compact representation (used in further figures) of the same gadget used to connect u and v. The length of the vertical lines correspond to the number of degree-one neighbors of the corresponding anchor; we omit the line for the first and last anchor. The diamond shaped vertices indicate α-anchors. Each choice-gadget in the construction connects two α-anchors as visualized above. *Right:* A gadget constructed for each color. It consists of $2(\ell - 1)$ many α-anchors (so $\ell = 5$ in the example) that are connected in a cycle via choice-gadgets (in the example there are 4 vertices per color).

anchors for color $i \in [\ell]$. Set $A_V = \{1\} \cup \{p\beta \mid p \in [n/\ell]\}$ containing $1 + n/\ell$ elements, thus accommodating one cut for each vertex of color i. Let j' be the "successor index" of j, formally:

$$j' = \begin{cases} 1, & \text{if } (j = \ell \wedge i \neq 1) \vee (j + 1 = i = \ell) \\ 2, & \text{if } j = \ell \wedge 1 = i \\ j + 1, & \text{if } j + 1 \neq i \wedge j + 1 \leq \ell \\ j + 2, & \text{if } j + 1 = i \wedge j + 2 \leq \ell \end{cases}$$

For each $j \in [\ell] \setminus \{i\}$, connect N_j^i and $P_{j'}^i$ by an A_V-choice gadget. Set $A_E^i = \{(p-1)\beta + \lambda(\{u, v_p^i\}) \mid p \in [n/\ell] \wedge u \in V(G) \wedge \{u, v_p^i\} \in E(G)\} \cup \{\beta n/\ell + m + 1\}$ containing one element for each edge incident to each vertex of color i and a large final element to make sure each A_E^i-choice has exactly $\beta n/\ell + m + 1$ many vertices. Next, connect P_j^i and N_j^i by an A_E^i-choice gadget. For $i, j \in [\ell], i \neq j$ connect P_j^i and N_i^j by an $[m + 1]$-choice gadget (leaving one m cut possibilities), see Figure 3 for an illustration and some intuition. Finally, set $k = 2(\ell - 1)\ell + 2\binom{\ell}{2} = 3(\ell - 1)\ell$.

Observe that removing in H the degree-one vertices, we are left with ℓ cycles that are pairwise connected by two paths. Thus, the feedback edge number of the constructed graph H is $\ell + 2\binom{\ell}{2} - (\ell - 1) = 2\binom{\ell}{2} + 1$.

We show that any solution for the instance (H, k, τ_ρ) creates connected components of exactly the same size. Let F be a solution to the constructed Bounded-Density Edge Deletion instance (H, k, τ_ρ), that is, $|F| \leq k$ and the densest subgraph in $G - F$ has density at most τ_ρ. Observe that each "large" α-anchor is in a different connected component in $H - F$ as $1/(1 - \tau_\rho) = h < 2\alpha$. Thus, F contains at least one edge of each choice gadget as each choice gadget connects two α-anchors. Since there are k choice gadgets, it follows that F contains exactly one edge from each choice gadget. Thus, $H - F$ contains exactly $2(\ell - 1)\ell$ many connected components. Note that H consists of $2(\ell - 1)\ell$ many α-anchors, $(\ell - 1)$ many

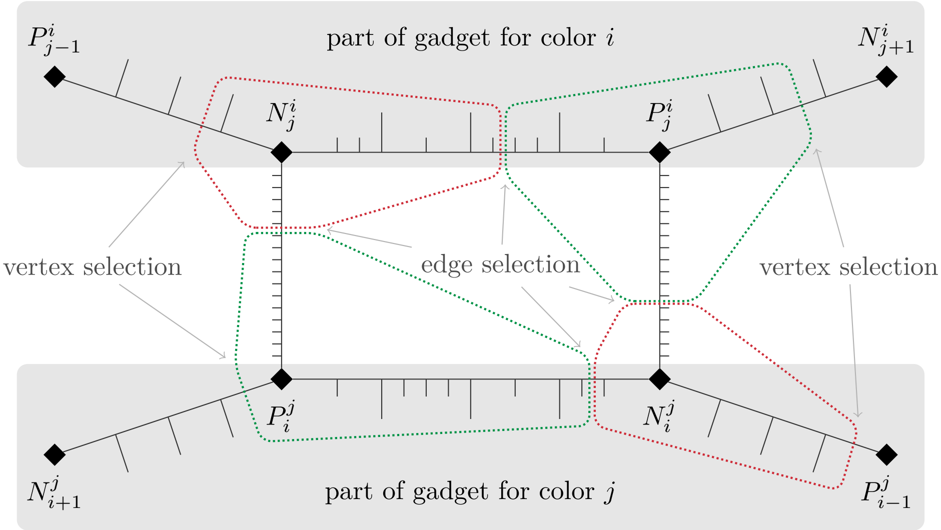


Figure 3 Illustration of the representation of edges of the original graph in the constructed graph. The top part (highlighted in gray) indicates part of the gadget (cycle) created for color i; correspondingly on the bottom for color j. The intuition for the reduction is as follows. Consider a solution for H, that is, a set of edges F so that $\rho^*(H - F) \leq \tau_\rho$. Some connected components in $H - F$ are indicated by dashed lines enclosing parts of the vertices. The size constraint for each connected component (enforced by the density threshold $\tau_\rho < 1$) ensures that each connected component of $H - F$ contains at most one α-anchor (indicated by diamond-shaped vertices), at most n/ℓ many β-anchors (indicated by the longer vertical lines in the choice-gadgets, here $n/\ell = 4$), and an additional $m + 1$ vertices. In the vertex-selection part (the A_V-choice gadgets that only contains β-anchors) there are n/ℓ many choices to cut; each corresponding to selecting one of the n/ℓ vertices of the respective color. The bound of at most n/ℓ many β-anchors per connected component enforces the same choice throughout the color gadget (see right side of Figure 2). The edge selection follows the same idea as the vertex selection for each color (the number are just smaller): The $[m+1]$-choice gadgets between N^i_j and P^j_i as well as between P^i_j and N^j_i imply that $2m+2$ vertices need to be distributed to the connected components around $N^i_j, P^j_i, P^i_j, N^j_i$. Moreover, the vertex pairs $\{N^i_j, P^i_j\}$ and $\{N^j_i, P^j_i\}$ are connected via an A^i_E- resp. A^j_E-choice gadget. Both of these gadgets contain $n/\ell + m + 1$ vertices. Thus, $4m + 4$ vertices needs to be distributed to the connected components around $N^i_j, P^j_i, P^i_j, N^j_i$. By construction, this requires a cut at an anchor representing the same edge in all four connections. This implies that the selected vertices in the color gadgets are adjacent in the original graph. As one vertex needs to be selected per color a clique needs to be selected.

A_V-choice and A^i_E-choice gadgets for each $i \in [\ell]$, and $2\binom{\ell}{2}$ many $[m+1]$-choice gadgets. Recall that a A-choice contains exactly $\max_{a \in A}\{a\}$ vertices. Thus, the number of vertices in H is

$$2(\ell-1)\ell\cdot\alpha+(\ell-1)\ell\cdot(2\beta n/\ell+m+1)+(\ell-1)\ell\cdot(m+1) = 2(\ell-1)\ell\cdot(\alpha+\beta n/\ell+m+1) = 2(\ell-1)\ell\cdot h.$$

Hence, by pigeon principle, each connected component in $H - F$ contains exactly h vertices. Thus, F certifies also a solution to the Equitable Connected Partition instance $(H, 2(\ell-1)\ell)$. The correctness of the reduction follows from the arguments of Enciso et al. [9], as they use the same reduction for Equitable Connected Partition. ◀

We remark that the above theorem also implies W[1]-hardness with respect to the treewidth for Bounded-Density Edge Deletion and T_{h+1}-Free Edge Deletion as the treewidth of is upper bounded by twice the feedback edge number. This resolves an open question of Enright and Meeks [10].

4 Bounded-Density Vertex Deletion

In this section we show that Bounded-Density Vertex Deletion is NP-complete on several graph classes. We also prove the W[2]-hardness with respect to the parameter k, the number of vertices to remove.

4.1 Polynomial-time algorithm for trees

In this part we discuss the case of trees and show that the problem is polynomial-time solvable. As in the edge deletion variant, we only need to consider the case that $\tau_\rho < 1$ (see the discussion before Theorem 7). Thus, we need to delete vertices such that each connected component in the resulting graph has at most $h = 1/(1 - \tau_\rho)$ many vertices.

Shen and Smith [32] consider the problem of deleting k vertices to minimize the size of the largest connected component. They established an algorithm in $O(n^3 \log n)$ time for trees. We improve on this as follows.

▶ **Theorem 16.** *Bounded-Density Vertex Deletion can be solved in $O(n)$ time on trees.*

Proof. Let T be the input tree. We propose a simple greedy algorithm that works bottom up from the leaves. In each vertex we visit we store the size (number of vertices) of the current subtree. For a vertex v this can be easily computed by summing up the values in its children and adding one. Whenever a vertex reaches subtree size above h, we delete the vertex. Naturally, when computing the subtree size, then deleted children will be ignored. This algorithm runs in $O(n)$ time.

Clearly, the algorithm provides a graph where each connected component contains at most h vertices. It remains to show that there is no smaller solution. To this end, consider a subtree T_v rooted at vertex v. If T_v contains more than h vertices, then any solution must delete at least one vertex in T_v. If T_v has size less than h, then we claim there is an optimal solution not deleting any vertex in T_v: Assume otherwise and let $S \subseteq V(T)$ be an optimal solution deleting a vertex u in T_v. Then removing u from S and adding the parent from v to S results in another solution of the same cost – a contradiction. Hence, our algorithm produces an optimal solution. ◀

4.2 NP-hardness results

We prove in the following the NP-hardness of Bounded-Density Vertex Deletion by reduction from Feedback Vertex Set, which remains NP-hard on Hamiltonian planar 4-regular graphs [5]. Given an instance of Feedback Vertex Set, that is a graph G, and an integer k, the problem consists in deciding the existence of a subset $V' \subseteq V(G)$ with $|V'| \leq k$ such that the $G - V'$ is cycle-free.

Bounded-Density Vertex Deletion is closely related to Feedback Vertex Set. Given a graph G and a subset $V' \subseteq V(G)$, we have that $G - V'$ is cycle-free if and only if $\rho^*(G - V') < 1$. Thus Bounded-Density Vertex Deletion is NP-complete on all classes of graphs where Feedback Vertex Set is NP-complete, for example on planar and maximum degree 4 graphs. We can even prove a stronger result as follows:

Let G be an undirected graph. The bipartite incidence graph of G (also called subdivision of G) is the bipartite graph H whose vertex set is $V(G) \cup E(G)$ and there is an edge in H between $v \in V(G)$ and $e \in E(G)$ if and only if e is incident to v in G.

▶ **Theorem 17.** *Bounded-Density Vertex Deletion is* NP*-complete for $\tau_\rho < 1$ even for planar bipartite graphs with maximum degree 4.*

Proof. We prove the NP-hardness by reduction from Feedback Vertex Set. Considering a graph G on n vertices and an integer k, instance of Feedback Vertex Set, let H be the bipartite incidence graph of G. Remark that if G is planar of maximum degree 4 then H is still planar with maximum degree 4. We show in the following that G contains a subset $V' \subseteq V(G)$ with $|V'| \leq k$ such that $G - V'$ is cycle-free if and only if H contains a subset $F \subseteq V(G) \cup E(G)$ with $|F| \leq k$ such that the $\rho^*(H - F) \leq 1 - 1/n^2$.

Note that to any cycle $v_1, v_2, \ldots, v_i, v_1$ in G it corresponds a cycle $v_1, v_1v_2, v_2, \ldots, v_i, v_iv_1, v_1$ in H. Moreover, to any cycle from H where the vertices alternate between vertices from $V(G)$ and $E(G)$, there corresponds a cycle in G. Furthermore, a subgraph $G[V']$ is cycle-free if and only if $\rho(G[V']) < 1$.

If G contains a subset $F \subseteq V$ with $|F| \leq k$ such that the $G - F$ is cycle-free, then $H - F$ contains no cycle and thus $\rho^*(H - F) \leq 1 - 1/n^2$.

Consider, now, that H contains a subset $F \subseteq V(G) \cup E(G)$ with $|F| \leq k$ such that $\rho^*(H - F) \leq 1 - 1/n^2$. Any vertex $e \in F \cap E(G)$ from H has two neighbors. We can replace it by one of its neighbors in F. This exchange makes the degree of e less than or equal to 1 in $H - F$, and therefore guarantees that e is not in any cycle. Thus, exchanging e in F for one of its neighbors cannot create any cycles, and moreover $|F| \leq k$. At the end of these exchanges, F contains only vertices from $V(G)$ and moreover $H - F$ contains no cycle. Thus F is a solution for the instance (G, k) of Feedback Vertex Set. ◀

In the next reduction we use the following problem:

Dissociation Set

Input:	A graph G and an integer k
Question:	Is there a subset of vertices $S \subseteq V(G)$ of size at least k such that $G[S]$ has maximum degree at most 1?

Dissociation Set is NP-hard even in line graphs of planar bipartite graphs [28].

▶ **Theorem 18.** *Bounded-Density Vertex Deletion is* NP*-complete for $\tau_\rho = 1/2$ even for line graphs of planar bipartite graphs.*

Proof. Given an instance (G, k') of Dissociation Set on planar line graphs of planar bipartite graphs we construct the instance $(G, k = n - k', \tau_\rho = 1/2)$ of Bounded-Density Vertex Deletion. We can easily see that G has a solution S of size at least k' if and only if $\rho^*(G[S]) \leq 1/2$ as $G[S]$ has density at most 1/2 and $V(G) \backslash S$ has size at most k. ◀

The two previous results show hardness for small values of τ_ρ. Next we show that Bounded-Density Vertex Deletion remains NP-hard for most values of τ_ρ.

▶ **Theorem 19** ($\star$). *Bounded-Density Vertex Deletion is* NP*-complete for any rational τ_ρ such that $0 \leq \tau_\rho \leq n^{1-1/c}$, where c is any constant.*

The last result in this subsection concerns split graphs. The following hardness result holds for large (non-constant) values of τ_ρ.

▶ **Theorem 20** ($\star$). *Bounded-Density Vertex Deletion remains NP-hard on split graphs.*

4.3 Parameterized complexity results

We show that Bounded-Density Vertex Deletion is FPT with respect to vertex cover number and W[2]-hard with respect to k, the number of vertices to remove.

▶ **Theorem 21.** *Bounded-Density Vertex Deletion can be solved in time $2^{O(\ell^2)}n^{O(1)}$ where ℓ is the vertex cover number.*

Proof. Let (G, k, τ_ρ) be an instance of Bounded-Density Vertex Deletion where G has vertex cover number ℓ. One can find a minimum vertex cover of size ℓ in time $O(2^\ell + n + m)$ [8]. Denote by C the set of vertices that belongs to the minimum vertex cover, and by $S = V(G) - C$ the set of vertices in the independent set. We divide the vertices in S into at most 2^ℓ classes $I_1, \ldots, I_{2^\ell}$, where two vertices $v_1, v_2 \in S$ are in the same class I_i if they have the same neighbors in C.

Notice that for $k \geq \ell$ we always have a yes-instance, as deleting the ℓ vertices in the vertex cover yields a graph with density 0. Thus, we are interested in the case where we delete at most $\ell - 1$ vertices, that is $k \leq \ell - 1$.

The vertices in each I_i (with $i = 1, \ldots, 2^\ell$) are all indistinguishable from each other, and we need to delete between 0 and $\ell - 1$ vertices S. Thus we need to check at most $(\ell 2^\ell)^\ell$ sets of vertices from S as candidates for deletion. For vertices in C we check all 2^ℓ subsets of vertices from C as candidates for deletion. In total, we check at most $2^\ell(\ell 2^\ell)^\ell$ subsets as candidates for deletion, yielding a running time of $2^{\ell^2+\ell\log\ell+\ell}n^{O(1)}$. ◀

▶ Remark 22. Note that if $\tau_\rho \geq \ell$, then nothing needs to be deleted: On the one hand, any vertex from the independent set has degree at most $\ell \leq \tau_\rho$. By Lemma 2 (4.) no vertex of the independent set belongs to a subgraph of density greater than τ_ρ. On the other hand, the vertex cover has density at most $(\ell - 1)/2 < \tau_\rho$.

The following hardness result holds for large (non-constant) values of τ_ρ.

▶ **Theorem 23** ($\star$)**.** *Bounded-Density Vertex Deletion is W[2]-hard with respect to the number k of vertices to remove.*

5 Conclusion

Our work provides a first (parameterized) analysis of Bounded-Density Edge Deletion and Bounded-Density Vertex Deletion. Both problems turn out to be NP-hard even in restricted cases but polynomial-time solvable on trees and cliques. While the W[1]-hardness for Bounded-Density Edge Deletion with respect to the feedback edge number rules out fixed-parameter tractability with respect to many other parameterizations, the respective reduction relies on a very specific target density. In fact, Figure 1 seems to indicate that the computational complexity of Bounded-Density Edge Deletion might change when altering the target density τ_ρ a bit. Does this alternating pattern between tractable and intractable target density extend beyond target density 2? For example the polynomial-time solvable f-Factor problem might be helpful in designing polynomial-time algorithms for Bounded-Density Edge Deletion with $\tau_\rho \in \mathbb{N}$ (any $2\tau_\rho$-regular subgraph would be a valid resulting graph $G - F$). Could such behavior be exploited in approximation algorithms?

We leave also several open questions concerning the parameterized complexity of these problems. Is Bounded-Density Vertex Deletion fixed-parameter tractable with respect to the treewidth (plus the solution size)? What is the parameterized complexity with respect to parameters that are smaller than the vertex cover number, e. g., vertex integrity or twin cover number?

References

1 Béla Bollobás. *Random Graphs*, pages 215–252. Springer New York, New York, NY, 1998. doi:10.1007/978-1-4612-0619-4_7.

2 Marthe Bonamy, Benjamin Lévêque, and Alexandre Pinlou. Graphs with maximum degree $\Delta \geq 17$ and maximum average degree less than 3 are list 2-distance $(\Delta + 2)$-colorable. *Discrete Mathematics*, 317:19–32, 2014.

3 Oleg V Borodin, Alexandr Kostochka, and Matthew Yancey. On 1-improper 2-coloring of sparse graphs. *Discrete Mathematics*, 313(22):2638–2649, 2013.

4 Oleg V Borodin and Alexandr V Kostochka. Vertex decompositions of sparse graphs into an independent vertex set and a subgraph of maximum degree at most 1. *Siberian mathematical journal*, 52(5):796–801, 2011.

5 Dario Cavallaro and Till Fluschnik. Feedback vertex set on hamiltonian graphs. In *47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2021)*, volume 12911 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2021. doi:10.1007/978-3-030-86838-3_16.

6 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Comput. Sci. Rev.*, 48:100556, 2023. doi:10.1016/J.COSREV.2023.100556.

7 William H. Cunningham. Optimal attack and reinforcement of a network. *J. Assoc. Comput. Mach.*, 32(3):549–561, 1985.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad A. Kanj, Frances A. Rosamond, and Ondrej Suchý. What makes equitable connected partition easy. In *In Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *LNCS*, pages 122–133. Springer, 2009.

10 Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: a new application for treewidth. *Algorithmica*, 80:1857–1889, 2018.

11 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

12 András Frank and Éva Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

13 Vincent Froese, André Nichterlein, and Rolf Niedermeier. Win-win kernelization for degree sequence completion problems. *J. Comput. Syst. Sci.*, 82(6):1100–1111, 2016. doi:10.1016/J.JCSS.2016.03.009.

14 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. Association for Computing Machinery.

15 A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, USA, 1984.

16 Arthur M. Hobbs. Network survivability. In *Applications of discrete mathematics*, pages 332–353. McGraw-Hill, New York, 1991.

17 Lavanya Kannan, Arthur Hobbs, Hong-Jian Lai, and Hongyuan Lai. Transforming a graph into a 1-balanced graph. *Discrete Appl. Math.*, 157(2):300–308, 2009.

18 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

19 Michael Kopreski and Gexin Yu. Maximum average degree and relaxed coloring. *Discrete Mathematics*, 340(10):2528–2530, 2017.

20 G. Laman. On graphs and rigidity of plane skeletal structures. *J. Engrg. Math.*, 4:331–340, 1970.

21 Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants. *CoRR*, abs/2303.14467, 2023. `doi:10.48550/arXiv.2303.14467`.

22 H. W. Lenstra. Integer Programming with a Fixed Number of Variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. URL: `http://www.jstor.org/stable/3689168`.

23 Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *J. Comput. Syst. Sci.*, 78(1):179–191, 2012. `doi:10.1016/J.JCSS.2011.02.001`.

24 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|)|e|) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27, 1980. `doi:10.1109/SFCS.1980.12`.

25 Wojciech Nadara and Marcin Smulewicz. Decreasing the Maximum Average Degree by Deleting an Independent Set or a d-Degenerate Subgraph. *Electron. J. Comb.*, 29(1), 2022.

26 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Prolegomena*, pages 21–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

27 André Nichterlein. *Degree-constrained editing of small-degree graphs.* PhD thesis, Berlin Institute of Technology, 2015. URL: `https://opus4.kobv.de/opus4-tuberlin/frontdoor/index/index/docId/6520`.

28 Yury Orlovich, Alexandre Dolgui, Gerd Finke, Valery Gordon, and Frank Werner. The complexity of dissociation set problems in graphs. *Discrete Appl. Math.*, 159(13):1352–1366, 2011.

29 Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982. `doi:10.1002/NET.3230120206`.

30 Victor Reis and Thomas Rothvoss. The subspace flatness conjecture and faster integer programming. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023. `doi:10.1109/FOCS57990.2023.00060`.

31 Andrzej Ruciński and Andrew Vince. Strongly balanced graphs and random graphs. *Journal of graph theory*, 10(2):251–264, 1986.

32 Siqian Shen and J Cole Smith. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks*, 60(2):103–119, 2012.

33 Wenying Xi and Wensong Lin. On maximum p3-packing in claw-free subcubic graphs. *J. Comb. Optim.*, 41(3):694–709, April 2021.

The Simultaneous Interval Number

A New Width Parameter that Measures the Similarity to Interval Graphs

Jesse Beisegel
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Nina Chiarelli
FAMNIT and IAM, University of Primorska, Koper, Slovenia

Ekkehard Köhler
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Martin Milanič
FAMNIT and IAM, University of Primorska, Koper, Slovenia

Peter Muršič
FAMNIT, University of Primorska, Koper, Slovenia

Robert Scheffler
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Abstract

We propose a novel way of generalizing the class of interval graphs, via a graph width parameter called simultaneous interval number. This parameter is related to the simultaneous representation problem for interval graphs and defined as the smallest number d of labels such that the graph admits a d-simultaneous interval representation, that is, an assignment of intervals and label sets to the vertices such that two vertices are adjacent if and only if the corresponding intervals, as well as their label sets, intersect. We show that this parameter is NP-hard to compute and give several bounds for the parameter, showing in particular that it is sandwiched between pathwidth and linear mim-width. For classes of graphs with bounded parameter values, assuming that the graph is equipped with a simultaneous interval representation with a constant number of labels, we give FPT algorithms for the clique, independent set, and dominating set problems, and hardness results for the independent dominating set and coloring problems. The FPT results for independent set and dominating set are for the simultaneous interval number plus solution size. In contrast, both problems are known to be W[1]-hard for linear mim-width plus solution size.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Interval graph, simultaneous representation, width parameter, algorithm, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.7

Related Version *Full Version*: https://arxiv.org/abs/2404.10670 [1]

Funding The authors acknowledge partial support by the Slovenian Research and Innovation Agency (I0-0035, research programs P1-0285 and P1-0404, and research projects N1-0160, N1-0210, J1-3001, J1-3002, J1-3003, J1-4008, and J1-4084) and by the research program CogniCom (0013103) at the University of Primorska.

Acknowledgements The authors would like to thank the reviewers for their helpful remarks; in particular, for their hints on thinness and boxicity.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 7; pp. 7:1–7:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Interval graphs are among the best-known and most studied graph classes, due to their intuitive representation with an interval intersection model, their rich structure, and many algorithmic advantages. Many problems that are NP-hard on general graphs can be solved in polynomial time on interval graphs. Examples are the coloring problem [35, 37, 47], the dominating set problem [12], and the Hamiltonian cycle problem [46]. Furthermore, due to their definition via interval representations, there are plenty of real-world applications for interval graphs (see [45] for a nice, short overview of such applications).

There are several different ways to generalize the concept of an interval graph. One of these concepts are the so-called *d-interval graphs* where every vertex is represented by a set of d intervals on the real line and two vertices are adjacent if any pair of their intervals intersect. A subclass of these graphs are the *d-track interval graphs* where we have d parallel lines and every vertex is represented by d intervals, one on each line. It is easy to see that any graph is a d-track interval graph (and, thus, a d-interval graph) for some d. Therefore, it makes sense to define the parameters *interval number* $\mathsf{i}(G)$ [34] and *track number* $\mathsf{t}(G)$ [36] as the minimal numbers d such that G is a d-interval (resp. d-track interval) graph. There is some work on these graph classes concerning parameterized complexity [24, 44]. However, most of the classical graph problems are NP-hard for graphs with $\mathsf{i}(G) = 2$ or $\mathsf{t}(G) = 3$ [21, 22]. Furthermore, even the independent set problem and the dominating set problem are $\mathsf{W}[1]$-hard when parameterized by the solution size for graphs with $\mathsf{t}(G) = 2$ [44].

Another way to define a whole family of generalizations of interval graphs comes from the so-called simultaneous representation problems. In this generalization, we are given d interval graphs $G_1, \dots, G_d$ which may share some vertices and asks for an interval representation that assigns to every vertex in $V(G_1) \cup \dots \cup V(G_d)$ exactly one interval such that for every $i \in \{1, \dots, d\}$ two vertices of G_i are adjacent if and only if their intervals intersect. The problem of deciding whether a given set of graphs has such a simultaneous representation was introduced in 2009 by Jampani and Lubiw [41], where they considered chordal graphs, comparability graphs, and permutation graphs, all classes of graphs that can also be defined via certain intersection representations. A year later, the same authors considered the problem of simultaneous interval representations [42]. Since then, there has been several results on the complexity of this problem for different classes of graphs [4, 7, 58].

An equivalent definition for a simultaneous interval representation can be given as follows: For some interval model we add additional label sets in the form of subsets of $\{1, \dots, d\}$ and two vertices belonging to two intervals are adjacent if these intervals intersect and the intersection of their label sets is non-empty. This definition leads to an intuitive application in scheduling, where each of the labels $1, \dots, d$ represents some machine and an interval represents a job with its processing window (the interval) and the set of machines needed to perform the job (the label set). An independent set in such a graph would then represent a conflict-free schedule of a subset of jobs.

Similar to d-interval graphs and d-track interval graphs, any graph can be defined as a d-simultaneous interval graph for some d. Thus, we can introduce the *simultaneous interval number* $\mathsf{si}(G)$ as the smallest number d for which G is a d-simultaneous interval graph. Many width parameters are unbounded for interval graphs, as these tend to grow with the clique number (for example treewidth/pathwidth is unbounded for interval graphs). Furthermore, even width parameters that can be bounded for dense graphs, such as cliquewidth or twin-width, are unbounded for interval graphs [8, 31]. On the other hand, those parameters that are bounded for interval graphs, such as linear mim-width or tree-independence number

Table 1 Parameterized complexity summary. Abbreviations mean ind → independent, dom → dominating, W[1] → W[1]-hard, W[2] → W[2]-hard, pNPh → para-NP-hard, tree-α → tree-independence number. Green results are given in this paper. Hardness results for problems with given solution size k means that the problem is hard when parameterized by $p+k$. For space reasons, we omitted the $\mathcal{O}$ and $\mathcal{O}^*$ notations in the running time bounds.

problem\parameter	$p = \mathsf{si}(G)$	$p = \mathsf{linear\text{-}mim}(G)$	$p = \mathsf{tree}\text{-}\alpha(G)$	$p = \mathsf{t}(G)$
clique	$p2^{2^p+2p}$	pNPh [60]	pNPh [23]	pNPh [26]
clique of size k	2^{kp}	?	2^{k^p} [14, 19]	$p^k k^{k+2}$ [24]
coloring	pNPh	pNPh [28]	pNPh [28]	pNPh [28]
k-coloring	k^{kp}	n^{kp} [32]	k^{k^p} [14, 19]	pNPh [22]
ind set	n^p	W[1]/n^{2p} [25, 39]	n^p [19, 62]	pNPh [22]
ind set of size k	2^{kp}	W[1]/n^{2p} [25, 39]	?	W[1] [24]
dom set	n^{2p}	W[1]/n^{2p} [25, 39]	pNPh [3, 16]	pNPh [22]
dom set of size k	2^{kp}	W[1]/n^{2p} [25, 39]	W[2] [49]	W[1] [24]
ind dom set	W[1]/n^{2p}	W[1]/n^{2p} [25, 39]	?	pNPh [22]
ind dom set of size k	n^{2p}	W[1]/n^{2p} [25, 39]	?	W[1] [24]

(see [19, 40]), do not properly reflect the structural advantages of interval graphs. Many of the problems that are easy for interval graphs, such as coloring or independent set, are either para-NP-hard or W[1]-hard (see Table 1). Furthermore, the maximum clique problem is para-NP-hard when parameterized by one of those parameters, even though the structure of the maximal cliques is very restricted for interval graphs.

When parameterized by the simultaneous interval number, however, the maximum clique problem becomes FPT, as we will show. In addition, some of the problems that are W[1]-hard when parameterized by linear mim-width plus solution size, such as independent set and dominating set (see [25, 39]), are FPT when parameterized by simultaneous interval number plus solution size. Therefore, we argue that the simultaneous interval number is a strong candidate to fill the gap in describing graphs with a structure similar to interval graphs.

Our Contribution. We introduce a new graph width parameter, the *simultaneous interval number*, in Section 2. This parameter is compared to most of the other common width parameters such as treewidth, cliquewidth, or mim-width in Section 3, where we also give several bounds involving the order and the size of the graph, the edge clique cover number, the clique number, and other width parameters. In Section 4 we show that the computation of the simultaneous interval number is NP-hard. Furthermore, we give results on the parameterized complexity of several graph problems, such as clique (Section 5), coloring (Section 6), and variants of the independent set and dominating set problems (Section 7). For an overview of these results see Table 1. Proofs omitted due to lack of space can be found in the full version [1].

Definitions and Notation. Unless stated otherwise, all the graphs considered are simple, finite, non-empty and undirected. Given a graph G, we denote by $V(G)$ its vertex set and by $E(G)$ its edge set. Often we will denote the number of vertices of graph, i.e., $|V(G)|$, as n and the number of edges, i.e., $|E(G)|$, as m. A *matching* in a graph is a set of pairwise disjoint edges; a matching is *induced* if no two vertices belonging to different edges of the matching are adjacent.

Next we define the term *class of intersection graphs*. Such a graph class $\mathcal{C}$ can be defined via a family $S_{\mathcal{C}}$ of sets whose elements are also families of sets. For the sake of convenience, we assume that $S_{\mathcal{C}}$ contains a set family that contains a non-empty set. A *$\mathcal{C}$-representation* of a graph G is a mapping $R : V(G) \to \mathcal{F}$ where $\mathcal{F} \in S_{\mathcal{C}}$ such that $xy \in E(G)$ if and only if $R(x) \cap R(y) \neq \emptyset$. We call $\mathcal{F}$ the *ground set family* of R. By definition, $\mathcal{C}$ consists precisely of graphs G having a $\mathcal{C}$-representation.

The class of *chordal graphs* is defined via the set $S_{\mathcal{C}}$ that contains for every tree the set of its subtrees. For the class of *interval graphs*, the set $S_{\mathcal{C}}$ contains only the one set family, namely the set of all open intervals of the real line. For any interval representation R of graph G, we define $\ell(v)$ and $r(v)$ to be the left and right endpoints of the interval $R(v)$.

A graph G is a *bipartite graph* if its vertex set can be partitioned into two independent sets A and B. Furthermore, a bipartite graph is *complete bipartite* if every vertex of A is adjacent to every vertex of B. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. A graph is a *complete split graph* if there exists a partition in which every vertex of the independent set is adjacent to all the vertices of the clique. A graph is *C_4-free* if it does not contain an induced cycle of length 4.

2 Simultaneous Representations and Simultaneous Interval Number

In [41, 43], Jampani and Lubiw introduce the concept of *simultaneous representations* as well as the *simultaneous representation problem*. This concept was then taken up by Bok and Jedličková [7] who give the following definition:

▶ **Definition 2.1.** *Let $\mathcal{C}$ be a class of intersection graphs. Graphs $G_1, \dots G_d \in \mathcal{C}$ are* simultaneously $\mathcal{C}$-representable *if there exist $\mathcal{C}$-representations $R_1, \dots, R_d$ of $G_1, \dots G_d$ with a common ground set family $\mathcal{F} \in S_{\mathcal{C}}$ such that*

$$\forall i, j \in \{1, \dots, d\}, \forall v \in V(G_i) \cap V(G_j) \colon R_i(v) = R_j(v).$$

*In particular, we say that $G = G_1 \cup \ldots \cup G_d$ is a d-*simultaneous $\mathcal{C}$-graph.

For convenience of notation, we will oftentimes use the following equivalent definition of a simultaneous representation.

▶ **Definition 2.2.** *Let $d \in \mathbb{N}$, let G be a graph, and let $L : V(G) \to \mathcal{P}(\{1, \dots, d\})$ be a labeling of the vertices of G. Furthermore, let $G' \in \mathcal{C}$ with $V(G) = V(G')$ and $E(G) \subseteq E(G')$ be a graph with a $\mathcal{C}$-representation R. We say that (R, L) is a d-*simultaneous $\mathcal{C}$-representation *of G if it holds that $vw \in E(G)$ if and only if $R(v) \cap R(w) \neq \emptyset$ and $L(v) \cap L(w) \neq \emptyset$.*

Note that this definition allows the *empty set* as a label set. Obviously, any vertex with an empty label set is isolated. Therefore, the graphs admitting a 0-simultaneous $\mathcal{C}$-representation are exactly the edgeless graphs.

▶ **Observation 2.3.** *Let $\mathcal{C}$ be a class of intersection graphs. Let the graphs $G_1, \dots, G_d \in \mathcal{C}$ be simultaneously $\mathcal{C}$-representable with $\mathcal{C}$-representations $R_1, \dots, R_d$ with a common ground set family $\mathcal{F}$. Let $G := G_1 \cup \dots \cup G_d$ and let $R \colon V(G) \to \mathcal{F}$ be defined as $R(v) := R_i(v)$ for any i with $v \in V(G_i)$. Let L be the labeling given by $L(v) = \{i \colon v \in G_i\}$ for all $v \in V(G)$. Then (R, L) is a d-simultaneous $\mathcal{C}$-representation of G.*

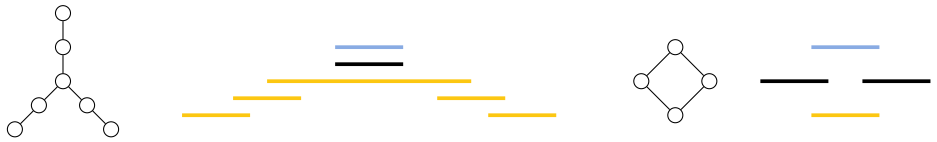


Figure 1 Two forbidden induced subgraphs of interval graphs with 2-simultaneous interval representations. Yellow intervals have label set $\{1\}$, blue intervals have label set $\{2\}$ and black intervals have label set $\{1, 2\}$. Note that the representation of the 4-cycle can be extended to a 2-simultaneous interval representation of cycles of arbitrary length.

This observation implies that every d-simultaneous $\mathcal{C}$-graph has a d-simultaneous $\mathcal{C}$-representation. However, the converse is not true in general.[1] However, if we exclude empty label sets and unused labels, then there is an analogous result to Observation 2.3.

▶ **Observation 2.4.** *Let (R, L) be a d-simultaneous $\mathcal{C}$-representation of a graph G with $L(v) \neq \emptyset$ for all $v \in V(G)$ and such that for all $i \in \{1, \dots, d\}$ there exists a vertex v with $i \in L(v)$. Let G_i be the subgraph of G induced by the vertex set $\{v : i \in L(v)\}$ and let R_i be the restriction of R to $V(G_i)$. Then the graphs $G_1, \dots, G_d$ are simultaneously $\mathcal{C}$-representable with $\mathcal{C}$-representations $R_1, \dots, R_d$.*

A vertex with an empty label set would have to be considered as a vertex that is in none of the graphs of a simultaneous representation. However, this technical addition to the definition is very useful to address the issue of isolated vertices and leads to more compact statements and simpler proofs. For all of the classes considered here, it is always possible to represent isolated vertices without the empty label set. For example, for interval graphs we can always represent such a vertex with an interval that intersects nothing else. However, in general we cannot assume that this is possible for any class of intersection graphs (see Footnote 1).

▶ **Theorem 2.5.** *For every class of intersection graphs $\mathcal{C}$, every graph G has an $|E(G)|$-simultaneous $\mathcal{C}$-representation.*

In particular, this theorem holds for the class of intervals graphs, motivating the following definition.

▶ **Definition 2.6.** *Let G be a graph. The* simultaneous interval number $\mathsf{si}(G)$ *of G is the smallest integer d such that there exists a d-simultaneous interval representation of G.*

As observed before, the graphs with simultaneous interval number 0 are exactly the edgeless graphs. Furthermore, the graphs with simultaneous interval number at most 1 are exactly the interval graphs, and the class of graphs with the simultaneous interval number equal to 2 contains some asteroidal triples and all cycles (see Figure 1).

In the following, we show some bounds on the simultaneous interval number. The first result is implied directly by Theorem 2.5.

▶ **Corollary 2.7.** *For any graph G it holds that $\mathsf{si}(G) \leq |E(G)|$.*

Next we show that this bound is tight, up to a constant factor.

[1] As an example, we consider the class $\mathcal{K}$ of complete graphs which can be represented as intersection graphs via the set $S_{\mathcal{K}} = \{\{1\}\}$. The n-vertex edgeless graph has a 1-simultaneous $\mathcal{K}$-representation where all vertices are labeled with the empty set. However, it is not a d-simultaneous $\mathcal{K}$-graph for any d.

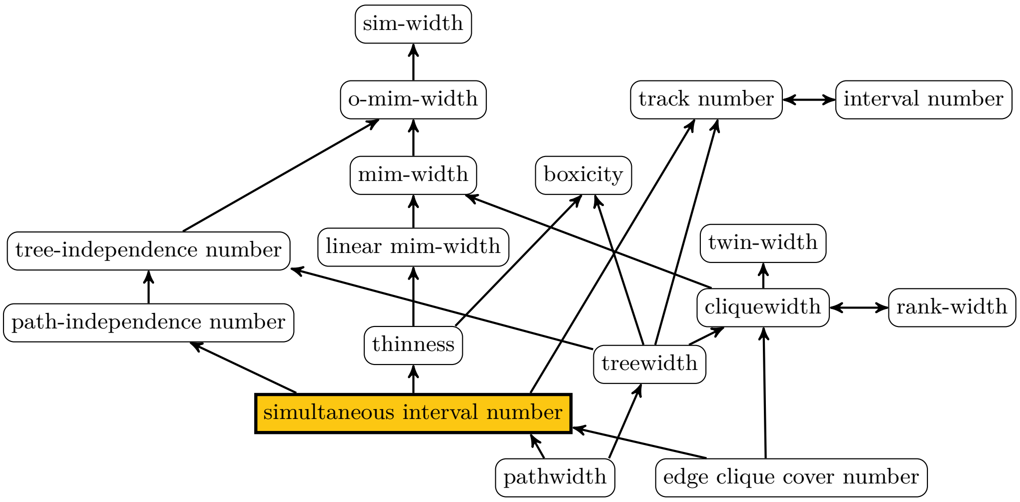


Figure 2 Diagram illustrating the relations between different graph width parameters. A directed edge from parameter P to parameter Q means that a bounded value of P implies a bounded value for Q. If a directed path from P to Q is missing, then parameter Q is unbounded for the graphs of bounded P.

▶ **Theorem 2.8.** *Let G be a complete 3-partite graph with parts of equal size. Then,* $\mathsf{si}(G) = \frac{1}{9}|V(G)|^2 = \frac{1}{3}|E(G)|$.

▶ **Theorem 2.9.** *Let $G = (V, E)$ be a bipartite graph with a bipartition $V = X \dot{\cup} Y$. Then* $\mathsf{si}(G) \leq \min\{|X|, |Y|\}$. *This bound is tight for complete bipartite graphs.*

The *complement of a matching* is a graph obtained from a complete graph of even order n by removing from it $\frac{n}{2}$ pairwise disjoint edges.

▶ **Lemma 2.10.** *If G is the complement of a matching with n vertices, then* $\mathsf{si}(G) \geq \log_2(n-1)$.

We will see later, in Lemma 5.5, that this bound is tight.

3 Placing si(G) in the Zoo of Graph Width Parameters

In this section we compare the simultaneous interval number to several other graph width parameters. See Figure 2 for an overview. A verification of the figure can be found in the full version [1].

3.1 Lower Bounds

It is easy to see that d-simultaneous interval graphs are d-track interval graphs. This implies the following result.

▶ **Theorem 3.1.** *Every graph satisfies* $\mathsf{t}(G) \leq \mathsf{si}(G)$.

The concept of *thinness* was introduced by Mannino et al. [51].

▶ **Definition 3.2** (Thinness). *The* thinness $\mathsf{thin}(G)$ *of a graph G is the smallest integer k such that there is a partition $\{V_1, \ldots, V_k\}$ of $V(G)$ and a vertex ordering $(v_1, \ldots, v_n)$ of G fulfilling that for any three vertices v_a, v_b, v_c with $a < b < c$ and $v_a, v_b \in V_i$ for some i it holds that $v_b v_c \in E(G)$ if $v_a v_c \in E(G)$.*

▶ **Theorem 3.3.** *For any graph G it holds that* $\mathsf{thin}(G) \leq 2^{\mathsf{si}(G)}$.

Complements of matchings with n edges have thinness n [13]. We will later see in Lemma 5.5 that the simultaneous interval number of such a graph is $\mathcal{O}(\log n)$. This implies that the bound given in Theorem 3.3 is asymptotically sharp. Bipartite permutation graphs and, hence, also complete bipartite graphs have thinness at most 2 [10]. As we have seen in Theorem 2.9, the simultaneous interval number of complete bipartite graphs is unbounded. Therefore, this class shows that bounded thinness does not imply bounded simultaneous interval number.

The concept of a linearized version of mim-width was introduced by Vatshelle [60] as mim-width using a caterpillar decomposition. This concept has since been called *linear mim-width* (for example by Golovach et al. [30]).

▶ **Definition 3.4** (Linear mim-width). *Given a graph G and a vertex ordering $\sigma = (v_1, \ldots, v_n)$ of G, we define the quantity* $\mathsf{linear\text{-}mim}(G, \sigma, i)$ *for $1 \leq i \leq n$ to be the maximum size of an induced matching in the bipartite graph that contains all the edges of G between the two sets $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$. We define* $\mathsf{linear\text{-}mim}(G, \sigma) := \max_{i \in \{1,\ldots,n\}} \mathsf{linear\text{-}mim}(G, \sigma, i)$*. The* linear mim-width *of G, denoted* $\mathsf{linear\text{-}mim}(G)$*, is defined as the minimum value* $\mathsf{linear\text{-}mim}(G, \sigma)$ *among all vertex orderings σ of G.*

It was shown by Bonomo and de Estrada [9] that for any graph G it holds that $\mathsf{linear\text{-}mim}(G) \leq \mathsf{thin}(G)$. Combining this with Theorem 3.3 we see that bounded simultaneous interval number also implies bounded linear mim-width. Moreover, using a more direct argumentation, the lower bound on the simultaneous interval number given by the logarithm of the linear mim-width can be improved to a linear lower bound.

▶ **Theorem 3.5.** *For any graph G it holds that* $\mathsf{linear\text{-}mim}(G) \leq \mathsf{si}(G)$.

A *tree decomposition* of a graph G is a pair $(T, \{X_t\}_{t \in V(T)})$ consisting of a tree T and a mapping asigning to each node $t \in V(T)$ a set $X_t \subseteq V(G)$ (called a *bag*) such that the following conditions are satisfied: (i) the union of all the bags equals $V(G)$, (ii) for every edge $uv \in E(G)$ there exists a bag X_t such that $u, v \in X_t$, and (iii) for every vertex $v \in V(G)$ the bags containing v form a subtree of T. A *path decomposition* of G is a tree decomposition of G such that T is a path. For simplicity, we will denote a path decomposition simply by the corresponding sequence $\mathcal{P} = (X_1, \ldots, X_k)$ of bags. Note also that in this case, condition (iii) simplifies to: for every vertex $v \in V(G)$ the bags containing v form a consecutive subsequence of $\mathcal{P}$. The *width* of a tree decomposition is the maximal size of its bags minus 1. The *treewidth* of a graph G, denoted by $\mathsf{tw}(G)$, is the minimum width of a tree decomposition of G. The *pathwidth*, denoted by $\mathsf{pw}(G)$, is defined analogously, with respect to path decompositions. Yolov [62] and independently Dallard et al. [19] introduced the parameter called tree-independence number (or *α-treewidth*). The *independence number* of a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of a graph G is defined as the maximum cardinality of an independent set I in G such that there exists a bag X_t with $I \subseteq X_t$, or, equivalently, the maximum, over all bags X_t, of the independence number of the subgraph of G induced by the bag X_t. The *tree-independence number* of a graph G, denoted by $\mathsf{tree\text{-}}\alpha(G)$, is defined as the minimum independence number of a tree decomposition of G. We define the *path-independence number*, denoted by $\mathsf{path\text{-}}\alpha(G)$, analogously, with respect to path decompositions.

We now prove a characterization of the path-independence number, which relies on the concept of the *intersection* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, denoted by $G_1 \cap G_2$ and defined as the graph $(V_1 \cap V_2, E_1 \cap E_2)$. In the proof we will use the following two known facts about path decompositions and interval graphs (see [6, 27]):

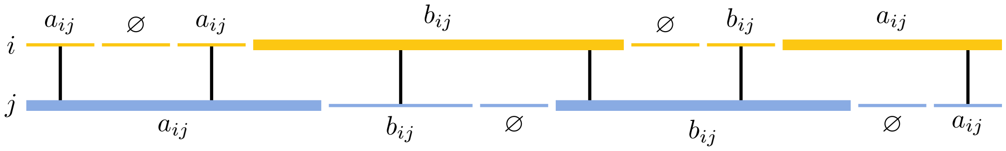


Figure 3 Illustration of the proof of Theorem 3.9. The thick intervals mark the active intervals. A black edge between two intervals means that the corresponding vertices are adjacent in G. The symbol $\varnothing$ means that the corresponding vertex has neither label a_{ij} nor the label b_{ij}. However, the vertex will have other labels.

- Let G be a graph, let $\mathcal{P}$ be a path decomposition of G, and let $S \subseteq V(G)$ be a set of vertices of G such that for every two vertices $u, v \in S$ there exists a bag X_i of P such that $u, v \in X_i$. Then there exists a bag X_j of $\mathcal{P}$ such that $S \subseteq X_j$.
- A graph G is an interval graph if and only if it admits a path decomposition in which each bag is a clique in G.

▶ **Theorem 3.6.** *Let G be a graph. Then, the path-independence number of G equals the minimum integer $k \geq 0$ such that G is the intersection of an interval graph and a graph with independence number at most k.*

With a similar approach as that used to prove Theorem 3.6, it can be proved that the tree-independence number of a graph G equals the minimum integer $k \geq 0$ such that G is the intersection of a chordal graph and a graph with independence number at most k.

Theorem 3.6 has the following consequence.

▶ **Corollary 3.7.** *Every graph G satisfies* $\mathsf{path}\text{-}\alpha(G) \leq \mathsf{si}(G)$.

Note that complements of matchings have independence number 2 and, thus, also path-independence number at most 2. Due to Lemma 2.10, they form a class of graphs with bounded path independence number but unbounded simultaneous interval number.

In the spirit of Dallard et al. [18], we can also show that graphs with bounded simultaneous interval number are (pw, ω)-bounded (and consequently (tw, ω)-bounded; note that Chaplick et al. [14] refer to the same property as the *clique-treewidth property*). A graph class $\mathcal{G}$ is said to be (pw, ω)*-bounded* (resp., (tw, ω)*-bounded*) if there is a function f such that for all graphs $G \in \mathcal{G}$ and all induced subgraphs G' of G, it holds that $\mathsf{pw}(G') \leq f(\omega(G'))$ (resp., $\mathsf{tw}(G') \leq f(\omega(G'))$), where $\omega(G')$ is the clique number of G'.

▶ **Theorem 3.8.** *Every graph G satisfies* $\mathsf{pw}(G) \leq \mathsf{si}(G)\omega(G) - 1$.

3.2 Upper Bounds

We begin our discussion on upper bounds by proving that bounded pathwidth implies bounded simultaneous interval number.

▶ **Theorem 3.9.** *Every graph G satisfies* $\mathsf{si}(G) \leq \mathsf{pw}(G)^2 + \mathsf{pw}(G)$.

Proof. Let $k := \mathsf{pw}(G) + 1$. Consider a path decomposition $\mathcal{P}$ of G with maximal bag size k. It is easy to see that we can transform $\mathcal{P}$ in such a way that every bag has size k. Furthermore, we can ensure that two consecutive bags differ only in two vertices, i.e., both vertices are part of exactly one of the two bags and all the other vertices are part of both bags or of none of them. This can be done by adding a sequence of new bags between two old ones in which the vertices are removed and introduced one by one. Let $\mathcal{P}' = (X_1, \ldots, X_p)$ be the resulting path decomposition of G. Now there exists a mapping $f : V(G) \to \{1, \ldots, k\}$ such that every bag of $\mathcal{P}'$ contains a vertex v with $f(v) = i$ for all $i \in \{1, \ldots, k\}$. For every vertex of G, we define the interval $R(v)$ as $(a - \varepsilon, b + \varepsilon)$ where $0 < \varepsilon < \frac{1}{2}$, a is the smallest index such that X_a contains v and b is the largest index such that X_b contains v. It follows that the intervals of two vertices have a non-empty intersection if and only if these vertices are part of a common bag. Therefore, intervals of vertices with the same f-value have an empty intersection.

It remains to show that we can label the vertices with at most $k \cdot (k-1)$ labels in such a way that the defined intervals form a simultaneous interval representation of G. For every set $\{i, j\} \subseteq \{1, \ldots, k\}$ with $i \neq j$, we introduce labels a_{ij} and b_{ij}. Note that $a_{ij} = a_{ji}$ and $b_{ij} = b_{ji}$. In the following we describe a procedure how to label the vertices of G (see Figure 3 for an illustration). During that labeling procedure, we will always have one *active vertex* $\hat{v}$ and one *active label* $c_{ij} \in \{a_{ij}, b_{ij}\}$. To define the first active vertex let x be the vertex with $f(x) = i$ whose interval ends first and let y be the vertex with $f(y) = j$ whose interval ends first. Without loss of generality, we may assume that $r(x) < r(y)$. We define the first active vertex $\hat{v}$ to be y. The first active label c_{ij} is a_{ij}. The active vertex $\hat{v}$ gets the label c_{ij}. For all vertices z with $f(z) \in \{i, j\} \setminus f(\hat{v})$ and $\ell(\hat{v}) < r(z) < r(\hat{v})$, we add c_{ij} to $L(z)$ if and only if $\hat{v}z \in E(G)$. Now consider the vertex w with $f(w) \in \{i, j\} \setminus f(\hat{v})$ and $\ell(w) < r(\hat{v}) < r(w)$. Vertex w becomes the new active interval. If $\hat{v}w \in E(G)$, then the active label stays the same, otherwise the new active label becomes the other one. In any case w gets the new active label. Note that $L(\hat{v}) \cap L(w) \neq \emptyset$ if and only if $\hat{v}w \in E(G)$. We repeat this procedure until the end of the interval representation. Furthermore, we repeat the whole procedure for all sets $\{i, j\} \subseteq \{1, \ldots, k\}$. In the end, we obtain a d-simultaneous interval representation (R, L) of G where $d = 2\binom{k}{2} = k(k-1) = \mathsf{pw}(G)^2 + \mathsf{pw}(G)$. ◀

Observe at this point that bounded simultaneous interval number does not imply bounded pathwidth as is proven by the class of interval graphs.

An *edge clique cover* of a graph G is a set $\mathcal{K}$ of cliques of G such that every edge of G is contained in some clique of $\mathcal{K}$. We denote by $\mathsf{ecc}(G)$ the *edge clique cover number* of G, that is, the minimum size of an edge clique cover of G.

▶ **Lemma 3.10.** *Let $\mathcal{C}$ be a class of intersection graphs. Let G be a graph, let $d \geq 0$ be an integer, and let R be a $\mathcal{C}$-representation of some graph $F \in \mathcal{C}$. Then, there exists a d-simultaneous $\mathcal{C}$-representation (R, L) of G if and only if there exists a graph H with $\mathsf{ecc}(H) \leq d$ and G is the intersection of F and H.*

Lemma 3.10 implies the following.

▶ **Corollary 3.11.** *Let $\mathcal{C}$ be a class of intersection graphs, let G be a graph, and let $d \geq 0$ be an integer. Then, G has an d-simultaneous $\mathcal{C}$-representation if and only if G is the intersection of a graph in $\mathcal{C}$ and a graph with edge clique cover number at most d.*

Lemma 3.10 also implies the following strengthening of Theorem 2.5.

▶ **Theorem 3.12.** *For every class of intersection graphs $\mathcal{C}$, every graph G has an $\mathsf{ecc}(G)$-simultaneous $\mathcal{C}$-representation.*

▶ **Corollary 3.13.** *Every graph G satisfies $\mathsf{si}(G) \leq \mathsf{ecc}(G)$.*

Interval graphs, and in particular paths, have unbounded edge clique cover number. Thus, bounded simultaneous interval number does not imply bounded edge clique cover number.

The bound given by Corollary 3.13 is tight. Let us denote by K_n^p the complete multipartite graph on p partite sets of the same size n and by $\lambda(n)$ the largest size of a family of mutually orthogonal Latin squares of order n. It is known that $\lambda(n) \leq n-1$ and that equality holds if and only if there exists a projective plane of order n. Thus $\lambda(q) = q-1$ if q is a prime power, but in general the exact computation of the value of $\lambda(n)$ is difficult. Park, Kim, and Sano showed in [55] that for any two integers p and n such that $3 \leq p \leq \lambda(n)+2$, the edge clique cover number of K_n^p equals n^2. Taking $p=3$, we thus obtain, by combining Theorem 2.8 and Corollary 3.13, that for the complete 3-partite graph G with parts of equal size, we have $\mathsf{si}(G) = \mathsf{ecc}(G) = \frac{|V(G)|^2}{9}$.

4 Complexity of Computing the Simultaneous Interval Number

Using the characterization from Definition 2.2, we can state three natural recognition problems for d-simultaneous $\mathcal{C}$-representations.

▶ **Problem 1** (Simultaneous $\mathcal{C}$-Representation Problem).
Input: *A graph G and a labeling $L : V(G) \to \mathcal{P}(\{1,\dots,d\})$ of G.*
Question: *Does there exist a d-simultaneous $\mathcal{C}$-representation (R, L) of G?*

By Observations 2.3 and 2.4, Problem 1 is a generalization of the simultaneous representation problems by Jampani and Lubiw [41].

In the second problem we are given the graph and some representation and want to find a suitable labeling.

▶ **Problem 2** (Simultaneous Labeling Problem Given a $\mathcal{C}$-Representation).
Input: *A graph G and a $\mathcal{C}$-representation R of a graph F with $V(G) = V(F)$ and $E(G) \subseteq E(F)$.*
Question: *What is the smallest number $d \in \mathbb{N}$ such that there exists a d-simultaneous $\mathcal{C}$-representation (R, L) of G?*

In the third version, we are given just a graph and wish to compute the smallest number of labels needed for the graph to have a d-simultaneous $\mathcal{C}$-representation.

▶ **Problem 3** (Generalized Simultaneous $\mathcal{C}$-Representation Problem).
Input: *A graph G.*
Question: *What is the smallest number $d \in \mathbb{N}$ such that there exists a d-simultaneous $\mathcal{C}$-representation of G?*

Recall the definition of a class of intersection graphs given on p. 4.

▶ **Theorem 4.1.** *The Simultaneous Labeling Problem Given a $\mathcal{C}$-Representation is* NP*-hard for any class of intersection graphs $\mathcal{C}$, even if all sets in the given $\mathcal{C}$-representation pairwise intersect.*

▶ **Theorem 4.2.** *The Generalized Simultaneous $\mathcal{C}$-Representation Problem is* NP*-hard for every class of intersection graphs that is a subclass of the class of C_4-free graphs and contains the class of complete split graphs.*

▶ **Corollary 4.3.** *Let $\mathcal{C}$ be the class of interval graphs or the class of chordal graphs. Then, the Generalized Simultaneous $\mathcal{C}$-Representation Problem is* NP*-hard.*

▶ **Corollary 4.4.** *It is* NP*-hard to compute the simultaneous interval number of a graph G.*

5 Cliques

In this section, we focus on the *Maximum Clique* problem: Given a graph $G = (V, E)$, compute a largest clique in G. The problem can be naturally generalized to the weighted case, where the input graph is equipped with a vertex weight function $w : V \to \mathbb{Q}_+$ and the task is to find a clique C in G maximizing its weight, $w(C)$, defined as the sum of the weights of the vertices in C.

▶ **Theorem 5.1.** *A graph G has at most $2^{2^{\mathsf{si}(G)}} \cdot n$ many maximal cliques.*

Proof. Let $d = \mathsf{si}(G)$ and fix a d-simultaneous interval representation (R, L) of G. Let C be a maximal clique of G. There exists a point p on the real line that is contained in any interval of the vertices contained in C. Furthermore, for every subset $S \subseteq \{1, \dots, d\}$, if there is any vertex $u \in C$ such that $L(u) = S$, then the clique C contains all the vertices v whose label set is exactly S and whose interval $R(v)$ contains p. There are at most n points on the real line such that the sets of intervals containing these points are pairwise incomparable with respect to inclusion. These are always points before the endpoint of some interval. For each of those points we have to decide for every subset of $\{1, \dots, d\}$ if vertices having this subset as label set are contained in the maximal cliques. There are 2^d many subsets. Therefore, there are 2^{2^d} different decisions and, thus, there are at most $2^{2^d} n$ many maximal cliques. ◀

▶ **Theorem 5.2.** *Given a graph G with n vertices and a d-simultaneous interval representation of G, the maximal cliques of G can be enumerated in time $\mathcal{O}(d \cdot 2^{2^d+2d} \cdot n \log n)$.*

This result implies directly that we can compute a maximum-weight clique of a graph G within the same time bound.

▶ **Corollary 5.3.** *Given a vertex-weighted graph G with n vertices and a d-simultaneous interval representation of G, we can find a maximum weight clique of G in time $\mathcal{O}(d \cdot 2^{2^d+2d} \cdot n \log n)$.*

Tsukiyama et al. [59] gave an algorithm that generates all maximal cliques in time $\mathcal{O}(n^3\mu)$ where μ is the number of maximal cliques. Using this algorithm, we can drop the requirement in Theorem 5.2 and Corollary 5.3 that the input graph is given together with a d-simultaneous interval representation.

▶ **Theorem 5.4.** *Given a vertex-weighted graph G with n vertices, we can find a list of all maximal cliques and a maximum weight clique of G in time $\mathcal{O}(2^{2^{\mathsf{si}(G)}} \cdot n^3)$.*

Let us remark that a faster dependency on n (although still slower than quadratic in n) could be obtained by using some of the more recent maximal clique enumeration algorithms (see, e.g., [15]).

Note that the unweighted maximum clique problem is already NP-hard for 2-unit interval graphs and 3-track interval graphs [26] while it is polynomial-time solvable for 2-track interval graphs. However, there is an FPT algorithm for the clique problem on d-interval graphs when parameterized by d plus solution size [24].

Next we prove that the bound given in Theorem 5.1 is tight. To this end, we consider complements of matchings. As we have seen in Lemma 2.10, the simultaneous interval number of those graphs is at least $\log_2(n-1)$ where n is the number of vertices. Here, we show that this bound is tight. Let M_m be the complement of a matching with m edges. Gregory and Pullman [33] showed that $\lim_{m\to\infty} \frac{\mathsf{ecc}(M_m)}{\log_2(m)} = 1$. As we have seen in Corollary 3.13, it holds that $\mathsf{si}(G) \leq \mathsf{ecc}(G)$. This implies the following result.

▶ **Lemma 5.5.** *For any $\varepsilon > 0$, there exists some $n' \in \mathbb{N}$ such that for all even $n \geq n'$, the following holds: If G is the complement of a matching with n vertices, then $\mathsf{si}(G) \leq (1+\varepsilon)\log_2 n$.*

Using this result, we are able to prove that the bound given in Theorem 5.1 is tight.

▶ **Theorem 5.6.** *For any ε with $0 < \varepsilon < 1$ and any $k \in \mathbb{N}$, there is an infinite family $\mathcal{F}$ of graphs such that any graph $G \in \mathcal{F}$ with n vertices has at least $2^{2^{(1-\varepsilon)\mathsf{si}(G)}} \cdot n^k$ many maximal cliques.*

This result shows that the bound on the running time of our approach for the Maximum Clique problem cannot be significantly improved. Furthermore, the following result shows that the Maximum Clique problem cannot be solved with a single-exponential FPT algorithm parameterized by the simultaneous interval number.

▶ **Theorem 5.7.** *Unless $\mathsf{P} = \mathsf{NP}$, for any fixed $k \in \mathbb{N}$ there is no algorithm that solves the Maximum Clique problem on complements of cubic graphs with n vertices in time $2^{\mathcal{O}(\mathsf{si})}n^k$.*

Note that the above result does not rule out the possibility that it may be possible to solve the Maximum Clique problem in time $2^{\mathcal{O}(d)}n^k$ when a d-simultaneous interval representation of the graph is given.

As graphs with bounded simultaneous interval number are (pw, ω)-bounded (Theorem 3.8), we can use the results from Chaplick et al. [14, Theorem 11] to show that the clique problem admits an FPT algorithm when parameterized by the simultaneous interval number plus solution size.

▶ **Theorem 5.8.** *Given an n-vertex graph G and an integer k, it can be determined in time $2^{\mathcal{O}(\mathsf{si}(G)k)}n$ whether G contains a clique of size k.*

6 Coloring

Circular-arc graphs have linear mim-width at most 2 [2, Lemma 4], path-independence number at most 2 [53, proof of Theorem 4.5] and track number at most 2. Since the Coloring problem is NP-hard on circular-arc graphs [28], the same holds for graphs whose linear mim-width, path-independence number, and track number are at most 2. This result does not transfer directly to the simultaneous interval number, as the simultaneous interval number of complements of matchings and, thus, of circular-arc graphs is unbounded, due to Lemma 2.10. Nevertheless, we can adapt a proof for the NP-hardness of the Coloring problem on circular-arc graphs given by Marx [52] to the case of graphs of simultaneous interval number 2. This proof uses the following definitions and results.

▶ **Problem 4** (Disjoint Paths).

Input: *Directed graphs G and H on the same vertex set.*

Question: *Are there paths P_e in G for each $e \in E(H)$ such that these paths are edge disjoint and path P_e together with edge e forms a directed cycle?*

Given a directed graph $G = (V, E)$, the *in-degree* (resp. *out-degree*) of a vertex $v \in V$ in G is the number of directed edges $(x, y) \in E$ such that $v = y$ (resp. $v = x$), and the *degree* $d_G(v)$ of v in G is the number of directed edges $(x, y) \in E$ such that $v \in \{x, y\}$. A directed graph $G = (V, E)$ is *Eulerian* if for each vertex $v \in V$, the in-degree of v equals its out-degree.

▶ **Theorem 6.1** (Vygen [61]). *The Disjoint Paths problem remains* NP*-complete even if G is acyclic and $G + H$ is Eulerian.*

▶ **Lemma 6.2** (Marx [52]). *If $G + H$ is Eulerian and G is acyclic, then every solution of the Disjoint Path problem given G and H uses every edge of G.*

▶ **Lemma 6.3.** *The Disjoint Paths problem remains* NP*-complete even if G is acyclic, $G + H$ is Eulerian, and every vertex in H has degree at most one.*

▶ **Theorem 6.4.** *The Coloring problem is* NP*-complete on graphs G with* $\mathsf{si}(G) \leq 2$ *even if a* 2*-simultaneous interval representation of G is given.*

Proof. We adapt a proof given by Marx [52] to establish NP-completeness of the Coloring problem on circular-arc graphs. Let (G, H) be an instance of the Disjoint Paths problem such that G is acyclic, $G + H$ is Eulerian, and $d_H(v) \leq 1$ for all $v \in V(G)$. Let $k = |E(H)|$.

Let $v_1, \dots, v_n$ be a topological sort of G. For every edge $(v_i, v_j) \in E(G)$ we construct an interval (i, j) with label set $\{1\}$. Note that $i < j$, due to the property of the topological sort. For every edge $(v_i, v_j) \in E(H)$ we may assume that $i > j$ since otherwise there is no path from v_j to v_i in G. We add the intervals $(0, j)$ and $(i, n + 1)$ with label set $\{1, 2\}$ and the interval (j, i) with label set $\{2\}$. We call the resulting 2-simultaneous interval graph G'.

We claim that (G, H) is a yes instance of the disjoint path problem if and only if G' can be colored with k colors. First assume that (G, H) is a yes instance. Fix a solution, that is, paths P_e in G for each $e \in E(H)$ such that these paths are edge-disjoint and path P_e together with edge e forms a directed cycle. By Lemma 6.2, the solution covers all the edges with k directed cycles. Let C be the ℓ-th cycle in the solution. For every edge $(v_i, v_j) \in E(C) \cap E(G)$ we color the corresponding interval (i, j) that has label $\{1\}$ with color ℓ. For the edge $(v_i, v_j) \in E(C) \cap E(H)$ we color with color ℓ the intervals $(0, j)$ and $(i, n+1)$ that have label set $\{1, 2\}$ as well as the interval (j, i) that has label set $\{2\}$. This leads to a proper coloring of G' since the only intervals with the same color that intersect each other do not share a label and, thus, their corresponding vertices are not adjacent.

Now assume the graph G' can be properly colored with k colors. As all the k intervals with label set $\{1, 2\}$ that start in 0 pairwise intersect, they have different colors. Now consider the subgraph of G induced by the intervals containing label 2. Since every vertex in H has degree at most one, whenever an interval ends before point $n + 1$, there is no other interval that ends at this point. Furthermore, there is exactly one interval that starts at this point. This implies that every point p in the interval $(0, n + 1)$ in which no interval ends belongs to exactly k intervals. Consequently, any two intervals such that the second one starts where the first one ends must have the same color. This implies, in particular, that the two intervals with label set $\{1, 2\}$ representing the same edge of H have the same color.

Now consider all the intervals that contain the label 1. There are k of those intervals that start in point 0. If exactly j of those intervals end in point i, then there are exactly j intervals that start in i, due to the Eulerian property of $G + H$. Thus, any non-integer point in $(0, n + 1)$ is contained in exactly k intervals. This also implies that for any of those points there is an interval with color $d \in \{1, \dots, k\}$. Therefore, the intervals with color d represent a directed cycle in $G + H$ containing exactly one edge of H.Thus, (G, H) is a yes instance of the disjoint path problem. ◀

As any class of graphs with bounded simultaneous interval number are (pw, ω)-bounded (Theorem 3.8), we can use the results from Chaplick et al. [14, Theorem 12] to show that the List k-Coloring problem admits an FPT algorithm when parameterized by k plus the simultaneous interval number.

▶ **Theorem 6.5.** *Given a graph G, we can solve the List k-Coloring problem on G in time* $k^{\mathcal{O}(\mathsf{si}(G)k)}n$.

7 Domination and Independent Sets

The Dominating Set problem and the Independent Set problem can be solved in polynomial time on interval graphs [29, 57]. However, when we parameterize these problems by the solution size and linear mim-width they are W[1]-hard [39]. If we parameterize the Dominating Set problem by tree-α and the solution size then it is W[2]-hard [49]. In contrast, when the problems are parameterized by simultaneous interval number and the solution size, then bounded-search-tree methods lead to FPT-algorithms.

▶ **Theorem 7.1.** *Given a graph G with n vertices and a d-simultaneous interval representation of G, we can decide whether G has a dominating set of size at most k or an independent set of size k in time $\mathcal{O}(2^{kd} \cdot n)$.*

Using a technique due to Fomin et al. [25], in [39] Jaffke et al. showed that a whole range of domination-type problems (including dominating and independent set) are W[1]-hard when parameterized by mim-width and solution size. While that approach cannot be easily adapted for the simultaneous interval number, it is possible to show that at least one of these problems is W[1]-hard when parameterized just by si.

▶ **Problem 5.** *Independent Dominating Set Problem (IDSP)*

Instance: *A graph G and an integer k.*

Question: *Does there exist a set X of at most k vertices of G such that $G[X]$ is edgeless and $N_G[X] = V(G)$?*

The results in [25] use a reduction from the *Multicolored Clique problem* (MCP), a technique popularized by Fellows et al. [24]. We will use a reduction from the *Multicolored Independent Set problem.*

▶ **Problem 6.** *Multicolored Independent Set Problem (MISP)*

Instance: *A graph G with a proper coloring of k colors.*

Question: *Is there an independent set I in G such that I contains exactly one vertex of each color?*

The MCP (and thus the MISP) was shown to be W[1]-hard when parameterized by solution size by Pietrzak [56] and by Fellows et al. [24]. In fact, in [17, 50] the authors show the following result under the assumption of the Exponential Time Hypothesis (ETH) which asserts that solving n-variable 3-SAT requires time $2^{\Omega(n)}$ (see [38]).

▶ **Theorem 7.2** (Cygan et al. [17], Lokshtanov et al. [50])**.** *Assuming the Exponential Time Hypothesis, there is no $f(k)n^{o(k)}$ time algorithm for the MCP (MISP) for any computable function f.*

For an instance G of the MCP we can assume that all color classes are of the same size q, since adding isolated vertices does not affect the existence or nonexistence of a multicolored clique. A similar assumption can be made for the MISP. For each color class $i \in \{1, \dots, k\}$, we denote the vertices in the class by $v_1^i, \dots, v_q^i$.

We will show that the IDSP is W[1]-hard when parameterized by the simultaneous interval number. To this end we will construct a reduction from the MISP in the following way. Let G together with a vertex partition $V(G) = V_1 \dot\cup \dots \dot\cup V_k$ be an instance of the MISP, where $V_i = \{v_1^i, \dots, v_q^i\}$ for all $i \in \{1, \dots, k\}$.

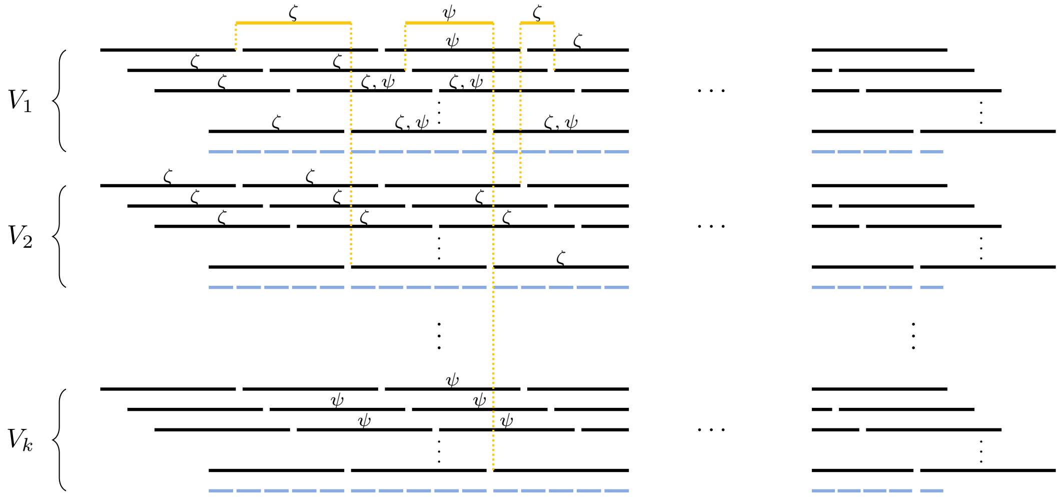


Figure 4 The yellow intervals represent the edges of G, the black intervals are the intervals of the W^i_j. The blue intervals are in S_i. Each of the rows marked V_i represent that vertex set of G. For visual reasons the intervals belonging to the V_i have not been shifted by ϵ as in the definition. For the same reason, we define $\zeta := k+1$ and $\psi := k+2$. The labels of the edge intervals are denoted completely above these. Each of the other intervals also contains the label i if it is associated with V_i. The intervals on the right have not been labeled.

Let $E(G) = \{e_1, \dots, e_m\}$. We will now define a $(k+2)$-simultaneous interval graph G' together with its $(k+2)$-simultaneous interval representation. For each vertex $v^i_j \in V(G)$ we will define a collection of $m+1$ (open) intervals (see Figure 4)

$$W^i_j := \left\{ \left(\gamma - 1 + (j-1)\frac{1}{q} + i\epsilon, \gamma + (j-1)\frac{1}{q} + i\epsilon \right) : \gamma \in \{1, \dots, m+1\} \right\},$$

where $\epsilon \ll \frac{1}{kq}$, i.e., all k shifts in sum are much smaller than one interval of a W^i_j.

We will denote the γ-th interval of v^i_j as $I^i_j(\gamma)$. These intervals will be referred to as the *vertex intervals*. Note that none of these intervals have common left endpoints or common right endpoints. Furthermore, for each of the V_i we add an additional collection of $2mq+2$ intervals

$$S_i := \left\{ \left(\frac{q-1}{q} + \gamma\frac{1}{q} + i\epsilon, 1 + \gamma\frac{1}{q} + i\epsilon \right) : \gamma \in \{0, \dots, 2mq+1\} \right\},$$

where again $\epsilon \ll \frac{1}{qk}$.

Finally, we add further intervals for each edge in G. Let $e_\gamma = v^i_a v^j_b$ be an edge with $v^i_a \in V_i$, $v^j_b \in V_j$. W.l.o.g. we may assume that $a \leq b$ and if $a = b$, then $i < j$. We add an interval of the form $I(e_\gamma) = (r(I^i_a(\gamma)), \ell(I^j_b(\gamma+1)))$. These intervals will be referred to as the *edge intervals*. As none of the intervals of different vertices have common endpoints, we can be sure that each of these edge intervals has strictly positive length. In the following, we will frequently identify the intervals and vertices of G' in order to simplify the notation.

In the next step, we assign a label set to each of the intervals in order to construct a simultaneous interval graph. To each interval in S_i we assign the label set $\{i\}$ and to each $I(e_\gamma)$ we assign the label set $\{k+1\}$ if γ is odd and $\{k+2\}$ if γ is even.

Before we label the vertex intervals, we need the following observation which follows easily from the definitions above.

▶ **Observation 7.3.** *Any interval $I_j^i(\gamma)$ intersects at most two edge intervals and these intervals have distinct labels.*

Any interval of a W_j^i is given at least the label i. Let $I_j^i(\gamma)$ be one of the intervals representing the vertices of G. If $I_j^i(\gamma)$ does not intersect any edge interval such that one of the endpoints of that edge is contained in V_i, then $L(I_j^i(\gamma)) = \{i\}$. If $I_j^i(\gamma)$ intersects some edge interval such that one of that edges endpoints is contained in V_i but is not identical to v_j^i, then we add the label of that edge to $L(I_j^i(\gamma))$. If $I_j^i(\gamma)$ intersects some edge interval such that one of that edges' endpoints is identical to v_j^i, then $L(I_j^i(\gamma))$ does not contain the label of that edge. Note that these last two rules cannot lead to a contradiction, due to Observation 7.3. Therefore, any interval $I_j^i(\gamma)$ has label set either $\{i\}$, $\{i, k+1\}$, $\{i, k+2\}$ or $\{i, k+1, k+2\}$.

▶ **Lemma 7.4.** *Any minimum independent dominating set of G' must contain all the vertices corresponding to the intervals in the set $W_{j_1}^1 \cup \ldots \cup W_{j_k}^k$ for some set of indices $\{j_1, \ldots, j_k\}$.*

▶ **Lemma 7.5.** *The vertices belonging to $W := W_{j_1}^1 \cup \ldots \cup W_{j_k}^k$ form an independent dominating set of G' if and only if $C := \{v_{j_1}^1, \ldots, v_{j_k}^k\}$ is a multicolored independent set of G.*

Combining Lemmas 7.4 and 7.5 with the fact that MISP is $\mathsf{W}[1]$-hard when parameterized by the solution size and Theorem 7.2 we get the following result.

▶ **Theorem 7.6.** *The IDSP is $\mathsf{W}[1]$-hard when parameterized by the simultaneous interval number even if a $\mathsf{si}(G)$-simultaneous interval representation is given. Furthermore, assuming the ETH, there is no $f(\mathsf{si})n^{o(\mathsf{si})}$-time algorithm for the ISDP for any computable function f.*

Note that this reduction cannot be easily adapted to show that independent dominating set is $\mathsf{W}[1]$-hard when parameterized by the simultaneous interval number *and* the solution size k, as the minimum size of an independent dominating set in G' is of the order $\Omega(km)$.

8 Conclusion

While we have presented some algorithmic properties for graphs of bounded simultaneous interval number, many open problems still remain. First and foremost is the computation of si. Unsurprisingly, the computation of si is NP-hard, however, we are not aware of any results regarding the decision problem whether si is at most some fixed value d. It still remains to be seen whether there exists a computable function f and an FPT or an XP algorithm that for a given graph G and integer d, either correctly determines that $\mathsf{si}(G) > d$ or computes an $f(d)$-simultaneous interval representation of G. Such FPT algorithms are known for treewidth [5, 48] and cliquewidth [54], and XP algorithms are known for the tree-independence number [20, 62].

Furthermore, the complexity status of many important problems is still open when parameterized by si, for example that of independent set and dominating set (see Table 1). Regarding the obtained FPT results, it remains to be shown whether the running times are best possible. Especially in the case of the clique problem, there is still a large discrepancy between the achieved running time and the lower bound.

The simultaneous representation problem has also been considered for chordal graphs [41]. This imposes the question whether similar results can be made for a *simultaneous chordal number*. In fact, some of the results given here for the simultaneous interval number can be directly translated for the simultaneous chordal number as well. However, as the Dominating Set problem is $\mathsf{W}[2]$-hard for chordal graphs, the FPT algorithm for that problem given in Theorem 7.1 does not carry over.

References

1 Jesse Beisegel, Nina Chiarelli, Ekkehard Köhler, Martin Milanič, Peter Muršič, and Robert Scheffler. The simultaneous interval number: A new width parameter that measures the similarity to interval graphs, 2024. arXiv:2404.10670.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

3 Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Information Processing Letters*, 19(1):37–40, 1984. doi:10.1016/0020-0190(84)90126-1.

4 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Transactions on Algorithms*, 12(2):46, 2016. Id/No 16. doi:10.1145/2738054.

5 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

6 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

7 Jan Bok and Nikola Jedličková. A note on simultaneous representation problem for interval and circular-arc graphs, 2018. arXiv:1811.04062.

8 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width. I: Tractable FO model checking. *Journal of the ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

9 Flavia Bonomo and Diego de Estrada. On the thinness and proper thinness of a graph. *Discrete Applied Mathematics*, 261:78–92, 2019. doi:10.1016/j.dam.2018.03.072.

10 Flavia Bonomo-Braberman and Gastón Abel Brito. Intersection models and forbidden pattern characterizations for 2-thin and proper 2-thin graphs. *Discrete Applied Mathematics*, 339:53–77, 2023. doi:10.1016/j.dam.2023.06.013.

11 Flavia Bonomo-Braberman, Carolina L. Gonzalez, Fabiano S. Oliveira, Moysés S. Sampaio Jr., and Jayme L. Szwarcfiter. Thinness of product graphs. *Discrete Applied Mathematics*, 312:52–71, 2022. doi:10.1016/j.dam.2021.04.003.

12 Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM Journal on Computing*, 11:191–199, 1982. doi:10.1137/0211015.

13 Sunil Chandran, Carlo Mannino, and Gianpaolo Oriolo. The indepedent set problem and the thinness of a graph. Unpublished manuscript cited in [11], 2007.

14 Steven Chaplick, Martin Töpfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. *Algorithmica*, 83(11):3281–3318, 2021. doi:10.1007/s00453-021-00846-3.

15 Carlo Comin and Romeo Rizzi. An improved upper bound on maximal clique listing via rectangular fast matrix multiplication. *Algorithmica*, 80(12):3525–3562, 2018. doi:10.1007/s00453-017-0402-5.

16 Derek G. Corneil and Yehoshua Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27–39, 1984. doi:10.1016/0166-218X(84)90088-X.

17 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Cham: Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. I: Graph classes with a forbidden structure. *SIAM Journal on Discrete Mathematics*, 35(4):2618–2646, 2021. doi:10.1137/20M1352119.

19 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. II: Tree-independence number. *Journal of Combinatorial Theory. Series B*, 164:404–442, 2024. doi:10.1016/j.jctb.2023.10.006.

20 Clément Dallard, Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, and Martin Milanič. Computing tree decompositions with small independence number. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51th International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. To appear.

21 H. N. de Ridder et al. Entry "2-interval" in Information System on Graph Classes and their Inclusions (ISGCI). URL: https://graphclasses.org/classes/gc_40.html.

22 H. N. de Ridder et al. Entry "3-track" in Information System on Graph Classes and their Inclusions (ISGCI). URL: https://graphclasses.org/classes/gc_1080.html.

23 H. N. de Ridder et al. Entry "$3K_1$-free" in Information System on Graph Classes and their Inclusions (ISGCI). URL: https://graphclasses.org/classes/AUTO_399.html.

24 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

25 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.

26 Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71:812–836, 2015. doi:10.1007/s00453-013-9828-6.

27 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965. doi:10.2140/pjm.1965.15.835.

28 Michael R. Garey, David S. Johnson, Gerald L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980. doi:10.1137/0601025.

29 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

30 Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve H. Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear MIM-width. *Algorithmica*, 80(2):714–741, 2018. doi:10.1007/s00453-017-0289-1.

31 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science*, 11(3):423–443, 2000. doi:10.1142/S0129054100000260.

32 Carolina Lucía Gonzalez and Felix Mann. On d-stable locally checkable problems parameterized by mim-width. *Discrete Applied Mathematics*, 347:1–22, 2024. doi:10.1016/j.dam.2023.12.015.

33 David A. Gregory and Norman J. Pullman. On a clique covering problem of Orlin. *Discrete Mathematics*, 41(1):97–99, 1982. doi:10.1016/0012-365X(82)90085-1.

34 Jerrold R. Griggs and Douglas B. West. Extremal values of the interval number of a graph. *SIAM Journal on Algebraic Discrete Methods*, 1(1):1–7, 1980. doi:10.1137/0601001.

35 Udaiprakash I. Gupta, Der-Tsai Lee, and Joseph Y.-T. Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, C-28(11):807–810, 1979. doi:10.1109/TC.1979.1675260.

36 András Gyárfás and Douglas B. West. Multitrack interval graphs. In *Proceedings of the Twenty-sixth Southeastern International Conference on Combinatorics, Graph Theory and Computing (Boca Raton, FL, 1995)*, volume 109 of *Congressus Numerantium*, pages 109–116, 1995.

37 Akihiro Hashimoto and James Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th Design Automation Workshop*, DAC '71, pages 155–169, New York, NY, USA, 1971. Association for Computing Machinery. doi:10.1145/800158.805069.

38 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. Special issue on the Fourteenth Annual IEEE Conference on Computational Complexity (Atlanta, GA, 1999). doi:10.1006/jcss.2000.1727.

39 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width. III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019. doi:10.1016/j.tcs.2019.09.012.

40 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width. I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, 2020. doi:10.1016/j.dam.2019.06.026.

41 Krishnam Raju Jampani and Anna Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. In Frank K. H. A. Dehne andMarina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 2009. doi:10.1007/978-3-642-03367-4_34.

42 Krishnam Raju Jampani and Anna Lubiw. Simultaneous interval graphs. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 206–217. Springer, 2010. doi:10.1007/978-3-642-17517-6_20.

43 Krishnam Raju Jampani and Anna Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. *Journal of Graph Algorithms and Applications*, 16(2):283–315, 2012. doi:10.7155/jgaa.00259.

44 Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theoretical Computer Science*, 411(49):4253–4262, 2010. doi:10.1016/j.tcs.2010.09.001.

45 Minghui Jiang. Recognizing d-interval graphs and d-track interval graphs. *Algorithmica*, 66(3):541–563, 2013. doi:10.1007/s00453-012-9651-5.

46 J. Mark Keil. Finding Hamiltonian circuits in interval graphs. *Information Processing Letters*, 20:201–206, 1985. doi:10.1016/0020-0190(85)90050-X.

47 Brian W. Kernighan, Daniel G. Schweikert, and G. Persky. An optimum channel-routing algorithm for polycell layouts of integrated circuits. In J. Michael Galey, Herbert M. Wall, Robert B. Hitchcock Sr., Ben E. Britt, Richard E. Merwin, Donald J. Humcke, and David B. Smithhisler, editors, *Proceedings of the 10th Design Automation Workshop, DAC '73, Portland, Oregon, USA, June 25-27, 1973*, pages 50–59. ACM, 1973. doi:10.1145/62882.62886.

48 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science—FOCS 2021*, pages 184–192. IEEE Computer Soc., Los Alamitos, CA, 2022. doi:10.1109/FOCS52979.2021.00026.

49 Chunmei Liu and Yinglei Song. Parameterized complexity and inapproximability of dominating set problem in chordal and near chordal graphs. *Journal of combinatorial optimization*, 22(4):684–698, 2011. doi:10.1007/s10878-010-9317-7.

50 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 105:41–71, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

51 Carlo Mannino, Gianpaolo Oriolo, Federico Ricci, and Sunil Chandran. The stable set problem and the thinness of a graph. *Operations Research Letters*, 35(1):1–9, 2007. doi:10.1016/j.orl.2006.01.009.

52 Dániel Marx. A short proof of the NP-completeness of circular arc coloring. Unpublished manuscript, 2003. URL: https://www.cs.bme.hu/~dmarx/papers/circularNP.pdf.

53 Martin Milanič and Paweł Rzążewski. Tree decompositions with bounded independence number: beyond independent sets, 2022. arXiv:2209.12315.

54 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory. Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

55 Boram Park, Suh-Ryung Kim, and Yoshio Sano. The competition numbers of complete multipartite graphs and mutually orthogonal Latin squares. *Discrete Mathematics*, 309(23-24):6464–6469, 2009. doi:10.1016/j.disc.2009.06.016.

56 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

57 Ganesan Ramalingam and C. Pandu Rangan. A unified approach to domination problems on interval graphs. *Information Processing Letters*, 27(5):271–274, 1988. doi:10.1016/0020-0190(88)90091-9.

58 Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer. Simultaneous representation of proper and unit interval graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 80:1–80:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ESA.2019.80.

59 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.

60 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012. URL: https://hdl.handle.net/1956/6166.

61 Jens Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90, 1995. doi:10.1016/0166-218X(93)E0177-Z.

62 Nikola Yolov. Minor-matching hypertree width. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 219–233. SIAM, 2018. doi:10.1137/1.9781611975031.16.

Correlation Clustering with Vertex Splitting

Matthias Bentert
University of Bergen, Norway

Alex Crane
University of Utah, Salt Lake City, UT, USA

Pål Grønås Drange
University of Bergen, Norway

Felix Reidl
Birkbeck, University of London, UK

Blair D. Sullivan
University of Utah, Salt Lake City, UT, USA

Abstract

We explore Cluster Editing and its generalization Correlation Clustering with a new operation called *permissive vertex splitting* which addresses finding overlapping clusters in the face of uncertain information. We determine that both problems are NP-hard, yet they exhibit significant differences in terms of parameterized complexity and approximability. For Cluster Editing with Permissive Vertex Splitting, we show a polynomial kernel when parameterized by the solution size and develop a polynomial-time 7-approximation. In the case of Correlation Clustering, we establish para-NP-hardness when parameterized by the solution size and demonstrate that computing an $n^{1-\varepsilon}$-approximation is NP-hard for any constant $\varepsilon > 0$. Additionally, we extend an established link between Correlation Clustering and Multicut to the setting with permissive vertex splits.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Approximation algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases graph modification, cluster editing, overlapping clustering, approximation, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.8

Related Version *Full Version*: https://arxiv.org/abs/2402.10335

Funding *Alex Crane*, *Felix Reidl*, and *Blair D. Sullivan*: Gordon & Betty Moore Foundation's Data Driven Discovery Initiative under award GBMF4560 to Blair D. Sullivan.
Matthias Bentert: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

1 Introduction

Discovering clusters, or communities, is a core task in understanding the vast amounts of relational data available. One limitation of many traditional clustering algorithms is the necessity of specifying a desired number of clusters as part of the input. The problem known as Cluster Editing avoids this by instead aiming to minimize the number of edge insertions and removals necessary to transform the input into a *cluster graph* (a disjoint union of cliques). This problem has been heavily studied in the graph-algorithms community, and was first proved to be fixed-parameter tractable with respect to the number of edge modifications (k) by Cai in 1996 [12]. The running time has significantly improved since, with the best known algorithm running in $O(1.62^k(n+m))$ time [11]. The problem also admits a polynomial kernel with $2k$ vertices [16].

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 8; pp. 8:1–8:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Formally in Cluster Editing, we consider a complete graph where each edge is labeled as positive (which we imagine as colored *blue*) or negative (colored *red*) and we ask for the minimum number of edges whose color must be changed so that there is a partition of the vertex set where all edges within each part are blue, and all edges between parts are red. This convention of an edge-labeled complete graph will be useful in our setting and easily maps onto the more common formalism for Cluster Editing with an incomplete, uncolored graph as input (imagine the graph edges as blue and its non-edges as red). We also note that other conventions for labelling positive/negative edges exist in the literature, e.g. using labels like $\langle + \rangle$ and $\langle - \rangle$.

In practice, the positive or negative association between objects is usually computed using a similarity metric which we can think of as an oracle function which, given two objects, computes a score that expresses the (dis)similarity of the inputs. For large-scale data, the assumption of complete information is then unrealistic for two reasons: First, the quadratic complexity of computing all pairwise associations is prohibitively expensive. Second, the similarity oracle may be unable to provide a clear answer for certain pairs – suggesting that objects can either be grouped together or kept separate, depending on other parts of the data or even external domain context.

Consequently, we should also consider cases in which the input is an incomplete graph with positive and negative labels on the existing edges and no information about pairs not joined by an edge. This scenario has previously been investigated by Demaine et al. [20] by allowing “0-weight edges” (zero-edges) in their cluster-editing framework[1]. For clarity, we will refer to the problem where zero-edges (non-edges) are allowed as Correlation Clustering and to the problem where the input graph is *complete* – i.e. every vertex pair is connected either by a blue or a red edge – as Cluster Editing.

The approximability of both Cluster Editing and Correlation Clustering are well-studied. First considered by Bansal, Blum, and Chawla [8], under the name *correlation clustering*[2], Cluster Editing admits a 1.73-approximation [17] when minimizing the number of disagreements (red edges within and blue edges between clusters). Other variants of Cluster Editing which maximize the number of agreements or the correlation (agreements minus disagreements) admit a PTAS (polynomial-time approximation scheme) and a $\Omega(\log n)$-approximation, respectively [8, 14]. In the more general setting of minimizing disagreements for Correlation Clustering (i.e., when zero-edges are present but never constitute a disagreement), an $O(\log n)$-approximation is known [20]. This result arises from the strong relation between Correlation Clustering and Multicut[3]. The connection was first observed with Multiway Cut by Bansal, Blum, and Chawla [8], before an approximation-preserving reduction from Multicut to Correlation Clustering was given independently by both Charikar, Guruswami, and Wirth [13] and Demaine et al. [20]. The connection to Multicut also implies that no constant-factor approximation is possible for Correlation Clustering, unless the Unique Games Conjecture is false [15].

These algorithmic advances provide a positive outlook on applying these clustering variants in practice, however, we need to also investigate whether the proposed clustering model could be improved. In particular, we need to question the underlying assumption that real-world data segregates into neat, disjoint clusters. The following domain examples illustrate why this assumption is probably too optimistic:

[1] In the version discussed by Demaine et al. [20], real weights are assigned to edges, reflecting the certainty level of the oracle in determining the similarity between objects. We only consider weights in $\{-1, 0, 1\}$, a common restriction in the literature.

[2] There is significant inconsistency in the literature regarding the nomenclature of these problems; as stated, we reserve the name Correlation Clustering for the problem where the input is incomplete.

[3] Given a set of pairs of terminals, $(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p)$, find a set of at most k edges such that after removing these edges, every pair (s_i, t_i) is disconnected

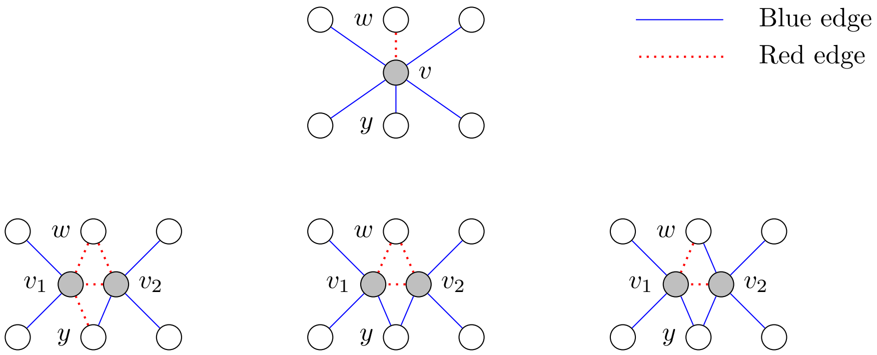


Figure 1 A vertex v in an (incomplete) correlation graph (top). The bottom row gives toy examples of exclusive (left), inclusive (center), and permissive (right) vertex splits of v into v_1 and v_2. For clarity, some red edges incident to v_1 and v_2 are omitted from each figure on the bottom row.

- Document classification: Individual documents often span multiple topics and should therefore belong to multiple topic-clusters;
- Sentiment analysis: A single piece of text can express very different emotions (e.g. sadness mixed with humor);
- Community detection: Individuals typically participate in multiple communities, such as family, professional, and hobbyist groups.
- Language processing: Homonyms like “bat” should belong both to an “animal” cluster as well as a “sports-equipment” cluster.

Hence, the emphasis in clustering has recently shifted towards algorithms for *overlapping clustering* [3, 4, 5, 6, 7, 9, 10, 18, 19, 22, 23, 24, 25, 26, 27, 28]. These models move away from the requirement that data must be partitioned into disjoint subsets by considering a variety of definitions for clusters which may intersect. One natural approach is to edit to a more general target graph class (instead of a cluster graph, consider minimizing the number of edge modifications required to achieve some more complex structure that exhibits strong community structure but allows overlap), but it is difficult to define generalizations that align with many applications.

Motivated by this, Abu-Khzam et al. [3] proposed an alternative model for overlapping clustering based on the concept of *splitting* a vertex into two new vertices, representing an object having two distinct roles within a dataset. This approach led to the problem Cluster Editing with Vertex Splitting, where edges can be added or deleted, and vertices can be split. Here, *splitting* a vertex v means replacing it with two copies, v_1 and v_2, ensuring the union of their (blue) neighbor sets equals the original vertex’s (blue) neighbor set. In fact, Abu-Khzam et al. [3] propose two different vertex splitting operations: one (*exclusive* splitting) where v_1 and v_2 are required to have disjoint (blue) neighborhoods, and another (*inclusive* splitting) where they are allowed to share (blue) neighbors. See Figure 1 for an example. Abu-Khzam et al. [1] show that Cluster Editing with Vertex Splitting is NP-hard and has a $6k$-vertex kernel, where k is the number of edits (edge modifications/vertex splits) allowed. The approximability of this problem remains unknown.

A significant limitation of both existing notions of vertex splitting is that they require red edges to be preserved by both copies of a split vertex. For example, consider a red edge uv in data arising from word classification, where u and v correspond to “bat” and “cat”, respectively. It could be that the edge was produced by our oracle as a result of “bat” being interpreted as a piece of sports equipment, not an animal. However, when “bat” is split so that each meaning has its own vertex, we wish to retain the red edge only on one of the copies of v (the one *not* corresponding to the small flying mammal, as this does have similarities with a cat). Motivated by this, we introduce a new operation called *permissive vertex splitting* which allows replacing a vertex v with two copies v_1 and v_2 with the restriction that if uv is a blue edge (or red edge, respectively), then at least one of uv_1 and uv_2 is a blue edge (red edge, respectively). Beyond that, we are free to choose what to do with the newly-created neighborhoods. We call the new problem variant, where edges can be added or deleted and vertices can be permissively split, Correlation Clustering with Permissive Vertex Splitting (CCPVS). We show that sequences of permissive vertex splits solving this problem correspond directly to a natural notion of overlapping clustering (see Definition 4), adding to the motivation for this definition of splitting.

Extending the prior work relating Correlation Clustering to Multicut, we show that CCPVS can be reduced to the new problem Multicut with Vertex Splitting (MCVS) and vice versa, meaning that the computational complexities of these problems are essentially the same. We then show that MCVS, and hence also CCPVS, are para-NP-hard (with respect to solution size), and NP-hard to approximate within an $n^{1-\epsilon}$ factor for any $\epsilon > 0$. Because of the inherent hardness of CCPVS, we then turn our attention to the setting where there are no zero-edges, i.e., to Cluster Editing with Permissive Vertex Splitting (CEPVS). We show that this problem remains NP-hard, but on the positive side admits a polynomial kernel (and thus is fixed-parameter tractable). Finally, we give a polynomial-time algorithm which provides a 7-approximation for CEPVS.

2 Preliminaries

We refer the reader to the textbook by Diestel [21] for standard graph-theoretic definitions and notation. A *star* is a tree with exactly one internal vertex. In particular, a star has at least two leaves. A *red clique* is a clique in which all edges are red. A *blue clique* is defined similarly. For a positive integer n, we denote by $[n] = \{1, 2, \ldots n\}$ the set of all positive integers up to n. An *incomplete correlation graph* is a simple, unweighted, and undirected graph $G = (V, B, R)$ with two disjoint edge relations B (blue) and R (red). If such a graph is complete, i.e., $B \cup R = \binom{V}{2}$, then we call it a *correlation graph*. For a vertex $v \in V$ we write $N^R(v)$ to denote the set of neighbors adjacent to v via red edges (*red neighbors*) and $N^B(v)$ for those adjacent via blue edges (*blue neighbors*). A *cluster graph* is a correlation graph in which the blue edges form vertex-disjoint cliques (and thus all edges between the cliques are red). We can now formally define our vertex-splitting operation.

▶ **Definition 1.** *A* permissive vertex split *of a vertex v in an (incomplete) correlation graph G is the replacement of v in G with two new vertices v_1 and v_2 such that*

- $N^R(v) \subseteq N^R(v_1) \cup N^R(v_2)$, *and*
- $N^B(v) \subseteq N^B(v_1) \cup N^B(v_2)$,

In other words, we create a new graph where every red (blue) neighbor of v is a red (blue) neighbor of at least one of v_1 or v_2. All other "edges" incident to v_1 and v_2 can be chosen arbitrarily. In particular, in incomplete correlation graphs, we can assume that all these other edges are neutral (i.e., the "edges" do not exist), while in correlation graphs, it is usually simpler to make these edges either red or blue to keep the graph complete. Notably, the edge v_1v_2 can always be assumed to be a red edge as splitting a vertex into two vertices that end up in the same (blue) connected component is never advantageous. For the remainder of this text, unless otherwise specified all vertex splits are permissive. Given a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$ of k vertex splits performed on an (incomplete) correlation graph, we denote the resulting (incomplete) correlation graph by $G_{|\sigma}$. Each vertex u in $G_{|\sigma}$ corresponds to exactly one vertex v in G. We say that v is u's *ancestor*, and that u is a *descendant* of v. If $u = v$, then u and v are *unsplit* vertices. Otherwise we say that v is a *split* vertex and that u is the descendant of a split vertex.

▶ **Definition 2.** *An* erroneous cycle *is a simple cycle that contains exactly one red edge. An (incomplete) correlation graph G* contains *an erroneous cycle if it contains a subgraph that is an erroneous cycle. A* bad triangle *is an erroneous cycle of length 3.*

Erroneous cycles are the canonical obstruction in Correlation Clustering [13, 20], and bad triangles are the canonical obstruction in Cluster Editing. Usually, these problems are formulated as edge editing problems, i.e., delete a minimum number of edges (blue or red) such that the resulting graph has no erroneous cycles/bad triangles. Previous work on Cluster Editing with (inclusive or exclusive) vertex splitting has allowed both edge edits and vertex splits as editing operations [3, 2, 1, 5]. However, we note that permissive vertex splitting is flexible enough to capture all editing operations. First, note that in the setting with blue and red edges, each edge-editing operation can be seen as changing the color of an edge. Now, consider any solution σ in which the color of an edge uv is changed. Then, we construct a new sequence of the same length where this edge edit is replaced by a vertex split. We choose one endpoint (without loss of generality v) and split it into v_1 and v_2. The neighborhood of v_1 is exactly the neighborhood of the initial vertex v except that the edge towards u has the other color. If the edge uv was initially red, then the vertex v_2 has all vertices in the graph as red neighbors. If the edge uv was blue, then we add blue edges between v_2 and all vertices that end up in the same (blue) connected component as (one descendant of) u in $G_{|\sigma}$. The result of the edge edit is now modeled exactly by v_1 and the operation is safe because v_2 cannot participate in any erroneous cycle as it is a twin of (one descendant of) u. Moving forward, we assume that all editing operations are vertex splits, and we say that a sequence σ of vertex splits *clusters* an (incomplete) correlation graph G if $G_{|\sigma}$ has no erroneous cycles. We now state the problems that we study:

Correlation Clustering with Permissive Vertex Splitting (CCPVS)

Input: An (incomplete) correlation graph G and a non-negative integer k.

Problem: Does there exist a sequence σ of at most k vertex splits which clusters G?

Cluster Editing with Permissive Vertex Splitting (CEPVS) is the same problem restricted to correlation graphs. We conclude this section with our main structural insight, which is that clustering an (incomplete) correlation graph G via a sequence of vertex splits is equivalent to performing a very natural notion of overlapping clustering on the vertices of G.

▶ **Definition 3.** *A* covering *of an (incomplete) correlation graph $G = (V, E)$ is a set family $\mathcal{F} \subseteq 2^V$ such that $\bigcup \mathcal{F} = V$. The* cost *of the covering $\mathcal{F}$ is*

$$\mathrm{cost}_G(\mathcal{F}) = \sum_{v \in V} (\#\mathcal{F}(v) - 1),$$

where $\#\mathcal{F}(v) := |\{X \mid v \in X \in \mathcal{F}\}|$ counts the number of sets in $\mathcal{F}$ which contain v.

▶ **Definition 4.** *An* overlapping clustering *of an (incomplete) correlation graph G is a covering $\mathcal{F}$ with the following two properties:*

- *for every blue edge $uv \in B$, there exists at least one cluster $X \in \mathcal{F}$ with $\{u, v\} \subseteq X$, and*
- *for every red edge $uv \in R$, there exists two distinct clusters $X, Y \in \mathcal{F}$ with $u \in X$ and $v \in Y$.*

For a specific edge uv, we say that a clustering covers *the edge if it is blue and the first condition holds and we say that it* resolves *the edge if it is red and the second condition holds.*

▶ **Lemma 5.** *An (incomplete) correlation graph G can be clustered with k vertex splits if and only if G has an overlapping clustering of cost k.*

Proof. For the first direction, let σ be a sequence of k vertex splits clustering $G = (V, B, R)$, i.e., σ produces a graph $G_{|\sigma} = (V_{|\sigma}, E_{|\sigma})$ with no erroneous cycles. We will construct an overlapping clustering $\mathcal{F}$ of cost at most k. We note that it is easy to extend any such overlapping clustering to one of cost exactly k. We begin by choosing an arbitrary vertex $v \in V_{|\sigma}$. We denote by v^* the ancestor of v in V. Let $C_v \subseteq V_{|\sigma}$ be the vertices of the connected component of v in the subgraph of $G_{|\sigma}$ induced by all blue edges, and C_{v^*} be the set of corresponding ancestor vertices in V. We add C_{v^*} to $\mathcal{F}$ and remove C_v from $G_{|\sigma}$. We repeat this process exhaustively. The resulting $\mathcal{F}$ is a covering of G, as each vertex in V has at least one descendant in $V_{|\sigma}$. Moreover, our construction guarantees that each vertex in $V_{|\sigma}$ is considered exactly once. Consequently, for each vertex $v \in V$ we have that $\#\mathcal{F}(v)$ is no greater than the number of descendants of v in $V_{|\sigma}$. Thus, $\mathcal{F}$ has cost at most k. Each blue edge is covered by construction.

For the final step, we show how to augment $\mathcal{F}$ such that all red edges are resolved while maintaining that $\text{cost}_G(\mathcal{F}) \leq k$. We begin by identifying some red edge uv which is not resolved by $\mathcal{F}$. This implies that each of u and v are contained in exactly one cluster $X \in \mathcal{F}$. The red edge uv implies that there is some red edge u_1v_1 in $G_{|\sigma}$, where u_1 is a descendant of u and v_1 is a descendant of v. Moreover, the construction of $\mathcal{F}$ guarantees that there is some blue path between v_1 and a descendant of u, but this latter descendant cannot be u_1 or else we have identified an erroneous cycle in $G_{|\sigma}$. Thus, u has multiple descendants in $G_{|\sigma}$ and is therefore a split vertex. Since u is a split vertex but is only contained in one cluster X in $\mathcal{F}$, we can add the cluster $\{u\}$ to $\mathcal{F}$, thereby resolving uv, while maintaining that $\text{cost}_G(\mathcal{F}) \leq k$. We repeat this process until all red edges are resolved.

For the other direction, let $\mathcal{F}$ be an overlapping clustering of G with cost k. For each vertex v that is contained in more than one cluster set in $\mathcal{F}$, we split v a total of $\#\mathcal{F}(v) - 1$ times. We assign each descendant to one set $X \in \mathcal{F}$ with $v \in X$ and we create blue edges towards all other vertices that are contained in X (or to the specific descendant of a vertex in X that was also assigned to X). All other edges incident to the descendant of v are red. We first show that this construction indeed corresponds to a series of vertex splits. For each blue edge uv, we have that there is some cluster set $X \in \mathcal{F}$ with $u, v \in X$. Hence, if u and/or v are split, then the blue edge uv corresponds to the blue edge between the two copies of u and v that are assigned to X. For each red edge uv, we have that there are some cluster sets $X \neq Y \in \mathcal{F}$ with $u \in X$ and $v \in Y$. Hence, if u and/or v are split, then the red edge uv corresponds to the red edge between (the descendant of) u assigned to X and (the descendant of) v that is assigned to Y. Moreover, we did exactly $\text{cost}(\mathcal{F})$ splits.

It remains to show that the sequence of splits results in a graph that contains no erroneous cycles. Suppose that an erroneous cycle $(u = v_0, v_1, \ldots, v_p = w, u)$ with red edge uw remains. Note that each vertex is assigned to exactly one cluster set in $\mathcal{F}$ as each unsplit vertex is contained in exactly one set in $\mathcal{F}$ and each descendant of a split vertex is assigned to a cluster set by construction. We will show that there is no blue edge between vertices that are assigned to different clusters and no red edge between vertices that are assigned to the same cluster set. This finishes the proof as u and w are then assigned to different cluster sets as they share a red edge, but w_i and w_{i-1} are assigned the same cluster set for each $i \in [p]$, a contradiction. First, assume that there is a blue edge xy where x and y are assigned to different cluster sets. If x and y are both unsplit vertices, then the blue edge between them is not covered by $\mathcal{F}$, a contradiction. Hence, at least one of the two vertices is the descendant of a split vertex and by construction, all edges to vertices that are assigned to different cluster sets are red. Now assume that there is a red edge xy where x and y are assigned to the same cluster set $X \in \mathcal{F}$. Again, if x and y are both unsplit vertices, then they are only contained in X in $\mathcal{F}$ and hence the red edge between them is not resolved by $\mathcal{F}$, a contradiction. So at least one of the two vertices is the descendant of a split vertex and by construction, all edges to vertices that are assigned to X are blue, a final contradiction. This concludes the proof. ◀

3 Incomplete Information

We first consider the more general problem, Correlation Clustering with Permissive Vertex Splitting, which allows for incomplete information. Without vertex splits, it has long been known that Correlation Clustering is in fact equivalent to Multicut [20], which is the problem of deleting a minimum number of edges from a graph $G = (V, E)$ such that every *terminal pair* of distinct vertices in a set $S \subseteq \binom{V}{2}$ is separated in the resulting graph. We define Multicut with Vertex Splitting (MCVS) and show that it is equivalent to CCPVS. We believe that this result is of independent interest, but it will also prove immediately useful as it facilitates the main results of this section. Specifically, CCPVS and MCVS are both para-NP-hard when parameterized by the number of vertex splits, and for any $\varepsilon > 0$ it is NP-hard to approximate either problem within a $n^{1-\varepsilon}$ factor.

First we must define our new Multicut variant. In this context we use standard graph terminology, i.e., we discuss simple, unweighted, and undirected graphs with a single edge relation E. Note that this is equivalent to a correlation graph where edges in E are blue and all other vertex pairs are red, so permissive vertex splits are still well-defined. However, in the Multicut context we can safely assume that all vertex splits are *exclusive*, i.e., whenever splitting a vertex v into descendants v_1 and v_2 we have that $N(v_1) \cup N(v_2) = N(v)$ and $N(v_1) \cap N(v_2) = \emptyset$. The reason is that in Multicut it is never advantageous to assign more edges than required. Note that in the classic version of Multicut, it does not make sense to have an edge between two vertices of a terminal pair. We decided to keep this restriction as it streamlines some of the following arguments. A related technical detail to discuss is what happens to a terminal pair when one of its two vertices is split. We work with the variant where the terminal pair is simply removed in this case. Note that this is equivalent to the variant where we can choose either of the descendants to replace the original vertex in the terminal pair, since, as previously mentioned, we may always assume that any two descendants of the same vertex end up in different connected components.

Multicut with Vertex Splitting (MCVS)

Input: A graph $G = (V, E)$, an integer k, and a set $S \subseteq \binom{V}{2}$ of *terminal pairs* with $S \cap E = \emptyset$.

Problem: Does there exist a sequence σ of at most k (exclusive) vertex splits such that each pair in S is separated in $G_{|\sigma}$?

We now show that CCPVS and MCVS are equivalent problems. Let $(G = (V, B, R), k)$ be an instance of CCPVS. We construct an equivalent instance $(H = (V', E'), S, k)$ of MCVS as follows. For each vertex $v \in V$ we create a vertex v' in V'. Additionally, for each blue edge $uw \in B$ we add the edge $u'w'$ to E'. Finally, for each red edge $uw \in R$ we add the terminal pair (u', w') to S. This completes the construction of H.

▶ **Theorem 6.** *For any integer $k \geq 0$, (G, k) is a yes-instance of* Correlation Clustering with Permissive Vertex Splitting *if and only if (H, S, k) is a yes-instance of* Multicut with Vertex Splitting.

Proof. For the first direction, let $\sigma = (\sigma_1, \sigma_2, \ldots)$ be a sequence of vertex splits clustering G. We will construct a sequence σ' of the same length which separates each pair in S by considering each σ_i in order. If σ_i splits vertex $v \in V$ into v_1 and v_2 then σ'_i splits v' into v'_1 and v'_2. By construction, each neighbor u' of v' corresponds to a blue neighbor u of v. If u is a blue neighbor of v_1, then we create the edge v'_1u'. Otherwise, we create the edge v'_2u'. This completes the construction of σ'. Now, we assume toward a contradiction that some terminal pair (v', u') is connected in $H_{|\sigma'}$. Then there is some path $(v' = w'_0, w_1, \ldots, w'_p = u')$ in $H_{|\sigma'}$. Note that our construction ensures that this path contains at least two edges, and that there is a corresponding blue path $(v = w_0, w_1, \ldots, w_p = u)$ in $G_{|\sigma}$. Moreover, because (v', u') is a terminal pair in $H_{|\sigma'}$, vu is a red edge in $G_{|\sigma}$. Thus, we have identified an erroneous cycle $(v = w_0, w_1, \ldots, w_p = u, v)$ in $G_{|\sigma}$, contradicting that σ clusters G.

For the other direction, let $\sigma' = (\sigma'_1, \sigma'_2, \ldots)$ be a sequence of vertex splits such that no terminal pair is connected in $H_{|\sigma'}$. As before, we will construct a solution σ of the same length by considering each σ'_i in order. If σ'_i splits v' into v'_1 and v'_2, then we split the corresponding vertex v into v_1 and v_2 as follows. If v' is a terminal with partner u', then our construction guarantees that vu is a red edge in G. We create the red edge v_1u if v'_1 (or one of its descendants) is in a different component from u (or one of its descendants) in $H_{|\sigma'}$. Otherwise, we create the red edge v_2u. We mark the relevant pair of descendants so that, when performing subsequent splits, the red edge is always assigned such that its endpoints in G_σ correspond to vertices in different connected components of $H_{|\sigma'}$. Next, for each $u' \in N(v')$, we create the blue edge v_1u if σ'_i assigns u' to $N(v'_1)$. Otherwise, we create the blue edge v_2u. We now assume toward a contradiction that there is an erroneous cycle $(v = w_0, w_1, \ldots, w_p = u, v)$ in $G_{|\sigma}$, with vu being the red edge. The blue path $(v = w_0, w_1, \ldots, w_p = u)$ guarantees that there is a path $(v' = w'_0, w'_1, \ldots, w'_p = u')$ from v' to u' in $H_{|\sigma'}$, and this together with the red edge vu implies that (v', u') is a terminal pair. This contradicts that no terminal pair is connected in $H_{|\sigma'}$. ◀

To reduce MCVS to CCPVS, we simply reverse the previous reduction of CCPVS to MCVS. Formally, let $(G = (V, E), S, k)$ be an instance of MCVS. We create an instance $(H = (V', B, R), k)$ of CCPVS as follows. For each vertex $v \in V$ we add vertex v' to V', for each edge $uw \in E$ we add the blue edge $u'w'$ to B, and finally for each terminal pair $(u, v) \in S$, we add the red edge $u'v'$ to R. Note that B and R are disjoint, as by definition no terminal pair in S is also an edge in E.

▶ **Theorem 7.** *For any integer $k \geq 0$, (G, S, k) is a yes-instance of* Multicut with Vertex Splitting *if and only if (H, k) is a yes-instance of* Correlation Clustering with Permissive Vertex Splitting.

Proof. Note that applying the reduction behind Theorem 6 to H results in the instance (G, S, k). Thus, Theorem 6 already shows that the two instances are equivalent. ◀

Theorems 6 and 7, together with the observation that both reductions exactly preserve the number of vertices, allow us to state the following strong notion of equivalence between CCPVS and MCVS.

▶ **Corollary 8.** *For any function f,* CCPVS *admits a kernel of size $f(k)$ if and only if* MCVS *does. Furthermore, the minimization variant of* CCPVS *admits a polynomial-time $f(n)$-approximation algorithm if and only if the minimization variant of* MCVS *does.*

Now that we have established the equivalence of MCVS and CCPVS, we are ready to show the hardness of both problems.

▶ **Theorem 9.** MCVS *is* NP*-hard even if $k = 2$ Additionally, for any $\varepsilon > 0$ it is* NP*-hard to approximate* MCVS *to within a factor of $n^{1-\varepsilon}$.*

Proof. Let $G = (V, E)$ be the input graph for k-Colorability with $k \geq 3$. We will construct an equivalent input instance $(H, S, k-1)$ for MCVS. We construct the graph H and terminal set S from G as follows. We add V to H and for each edge $uv \in E$, we add (u, v) to S. We then add a new vertex a to H, and create edges from a to all other vertices. This completes the construction.

We first argue that we may assume that any solution of $(H, S, k-1)$ *only* splits a. To see this, let σ be a sequence of vertex splits of length at most $k-1$ such that all terminal pairs are disconnected in $H_{|\sigma}$. Suppose that some vertex $v \neq a$ is split. Before this split, v only has one neighbor a^*, which is either equal to a or a descendant of a. We simply replace the split of v with a split of a^* into a_1^* and a_2^* such that $N(a_1^*) = \{v\}$ and $N(a_2^*) = N(a^*) \setminus \{v\}$. In the resulting graph, v is disconnected from all other vertices in V, and so it is disconnected from all of its terminal partners. We proceed with the assumption that in any solution a is the only split vertex.

Assume that $(H, S, k-1)$ is a yes-instance. We show that then G is k-colorable. By the above, $H_{|\sigma}$ contains k descendants $a_1, \ldots, a_k$ of a and these vertices naturally partition the set V into k sets $C_i = N(a_i)$ for $i \in [k]$. No terminal-pair can appear with both endpoints in one of these sets so the same holds for $E \subseteq S$. Hence, $C_1, \ldots, C_k$ is a valid k-coloring of G.

In the other direction, assume that G has a k-coloring with the color partition $C_1, \ldots, C_k$. Then we can split $a \in H$ a total of $k-1$ times into descendants $a_1, \ldots, a_k$ such that $N(a_i) = C_i$. Since $\{a_1, \ldots, a_k\}$ is independent it is easy to verify that these $k-1$ splits separate every terminal pair in S.

We conclude that MCVS is already NP-hard with parameter $k = 2$ as the above provides a reduction from 3-Colorability. The approximation hardness follows directly from the facts that, given any constant $\varepsilon > 0$, computing an $n^{1-\varepsilon}$-approximation for Chromatic Number is NP-hard [29], and that our constructed instance of MCVS has only $n+1$ vertices. ◀

Taken together with Corollary 8, Theorem 9 gives us the same result for CCPVS.

▶ **Corollary 10.** CCPVS *is* NP*-hard even if $k = 2$ Additionally, for any $\varepsilon > 0$ it is* NP*-hard to approximate* CCPVS *to within a factor of $n^{1-\varepsilon}$.*

4 Complete Information

We now restrict our study to correlation graphs, i.e., we study Cluster Editing with Permissive Vertex Splitting. Our main results are NP-hardness (Section 4.1), a polynomial kernel (Section 4.2), and a polynomial-time 7-approximation (Section 4.3). We begin by introducing a new structure and subsequent lemmas which will be helpful in attaining the latter two results.

▶ **Definition 11.** *A* bad star S *in a correlation graph* G *is a set* $\{v_0, v_1, \dots, v_{|S|-1}\}$ *of vertices where all edges in* $\{\{v_0, v_i\} \mid i \in [|S|-1]\}$ *are blue and all edges in* $\{\{v_i, v_j\} \mid i \neq j \in [|S|-1]\}$ *are red. The vertex* v_0 *is called the center and all other vertices are called leaves. The* weight *of a bad star* $\text{weight}(S)$ *is the number of leaves in the star minus one. A* bad star forest *is a collection* T *of vertex-disjoint bad stars. We write* $\text{weight}(T) := \sum_{S\in T} \text{weight}(S)$ *to denote the sum of weights of its members. A correlation graph* G contains *a bad star forest if it contains a subgraph which is a bad star forest.*

The first lemma states a useful lower bound in terms of bad stars.

▶ **Lemma 12.** *If* G *contains a bad star forest of weight* k *then we need at least* k *vertex splits to cluster* G*.*

Proof. We begin by showing that if $G = (V, B, R)$ contains a (not necessarily induced) subgraph $H = (V_H, B_H, R_H)$ and at least k vertex splits are needed to separate each pair in S, then at least k vertex splits are needed to cluster G. Suppose otherwise. Then, using Lemma 5, there is some overlapping clustering $\mathcal{F}$ of G with cost less than k. We will construct an overlapping clustering $\mathcal{F}_H$ of H with cost less than k. We begin by setting $\mathcal{F}_H = \{X \cap V_H \mid X \in \mathcal{F}\}$. It is clear that this is a covering of H, that every blue edge is covered, and that $\#\mathcal{F}_H(v) \leq \#\mathcal{F}(v)$ for every vertex $v \in V_H$. We now ensure that each red edge is resolved. Let uv be a red edge which is not resolved, so each of u and v belong to only a single cluster $X \in \mathcal{F}_H$. In this case we claim that $\mathcal{F}$ contains two distinct clusters $Y \neq Z$ such that $Y \cap V_H = Z \cap V_H = X$. Otherwise, either $\mathcal{F}$ does not resolve uv or one of u or v is contained in multiple clusters of $\mathcal{F}_H$, both contradictions. Thus, we can safely add the cluster $\{u\}$ (chosen without loss of generality) to $\mathcal{F}_H$, thereby resolving uv while maintaining that $\#\mathcal{F}_H(u) \leq \#\mathcal{F}(u)$. We repeat this process iteratively until all red edges are resolved. In doing so, we produce an overlapping clustering $\mathcal{F}_H$ of H with $\text{cost}_H(\mathcal{F}_H) \leq \text{cost}_G(\mathcal{F}) < k$, a contradiction.

It remains to show that a bad star forest of weight k requires at least k vertex splits to cluster. We begin by showing that a bad star S of weight k requires at least k vertex splits. Let x be the center vertex of S and let $\mathcal{F}$ be an overlapping clustering of S. Let the clusters in $\mathcal{F}$ which contain x be $C_1, C_2, \dots, C_p$. Moreover for each $1 \leq i \leq p$, let $\hat{C}_i = C_i \setminus \{x\}$. Note that we may assume each $\hat{C}_i$ is nonempty, as the cluster $\{x\}$ covers no blue edges and resolves no red edges in S, and can therefore be safely removed from $\mathcal{F}$. Observe also that each leaf v of S must be contained in some set $\hat{C}_i$ since the edge xv is blue. Now suppose that some leaf v is contained in two sets $\hat{C}_i \neq \hat{C}_j$. We remove v from the cluster C_j (chosen arbitrarily) and add the cluster $\{v\}$ to $\mathcal{F}$. The blue edge xv is still covered by C_i, the cluster $\{v\}$ ensures that all red edges incident to v are still resolved, and we have not increased the cost of the clustering. Thus, we may safely assume that the sets $\hat{C}_1, \hat{C}_2, \dots \hat{C}_p$ are a partition of the leaves of S. Consider one such set $\hat{C}_i$. These leaves induce a red clique and none of these red edges is resolved by C_i, so we have that at least $|\hat{C}_i| - 1$ of these leaves are contained in multiple clusters in $\mathcal{F}$. Since we also know that $\#\mathcal{F}(x) = p$, we conclude

$$\begin{aligned}\text{cost}_S(\mathcal{F}) &\geq (|\hat{C}_1| - 1) + (|\hat{C}_2| - 1) + \ldots + (|\hat{C}_p| - 1) + \#\mathcal{F}(x) - 1 \\ &= |\hat{C}_1| + |\hat{C}_2| + \cdots + |\hat{C}_p| - p + p - 1 = |S \setminus \{x\}| - 1 = \text{weight}(S) = k\end{aligned}$$

Finally, let T be a bad star forest made up of t bad stars $S_1, S_2, \dots, S_t$. Let k be the weight of T and suppose toward a contradiction that T admits an overlapping clustering $\mathcal{F}$ of cost less than k. Then, we repeat the technique from earlier in this proof to construct overlapping clusterings $\mathcal{F}_{S_1}, \mathcal{F}_{S_2}, \dots \mathcal{F}_{S_t}$ of the bad stars. Because the bad stars are vertex-disjoint, we have that $\mathrm{cost}_{S_1}(\mathcal{F}_{S_1}) + \mathrm{cost}_{S_2}(\mathcal{F}_{S_2}) + \ldots + \mathrm{cost}_{S_t}(\mathcal{F}_{S_t}) \leq \mathrm{cost}_T(\mathcal{F}) < k$. This implies that there is some S_i such that $\mathrm{cost}_{S_i}(\mathcal{F}_{S_i})$ is less than the weight of S_i, but we have already proven that this is impossible. ◀

The second lemma states that every optimal solution contains a cluster that contains all vertices of a sufficiently large blue clique.

▶ **Lemma 13.** *If a correlation graph G contains a blue clique C of size at least $k+1$, then any overlapping clustering $\mathcal{F}$ of cost at most k contains a set $X \in \mathcal{F}$ with $C \subseteq X$.*

Proof. Let C be a blue clique in G of size at least $k+1$ and let $\mathcal{F}$ be any overlapping clustering of cost at most k. Assume towards a contradiction that $\mathcal{F}$ does not contain a set X with $C \subseteq X$. Since $\mathcal{F}$ has cost at most k, there exists a vertex $v \in C$ that is contained in exactly one set $Y \in \mathcal{F}$. Moreover, since Y does not contain all vertices of C by assumption, there exists a vertex $u \in C \setminus Y$. Since C is a blue clique, the edge uv is blue and is covered by some set $Z \in \mathcal{F}$. Observe that $Y \neq Z$ as $u \in Z$ and $u \notin Y$. Since Z covers the edge uv, it holds that $v \in Z$, a contradiction to the assumption that v is only contained in Y. ◀

4.1 NP-hardness

We now show that Cluster Editing with Permissive Vertex Splitting is NP-hard.

▶ **Proposition 14.** *Deciding whether a given correlation graph admits an overlapping clustering of cost at most k is* NP*-hard.*

Proof. We reduce from Vertex Cover. Let $(G = (V, E), k')$ be an instance of Vertex Cover. We construct a correlation graph H as follows. Let U be a set of $k'+1$ vertices (not contained in V). The vertex set of H is $U \cup V$. For each edge $e = uv \in E$, we add a red edge uv to H. All other edges (including all edges incident to a vertex in U) are blue. Finally, we set $k = k'$.

We next show that the reduction is correct. First assume that there is an overlapping clustering $\mathcal{F}$ of H of cost at most k. By Lemma 13, for each vertex $v \in V$, there exists a set $X_v \in \mathcal{F}$ with $U \cup \{v\} \subseteq X_v$. Note that $U \subset X_u \cap X_v$ for any pair $u, v \in V$ and therefore $X_u = X_v$ as otherwise the cost of $\mathcal{F}$ is at least $|U| > k$. Hence, there exists a set $X \in \mathcal{F}$ with $U \cup V \subseteq X$. Since all blue edges are covered by X, we next focus on resolving all red edges. Note that since X contains all vertices in H and the cost of $\mathcal{F}$ is at most k, all remaining sets in $\mathcal{F}' = \mathcal{F} \setminus X$ contain at most k vertices combined. If for some red edge uv none of the two vertices u or v is contained in a set in $\mathcal{F}'$, then this red edge is not resolved by $\mathcal{F}$. Thus for each red edge, at least one of the two endpoints is contained in a set in $\mathcal{F}'$. Note that this immediately implies that G contains a vertex cover of size at most k (all vertices that are contained in a set in $\mathcal{F}'$).

For the other direction, assume that G contains a vertex cover S of size at most k. We construct an overlapping clustering $\mathcal{F}$ of H of cost at most k as follows. The family $\mathcal{F}$ contains one set $X = U \cup V$ and for each vertex $v \in S$, it contains a set $X_v = \{v\}$. Note that the cost of $\mathcal{F}$ is at most k and all blue edges in H are covered by X. Moreover, each red edge uw in H is resolved as by construction it holds that $\mathcal{F}$ contains the set $X_u = \{u\}$ or $X_w = \{w\}$. Without loss of generality, let $\mathcal{F}$ contain X_u. Then, the red edge uw is resolved as w is contained in X and u is contained in $X_u \neq X$. This concludes the proof. ◀

4.2 Polynomial Kernel

We next show that Cluster Editing with Permissive Vertex Splitting parameterized by k admits a polynomial kernel. Note that this is in stark contrast to the para-NP-hardness of Correlation Clustering with Permissive Vertex Splitting parameterized by k.

▶ **Theorem 15.** Cluster Editing with Permissive Vertex Splitting *parameterized by the number of vertex splits admits a kernel with $O(k^3)$ vertices.*

Proof. Let $(G = (V, B, R), k)$ be the input instance of CEPVS. We begin by computing an inclusion-maximal bad star forest T in G. If $\text{weight}(T) \geq k$, then we conclude, according to Lemma 12, that (G, k) is a no-instance and output an appropriate trivial kernel.

Otherwise let S be the vertices of T and note that $|S| \leq 3\,\text{weight}(T) \leq 3k$. Since T is inclusion-maximal, we know that $G \setminus S$ cannot contain any bad stars and in particular no bad triangles. We conclude that $G \setminus S$ is therefore a cluster graph. Let $C_1, C_2, \ldots, C_p$ be these clusters. We next exhaustively apply the following simple reduction rule.

▶ **Reduction rule 1.** *If G contains a blue clique C such that all edges with one endpoint in C are red, then remove C from G.*

Next, we bound the number p of cliques in $G \setminus S$ as follows. Assume that $G \setminus S$ contains at least $4k + 1$ cliques. Note that by application of Reduction Rule 1, all clusters in $G \setminus S$ have at least one blue edge towards S. Pick for each clique C in $G \setminus S$ one such blue edge towards S and let v_C be the endpoint in C of this edge. Note that these chosen edges form a collection of vertex-disjoint stars with all centers in S (but not necessarily all vertices in S being centers). Moreover, since the vertices v_C and $v_{C'}$ belong to different cliques for each pair $C \neq C'$ of cliques in $G \setminus S$, the edge between the two is red. Hence, all stars with at least two leaves in $V \setminus S$ are bad stars. Let $S' \subseteq S$ be the set of vertices in S that are not the center of such bad stars, that is, vertices in S for which we chose at most one incident blue edge as a representative for a clique. Let $S^* = S \setminus S'$. Since we chose at most one blue edge sv_C for each vertex $s \in S'$, the number of chosen blue edges included in bad stars is at least

$$(4k + 1) - |S'| \geq (4k + 1) - |S| + |S^*| \geq (4k + 1) - 3k + |S^*| = k + 1 + |S^*|.$$

Hence, the weight of the constructed collection of bad stars is at least $k+1$ and by Lemma 12, we conclude that (G, k) is a no-instance. Thus, if $G \setminus S$ contains at least $4k + 1$ cliques after applying Reduction Rule 1 exhaustively, we can return a trivial no-instance. Otherwise, the number p of cliques is bounded by $4k$.

We are now left with the task of bounding the size of each individual cluster C_i to arrive at a polynomial kernel. To that end, we apply the following marking and deletion procedure to each cluster: For a fixed cluster C_i, begin with an initially empty set M_i. For each vertex $v \in S$, arbitrarily mark $k + 1$ red and $k + 1$ blue neighbors of v in C_i by adding them to M_i (or all red/blue neighbors if there are at most k). Note that we mark at most $|M_i| \leq |S|(2k + 2) \leq 6k^2 + 6k$ vertices this way.

▶ **Reduction rule 2.** *For any cluster C_i, delete all (unmarked) vertices in $C_i \setminus M_i$ from G.*

Let $\hat{G}$ be the graph obtained after applying the reduction rule to some cluster C_i. We now need to show that this reduction rule is safe and sound. Let $R_i := C_i \setminus M_i$ be the vertices removed by the reduction rule.

First note that if $\mathcal{F}$ is an overlapping clustering of G, then $\mathcal{F} \setminus R_i$ (interpreted as a multiset[4], that is, the same cluster might appear multiple times in it) is trivially an overlapping clustering of $\hat{G}$ and $\text{cost}_{\hat{G}}(\mathcal{F} \setminus R_i) \leq \text{cost}_G(\mathcal{F})$. Thus, the reduction rule is safe.

To prove soundness, let $\hat{\mathcal{F}}$ be an overlapping clustering of $\hat{G}$ with $\text{cost}_{\hat{G}}(\hat{\mathcal{F}}) \leq k$. Let $u \in R_i$ be one of the removed vertices. We argue that we can include u in the clustering without increasing the cost. Note that since we removed a vertex, the size of C_i was initially at least $2k + 2$ and hence, by Lemma 13, we have that $\hat{\mathcal{F}}$ contains a set $\hat{C}$ with $C_i \subseteq \hat{C}$. We add u to this cluster and now argue that u does not have to be included in any further clusters if (G, k) is a yes-instance. To that end, we show that every edge incident to u is already covered/resolved by this new clustering.

Let uv be any blue edge incident to u. Note that if $v \in C_i$ then uv is covered by $\hat{C}$, so we may assume that $v \in S$. Then v has at least $k + 2$ blue neighbors in C_i as otherwise we would have marked u. Let N be a set of $k + 1$ neighbors of v in C_i that were marked. By Lemma 13, there exists a cluster set $X \in \hat{\mathcal{F}}$ with $N \cup \{v\} \subseteq X$. Hence, $X = \hat{C}$ as otherwise the cost of $\hat{\mathcal{F}}$ would be at least $k + 1$ as each vertex in N would appear in at least two sets. Thus, uv is covered by $\hat{C} \cup \{u\}$ in the constructed overlapping clustering.

Now let uv be any red edge incident to u. Again, we claim that because u was unmarked, v must have at least $k + 2$ red neighbors in C_i. Either $v \in S$, in which case the argument is the same as before, or $v \in V \setminus (S \cup C_i)$. In this case, all of C_i is contained in v's red neighborhood, and we have already observed that C_i has at least $2k + 2$ vertices. Let N be a set of $k + 1$ red neighbors of v in C_i that were marked. Note that v is contained in a set $X \neq \hat{C} \in \hat{\mathcal{F}}$ as otherwise each vertex in N would be contained in at least two sets and $\text{cost}(\hat{\mathcal{F}}) \geq k + 1$. Hence, uv is resolved as $u \in \hat{\mathcal{C}} \cup \{u\}$ and $v \in X$.

We conclude that the resulting clustering covers all blue edges incident to u and resolves all red edges incident to u at the same cost as the clustering $\hat{\mathcal{F}}$. By repeating the procedure for the remaining vertices of R_i we conclude that there exists a clustering $\mathcal{F}$ which clusters G and $\text{cost}_G(\mathcal{F}) = \text{cost}_{\hat{G}}(\hat{\mathcal{F}})$. Repeating this argument for every cluster demonstrates that Rule 2 is indeed sound.

Finally, note that after application of Rule 1 and Rule 2 to a yes-instance, we have $p \leq 4k$ clusters of size at most $|S|2(k+1) \leq 6k^2 + 6k$ each and therefore the total number of vertices in the end is at most $|S| + 4k(6k^2 + 6k) = 24k^3 + 24k^2 + 3k \in O(k^3)$. This concludes the proof. ◀

4.3 Constant-Factor Approximation

We conclude this section with a constant-factor approximation for Cluster Editing with Permissive Vertex Splitting. Again, this is in stark contrast to Correlation Clustering with Permissive Vertex Splitting.

▶ **Theorem 16.** Cluster Editing with Permissive Vertex Splitting *admits a 7-approximation in polynomial time.*

Proof. Let $G = (V, E)$ be a correlation graph. We again begin by computing an inclusion-maximal bad star forest T in G. By Lemma 12, the weight of T is at most opt (the minimum cost of an overlapping clustering of G). Since the number of vertices in a bad star is at most thrice its weight (a bad triangle has weight one and contains three vertices), the set S of

[4] Technically an overlapping clustering cannot be a multiset. We refer the reader to the proof of Lemma 12, in which we formally show how to adapt this construction into an overlapping clustering with cost bounded by $\text{cost}_G(\mathcal{F})$.

vertices in T is at most $3\,\mathrm{opt}$. Since T is inclusion-maximal, the graph induced by $V \setminus S$ does not contain any bad star, that is, the blue edges form a cluster graph. Let $\mathcal{C}$ be the set of (blue) cliques in this graph. If $|\mathcal{C}| \leq 1$, then we find a simple 3-approximation by putting each vertex $v \in S$ into its own cluster set X_v and adding one cluster set $X_S = V$. Note that the cost of this overlapping clustering is $|S| \leq 3\,\mathrm{opt}$. Hence, we assume for the remainder of the proof that $|\mathcal{C}| \geq 2$.

Next, we restrict our search to a solution that contains one cluster set X_S with $S \subseteq X_S$ and for each vertex $v \in S$ one cluster set $X_v = \{v\}$. Note that we can add these sets to any solution (if they are not already present within the solution) to get a new solution whose cost is at most $2|S| \leq 6\,\mathrm{opt}$ larger than the original. We call overlapping clusterings that satisfy the above *simple solutions* and we denote the minimum cost of a simple solution by opt'.

We next show that there is always a simple solution of cost opt' that contains for each clique $C \in \mathcal{C}$ a cluster set X_C with $C \subseteq X_C$. Start with any simple solution $\mathcal{F}$ of cost opt' and any clique $C \in \mathcal{C}$ and assume that $\mathcal{F}$ does not contain a cluster set containing C. We prove that in this case each vertex in C is contained in at least two cluster sets in $\mathcal{F}$. Assume towards a contradiction that some vertex $v \in C$ is contained in exactly one cluster set Y (note that by definition of overlapping clusterings, each vertex is contained in at least one cluster set). Since we assumed that no cluster set completely contains C, there exists a vertex $u \in C \setminus Y$. However, since u and v are contained in the same clique C, the edge between them is blue and has to be covered by some cluster set $Z \in \mathcal{F}$ (and hence Z has to contain both u and v). Note that $Z \neq Y$ since $u \in Z$ but $u \notin Y$. This contradicts the assumption that v is only contained in cluster set Y.

We construct a new simple solution $\mathcal{F}'$ of cost opt' by removing all vertices in C from all cluster sets in $\mathcal{F}$ and adding them all to X_S. In addition, we add one new cluster set $X_C = C$. Note that the cost of $\mathcal{F}'$ is at most the cost of $\mathcal{F}$ as we removed each vertex in C from at least two cluster sets and added them to exactly two cluster sets. Moreover, the new solution is indeed an overlapping clustering as all blue edges incident to a vertex in C are covered by X_S as all blue neighbors are either in S or in C. Since no red neighbors of any vertex in C are contained in C, the new cluster set X_C ensures that all red edges incident to vertices in C are resolved. Repeating the above for all cliques in $\mathcal{C}$ yields a simple solution of cost opt' that contains for each clique $C \in \mathcal{C}$ a cluster set X_C with $C \subseteq X_C$.

The next step is to show that there is always an optimal simple solution (a simple solution of cost opt') in which $X_C \neq X_{C'}$ for any pair $C \neq C' \in \mathcal{C}$. Start with any optimal simple solution $\mathcal{F}$ that contains a cluster set $X_C \supseteq C$ for each clique $C \in \mathcal{C}$ and assume that $X_{C_1} = X_{C_2}$ for some cliques $C_1 \neq C_2$. Observe that all vertices from at least one of the two cliques are contained in at least two cluster sets each as if there are vertices $u \in C_1$ and $v \in C_2$ that are only contained in $X_{C_1} = X_{C_2}$, then the red edge between them is not resolved by $\mathcal{F}$. Without loss of generality, let all vertices of C_1 be contained in at least two cluster sets each. Then, we construct a new simple solution $\mathcal{F}'$ of cost opt' by removing all vertices in C_1 from all cluster sets in $\mathcal{F}$ and adding them all to X_S and adding one new cluster set $X_{C_1} = C_1$. The proof that this is correct is exactly the same as before. The cost of $\mathcal{F}'$ is at most the cost of $\mathcal{F}$ as we removed each vertex in C_1 from at least two cluster sets and added them to exactly two cluster sets. Moreover, the new solution is indeed an overlapping clustering as all blue edges incident to a vertex in C_1 are covered by X_S and the new cluster set X_{C_1} ensures that all red edges incident to vertices in C are resolved. Repeating the above for all cliques in $\mathcal{C}$ yields an optimal simple solution that contains for each clique $C \in \mathcal{C}$ a cluster set $X_C \supseteq C$ such that $X_C \neq X_{C'}$ for all $C \neq C' \in \mathcal{C}$.

Next, we guess which clique $C^* \in \mathcal{C}$ satisfies $X_{C^*} = X_S$ in an optimal simple solution satisfying all of the above.[5] By that, we mean that we try all possibilities of the following and return the best solution found. For each clique $C \in \mathcal{C} \setminus \{C^*\}$, we compute a minimum vertex cover K_C of the blue edges between C and S in $O(n^3)$ time using Kőnig's theorem (note that the considered graph is bipartite by construction). We add each vertex in $K_C \cap S$ to X_C and each vertex in $K_C \cap C$ to X_S.

We claim that $\sum_{C\in\mathcal{C}\setminus\{C^*\}} |K_C| \leq \text{opt}' - |S|$. By the above arguments, we can start with an overlapping clustering $\mathcal{F}$ consisting of one cluster set $X_S = S \cup C^*$, one cluster set $X_C = C$ for each $C \in \mathcal{C} \setminus \{C^*\}$, and one cluster set $X_v = \{v\}$ for each $v \in S$. Note that all red edges are resolved and all blue edges except for those between S and $V \setminus (S \cup C^*)$ are covered. To cover a blue edge uv with $u \in S$ and $v \in C$ for some $C \in \mathcal{C} \setminus \{C^*\}$, there are three possibilities: We can add u to X_C, we can add v to X_S, or there exists a different cluster set $Y \in F$ with $\{u, v\} \subseteq Y$. Note that in the third case, we can remove v from Y and add it to X_S and still get an optimal simple solution. Moreover, the optimal way to cover all blue edges between S and $V \setminus (S \cup C^*)$ using the first two possibilities corresponds exactly to $\sum_{C\in\mathcal{C}\setminus\{C^*\}} |K_C|$. Since now all red edges are resolved and all blue edges are covered and the cost of the constructed overlapping clustering is

$$|S| + \sum_{C\in\mathcal{C}\setminus\{C^*\}} |K_C| \leq \text{opt}' \leq 7\,\text{opt},$$

we successfully computed a factor-7 approximation in polynomial time. ◀

We leave it as an open problem to improve the approximation factor. We conjecture that a refined concept of a simple solution might yield an approximation factor of 4.

5 Conclusion

We have introduced permissive vertex splitting, which generalizes the earlier exclusive and inclusive vertex splitting notions by allowing symmetry with respect to "positive" and "negative" pairwise similarity data. Our type of vertex splitting turns out to be quite satisfying, as it corresponds to a natural definition of overlapping clustering. Unfortunately, the general case of Correlation Clustering with Permissive Vertex Splitting is rather intractable, as it is para-NP-hard and admits no $n^{1-\varepsilon}$ approximation in polynomial time for any $\varepsilon > 0$ (unless $\mathsf{P} = \mathsf{NP}$). On the positive side, when restricted to datasets with complete data we obtain a kernel with $O(k^3)$ vertices and a polynomial time 7-approximation. Interesting questions remain, for example whether one can reduce our approximation factor to 4 (or even lower), whether a kernel with only linearly many vertices exists (as is the case when only inclusive splits and edge-edits are allowed [1]), or whether refined lower bounds in terms of running time or approximation factor can be found. Future work might also consider the parameterized complexity of Correlation Clustering with Permissive Vertex Splitting and Cluster Editing with Permissive Vertex Splitting with respect to structural parameters of the input, or with restrictions on the number of clusters which contain any given node.

Finally, we would like to amplify the call of Abu-Khzam et al. [1] to extend the study of vertex splitting (exclusive, inclusive, or permissive) to other classes of target graphs, many of which (e.g., bicluster graphs, s-cliques, s-clubs, s-plexes, k-cores, and γ-quasi-cliques) have been proposed as alternatives to cliques in clustering applications.

[5] We can assume that one such clique always exists, but even if this was not the case, the following proof still works if the guess $\{C^*\} = \emptyset$ yields an optimal simple solution.

References

1 Faisal N. Abu-Khzam, Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Judith Egan, Serge Gaspers, Alexis Shaw, Peter Shaw, Blair D. Sullivan, and Petra Wolf. Cluster editing with vertex splitting. *arXiv preprint*, 2023. `arXiv:1901.00156`.

2 Faisal N. Abu-Khzam, Joseph R. Barr, Amin Fakhereldine, and Peter Shaw. A greedy heuristic for cluster editing with vertex splitting. In *Proceedings of the 4th International Conference on Artificial Intelligence for Industries (AI4I)*, pages 38–41. IEEE, 2021.

3 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In *Proceedings of the 5th International Symposium on Combinatorial Optimization (ISCO)*, pages 1–13. Springer, 2018.

4 Sanjeev Arora, Rong Ge, Sushant Sachdeva, and Grant Schoenebeck. Finding overlapping communities in social networks: Toward a rigorous approach. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC)*, pages 37–54. Association for Computing Machinery, 2012.

5 Emmanuel Arrighi, Matthias Bentert, Pål Grønås Drange, Blair D. Sullivan, and Petra Wolf. Cluster editing with overlapping communities. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 2:1–2:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

6 Gard Askeland. Overlapping community detection using cluster editing with vertex splitting. Master's thesis, University of Bergen, Bergen, Norway, 2022.

7 Sanghamitra Bandyopadhyay, Garisha Chowdhary, and Debarka Sengupta. FOCS: Fast overlapped community search. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):2974–2985, 2015.

8 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

9 Jeffrey Baumes, Mark Goldberg, and Malik Magdon-Ismail. Efficient identification of overlapping communities. In *Proceedings of the 2005 IEEE International Conference on Intelligednce and Security Informatics (ISI)*, pages 27–36. Springer, 2005.

10 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*, 35(1):1–32, 2013.

11 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.

12 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

13 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

14 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending Grothendieck's inequality. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 54–60. IEEE, 2004.

15 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.

16 Jianer Chen and Jie Meng. A 2k kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

17 Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1082–1104. IEEE, 2023.

18 Alex Crane, Brian Lavallee, Blair D. Sullivan, and Nate Veldt. Overlapping and robust edge-colored clustering in hypergraphs. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 143–151, 2024.

19 George B. Davis and Kathleen M. Carley. Clearing the FOG: Fuzzy, overlapping groups for social networks. *Social Networks*, 30(3):201–212, 2008.

20 Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

21 Reinhard Diestel. *Graph Theory*. Springer, 2012.

22 Nan Du, Bai Wang, Bin Wu, and Yi Wang. Overlapping community detection in bipartite networks. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 176–179, 2008.

23 Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Overlapping community detection in labeled graphs. *Data Mining and Knowledge Discovery*, 28(5-6):1586–1610, 2014.

24 Reynaldo Gil-García and Aurora Pons-Porrata. Dynamic hierarchical algorithms for document clustering. *Pattern Recognition Letters*, 31(6):469–477, 2010.

25 Mark Goldberg, Stephen Kelley, Malik Magdon-Ismail, Konstantin Mertsalov, and Al Wallace. Finding overlapping communities in social networks. In *Proceedings of the 2nd IEEE International Conference on Social Computing (SC)*, pages 104–113, 2010.

26 Steve Gregory. An algorithm to find overlapping community structure in networks. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 91–102. Springer, 2007.

27 Qinna Wang and Eric Fleury. Uncovering overlapping community structure. In *Proceedings of the 2nd International Workshop on Complex Networks (COMPLEX NETWORKS)*, pages 176–186. Springer, 2010.

28 Xufei Wang, Lei Tang, Huiji Gao, and Huan Liu. Discovering overlapping groups in social media. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM)*, pages 569–578, 2010.

29 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Daisy Bloom Filters

Ioana O. Bercea
KTH Royal Institute of Technology, Stockholm, Sweden

Jakob Bæk Tejs Houen
BARC, University of Copenhagen, Denmark

Rasmus Pagh
BARC, University of Copenhagen, Denmark

Abstract

A filter is a widely used data structure for storing an approximation of a given set S of elements from some universe $\mathcal{U}$ (a countable set). It represents a superset $S' \supseteq S$ that is "close to S" in the sense that for $x \notin S$, the probability that $x \in S'$ is bounded by some $\varepsilon > 0$. The advantage of using a Bloom filter, when some false positives are acceptable, is that the space usage becomes smaller than what is required to store S exactly.

Though filters are well-understood from a worst-case perspective, it is clear that state-of-the-art constructions may not be close to optimal for particular distributions of data and queries. Suppose, for instance, that some elements are in S with probability close to 1. Then it would make sense to always include them in S', saving space by not having to represent these elements in the filter. Questions like this have been raised in the context of Weighted Bloom filters (Bruck, Gao and Jiang, ISIT 2006) and Bloom filter implementations that make use of access to learned components (Vaidya, Knorr, Mitzenmacher, and Krask, ICLR 2021).

In this paper, we present a lower bound for the expected space that such a filter requires. We also show that the lower bound is asymptotically tight by exhibiting a filter construction that executes queries and insertions in worst-case constant time, and has a false positive rate at most ε with high probability over input sets drawn from a product distribution. We also present a Bloom filter alternative, which we call the *Daisy Bloom filter*, that executes operations faster and uses significantly less space than the standard Bloom filter.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Bloom filters, input distribution, learned data structures

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.9

Related Version *Full Version*: `https://arxiv.org/abs/2205.14894`

Funding Supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

1 Introduction

This paper shows asymptotically matching upper and lower bounds for the space of an optimal (Bloom) filter when the input and queries come from specific distributions. For a set S of keys (the input set), a filter on S with parameter $\varepsilon \in (0, 1)$ is a data structure that answers membership queries of the form "is x in S?" with a one-sided error: if $x \in S$, then the filter always answers YES, otherwise it makes a mistake (i.e., a false positive) with probability at most ε. The Bloom filter [9] is the most widely known such filter, although more efficient constructions are known [2, 3, 5, 6, 20, 25, 34, 41, 42, 45]. Filters are also intimately related to dictionaries (or hash tables), the latter of which always answer membership queries exactly.

When errors can be tolerated, (Bloom) filters are much better than dictionaries at encoding the input set: they require $\Theta(n \log(1/\varepsilon))$ bits to represent a set of size n, versus the $\geq n \log(u/n)$ bits that a dictionary would require (here, u is the size of the universe). As such, filters are often used in conjunction with dictionaries to speed up negative queries. In particular, filters are often stored in a fast but small memory and are used to "filter out" a majority of negative queries to a dictionary (which might reside in big but slow memory). Because of this, they have proved to be extremely popular in practice and research on them continues to this day, both in the direction of practical implementations [19, 24, 44] and on the theoretical front [4, 5, 34]. For instance, recent advances in filter design have included making them dynamic, resizeable and lowering the overall space that they require.

The filter encoding. In this paper, we ask ourselves what should optimal filters look like when they encode sets that come from a specific distribution. While this question has been resolved for exact encodings (i.e., entropy), no similar concepts are known for filter encodings. Indeed, considering input distributions raises several technical questions. For instance, it is not even clear how to define the concept of approximate membership with respect to a set drawn from a distribution. Should we assume that the input set is given to us in full before we build our filter and allocate memory? Moreover, we would like to obtain designs that are never worse than filters with no knowledge of the input distribution, both in space allocated and time required to perform every operation. Should we then require that the false positive guarantee hold for every possible input set or just on average over the input distribution?

We also study optimality when additionally, we have access to a distribution over queries. This is especially important for applications in which the performance of the filter is measured over a sequence of queries, rather than for each query separately [11, 26]. At the extreme end of this one can consider adversarial settings, in which an adversary forces the filter to incur many false positives (which can cause a delay in the system by forcing the filter to repeatedly access the slow dictionary). In these settings, defining what it means for the filter to behave efficiently can be a challenge and several definitions have been considered [2, 38–40]. For us, the challenge is to use the query distribution to obtain gains, while making sure that the filter does not on average exhibit more false positives than usual. This is natural when each false positive has the same cost, independent of the query element.

To this end, we consider a natural generative model of input sets and queries. Specifically, we let $\mathcal{P}$ and $\mathcal{Q}$ denote two distributions over the universe $\mathcal{U}$ of keys and let p_x (and q_x, respectively) denote the probability that a specific key $x \in \mathcal{U}$ is sampled from $\mathcal{P}$ (and $\mathcal{Q}$, respectively). The input set S is generated by n independent draws (with replacement) from $\mathcal{P}$ and we let $\mathcal{P}_n$ denote this product distribution.[1]

We then define approximate membership for a fixed set S to mean that the average false positive probability over $\mathcal{Q}$ is at most ε. Specifically, let $\mathcal{F}$ denote the filter and let $\mathcal{F}(S, x) \in \{\mathsf{YES}, \mathsf{NO}\}$ denote the answer that $\mathcal{F}$ returns when queried on an element $x \in \mathcal{U}$, after having been given $S \subseteq \mathcal{U}$ as input. Then we propose the following definition:

▶ **Definition 1.** *For any ε with $0 < \varepsilon < 1$, we say that $\mathcal{F}$ is a $(\mathcal{Q}, \varepsilon)$*-filter for *$S$ if it satisfies the following conditions:*

1. *No false negatives: For all $x \in S$, we have that* $\Pr[\mathcal{F}(S, x) = \mathit{YES}] = 1$.
2. *Bounded false positive rate:*

$$\sum_{x \in \mathcal{U} \setminus S} q_x \cdot \Pr[\mathcal{F}(S, x) = \mathit{YES}] \leq \varepsilon$$

[1] We do not consider multiplicities although our design can be made to handle them by using techniques from counting filters [6, 10, 41, 43].

We note a detail in the above definition that has important technical consequences and that is, the false positive rate is not computed with respect to the input distribution (i.e., the probability of a false positive only depends on the internal randomness of the filter and not the random process of drawing the input set). As a consequence, we can argue about filter designs that work over all input sets except some that occur very rarely under $\mathcal{P}_n$. This is stronger than saying that $\mathcal{F}$ works only on average over $\mathcal{P}_n$. Moreover, we also want designs that do not require knowing the specific realization of the input set in advance. Our dependency on $\mathcal{P}_n$ shows up in the space requirements of the filter.

Access to $\mathcal{P}$ and $\mathcal{Q}$. For simplicity, we consider filter designs that have oracle access to $\mathcal{P}$ and $\mathcal{Q}$: upon seeing a key x, we also get p_x and q_x. We assume that this is done in constant time and do not account for the size of the oracle when we bound the size of the filter. Critics of this model have argued that assuming oracle access to a distribution over the universe is too strong of an assumption. Indeed, this is a valid concern, since we are talking about a data structure that is meant to save space over a dictionary. We try to alleviate this concern in several ways. On one hand, our construction can tolerate mistakes. In particular, our designs are robust even if we have a constant factor approximation for p_x and q_x, in the sense in which the space increases only by $O(n)$ bits and the time to perform each operation by an added constant. The assumption of access to such approximate oracles is standard [13, 23] and can be based on samples of historical information, on frequency estimators such as Count-Min [17] or Count-Sketch [16], or on machine learning models (see for instance, the neural-net based frequency predictor of Hsu et al. [32]). This view is indeed part of an emerging body of work on algorithms with predictions, to which the data structure perspective is just beginning to contribute [14, 18, 27–30, 36, 37, 48].

On the other hand, empirical studies have shown that significant gains are possible even when using off-the-shelf, "simplistic" learned components such as random forest classifiers. In particular, the Partitioned Learned Bloom Filter [48] and the Adaptive Learned Bloom Filter [18] consider settings in which the size of the learned component is comparable to the size of the filter itself (rather than proportional to the size of the universe), and compare the traditional Bloom filter design [9] with a learned design whose space includes the random forest classifier. In one experiment with a universe of $\approx 138,000$ keys and a classifier of 136Kb, [18] show that, within the range 150-300Kb, there is a 98% decrease in false positive rate compared to the original Bloom filter. This continues to hold for larger universe ($\approx$ 450, 000 keys) with total allocated space between 200Kb and 1000Kb. A discussion of how our current (theoretical) design compares to the ones in [18] and [48] can be found in Section 1.2.

Finally, strictly speaking, our designs do not necessarily rely on knowing p_x and q_x for every element inserted or queried. As we will see in the next section, our designs depend rather on knowing which subset of the universe a key x belongs to. This corresponds to a partitioning of the universe that mainly depends on the ratio q_x/p_x, rather than the individual values of p_x and q_x (with the exception of values of p_x and q_x that are very small, e.g., smaller than $1/n$). This can conceivably lead to even smaller oracles that just output the partition to which an element belongs. We also do not need to query the entire universe in order to set the internal parameters of the filter, in contrast to [18, 48].

Weighted Bloom filters. The design that we propose starts by gathering information about the input and query distributions, using $\mathsf{polylog}(n)$ samples.[2] This information is used to estimate the internal parameters of the filter which are then used to allocate space for the filter and implement the query and insert operations. Thus, the most important aspect of our design is in setting the aforementioned internal parameters.

[2] Elements that are inserted in the set during that time can be stored in a small dictionary that only requires $\mathsf{polylog}(n)$ bits, see Section 4.3.

As a baseline for comparison, we can consider the classic Bloom filter design which allocates an array of $\approx 1.44 \cdot n \log(1/\varepsilon)$ bits and hashes every key to $\log(1/\varepsilon)$ locations in the array. Upon insertion, the corresponding bits are set to 1 and a query returns a YES if and only if all locations are set to 1. The more locations we hash into, the lower the probability that we make a mistake. Thus, a natural approach for our problem would be to vary the number of hashed locations of x based on p_x and q_x. Indeed, this is the question investigated by Bruck, Gao and Jiang [12] in their Weighted Bloom filter design. More precisely, let k_x denote the number of locations that key x is hashed to. Then [12] investigates the optimal choice of the parameters k_x that limits the false positive rate in expectation over both the input and the query distribution. Their approach follows the original Bloom filter analysis and casts the problem as an unconstrained optimization problem in which k_x is allowed to be any real number (including negative). For more details, we refer the reader to Section 1.2 This formulation and the fact that their false positive rate is taken as an average over $\mathcal{P}_n$ leads to situations in which k_x can be made arbitrarily large and, with high probability, the filter is filled with 1s and has a high false positive probability (for instance, when a key is queried very rarely). To avoid such situations, as we shall see next, optimal choices for k_x exhibit some rather counter-intuitive trade-offs between p_x and q_x.

1.1 Our Contributions

We start by discussing near-optimal choices for k_x for a Weighted Bloom filter that is a $(\mathcal{Q}, \varepsilon)$-filter for sets drawn from $\mathcal{P}_n$. While this filter is not the most efficient of the filters we construct, reasoning through it helps us present our parametrizations and addresses the fact that Bloom filters remain well-liked in practice [35]. Specifically, we define k_x as follows:[3]

$$k_x \triangleq \begin{cases} 0 & \text{if or } p_x > 1/n \text{ or } q_x \le \varepsilon p_x \ , \\ \log(1/\varepsilon \cdot q_x/p_x) & \text{if } \varepsilon p_x < q_x \le \min\{p_x, \varepsilon/n\} \ , \\ \log(1/\varepsilon) & \text{if } q_x > p_x \text{ and } p_x \le \varepsilon/n \ , \\ \log(1/(np_x)) & \text{if } q_x > \varepsilon/n \text{ and } \varepsilon/n < p_x \le 1/n \ . \end{cases}$$

The first case covers the situation in which x is very likely to be included in the set or is queried very rarely (relative to p_x). Intuitively, it makes sense in these cases to always say YES when queried. Thus, we set $k_x = 0$ and store no information about these keys. Conversely, the third case considers the case in which x is queried so often (relative to p_x) that we need to explicitly keep the false positive probability below ε, which is achieved by setting $k_x = \log(1/\varepsilon)$. This is the largest number of hash functions we employ for any key, so in this sense, we are never worse than the classical Bloom filter. The second case interpolates smoothly between the first and third cases for elements that are rarely (but not very rarely) queried (compared to how likely they are to be inserted). Finally, the fourth case interpolates between the first and third case for elements that are not too rarely queried, in which case the precise query probability does not matter. See Figure 1 for a visualization of these regimes.

To further make sense of these regimes, we consider the case of uniform queries, i.e., $q_x = 1/u$, and assume that $\varepsilon > n/u$, a standard assumption in filter design (otherwise, the filter would essentially have to answer correctly on all queries and the lower bound of $n \log_2(1/\varepsilon) - O(1)$ would not hold [15,20]). Then in the two extremes, we would set $k_x = 0$ for elements with $p_x \ge 1/(u\varepsilon)$ (first case) and $k_x = \log(1/\varepsilon)$ when $p_x \le 1/u$ (third case). Keys

[3] Throughout the paper, we employ $\log x$ to denote $\log_2 x$ and $\ln x$ to denote $\log_e x$.

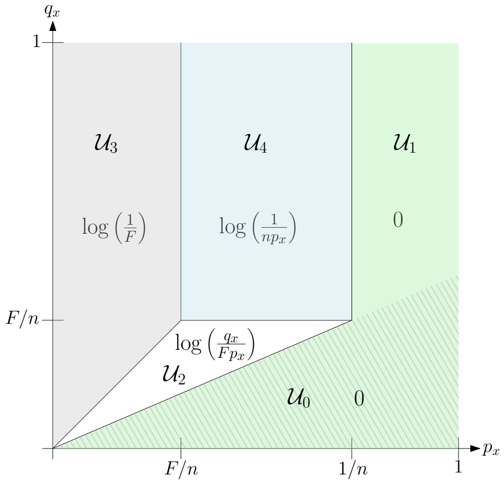


Figure 1 A schematic visualization of the different regimes for k_x.

with p_x between the two cases would exhibit the smooth interpolation $k_x = \log(1/(u\varepsilon) \cdot 1/p_x)$, corresponding to the intuition that the more likely an element is to be inserted, the less information we should store about it (i.e., smaller k_x).

The lower bound. Given the above parameters, we then define the quantity

$$\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) \triangleq \sum_{x \in \mathcal{U}} p_x k_x$$

and show that, perhaps surprisingly, it gives a lower bound for the expected space that any $(\mathcal{Q}, \varepsilon)$-filter requires when the input set is drawn from $\mathcal{P}_n$:

▶ **Theorem 2** (Lower bound - simplified). *Let A be an algorithm and assume that for any input set $S \subseteq \mathcal{U}$ with $|S| \leq n$, $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for S. Then the expected size of $A(S)$ must satisfy*

$$\mathbb{E}_{\mathcal{P}_n, A}\left[|A(S)|\right] \geq LB(\mathcal{P}_n, \mathcal{Q}, \varepsilon) - 1 - 6n\ ,$$

where S is sampled with respect to $\mathcal{P}_n$ and the queries are sampled with respect to $\mathcal{Q}$.

Previous approaches for filter lower bounds show that there exists a set $S \subseteq U$ of size n for which the filter needs to use $n \log_2(1/\varepsilon) - O(1)$ bits [15, 20]. This type of lower bound is still true in our model but it does not necessarily say anything meaningful, since the bad set S could be sampled in $\mathcal{P}_n$ with a negligible probability. Indeed, if we were to ignore the input distribution, then we would not be able to beat the worst input distribution and, in particular, we would need to use at least $\sup_{\mathcal{P}} \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) = \mathsf{LB}(\mathcal{Q}_n, \mathcal{Q}, \varepsilon) = n \log(1/\varepsilon)$ bits in expectation, where $\mathcal{Q}_n$ denotes a distribution over n independent draws from $\mathcal{Q}$.

In our model, it is therefore more natural to lower bound the *expected* size of the filter over the randomness of the input set. Finally, we remark that the full lower bound we prove is slightly stronger in that it holds for all but an unlikely collection of possible input sets, i.e. we only require that $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for $S \in \mathcal{T}$ where $\mathcal{T} \subseteq \mathbb{P}(U)$ and $\Pr_{\mathcal{P}_n}\left[S \notin \mathcal{T}\right] \leq \frac{1}{\log u}$ (see Theorem 6).

The space-efficient filter. We also show a filter design that asymptotically matches our space lower bound and executes operations in constant time in the worst case:

▶ **Theorem 3** (Space-efficient filter – simplified). *Given $0 < \varepsilon < 1$, there is a $(\mathcal{Q}, \varepsilon)$-filter with the following guarantees:*

- *it is a $(\mathcal{Q}, \varepsilon)$-filter with high probability over sets drawn from $\mathcal{P}_n$, if $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$,*
- *queries and insertions take constant time in the worst case,*
- *the space it requires is $(1 + o_n(1)) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits.*

The construction uses the k_x values from above in conjunction with the fingerprinting technique of Carter *et al.* [15] to obtain results that are comparable to state-of-the-art (classic) filter implementations that execute all operations (queries and insertions) in worst case constant time, and are space efficient, in the sense in which they require $(1 + o_n(1)) \cdot n \log(1/\varepsilon) + O(n)$ bits [1, 4–6]. The condition that $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$ can be seen as a generalization of the standard filter assumption that $\varepsilon \geq n/u$.

The Daisy Bloom filter. For completeness, we also present our variant of the Weighted Bloom filter, which we call the *Daisy Bloom filter*:[4]

▶ **Theorem 4** (Daisy Bloom filter – simplified). *Given $0 < \varepsilon < 1$, the Daisy Bloom filter has the following guarantees:*

- *it is a $(\mathcal{Q}, \varepsilon)$-filter with high probability over sets drawn from $\mathcal{P}_n$, if $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$,*
- *queries and insertions take at most $\lceil \log_2(1/\varepsilon) \rceil$ time in the worst case,*
- *the space it requires is $\log(e) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits.*

In contrast to the weighted Bloom filters of Bruck et al. [12], the Daisy Bloom filter executes operations in time that is at most $\lceil \log_2(1/\varepsilon) \rceil$ in the worst case (versus arbitrarily large) and achieves a false positive rate of at most ε with high probability over the input set (and not just on average). We also depart in our analysis from their unconstrained optimization approach (to setting k_x) and instead use Bernstein's inequality to argue that, if the length of the array is set to $\log(e) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits, then whp, at most half of the entries in the array will be set to 1 (see Section 5 for more details).

1.2 Related Work

Filters have been studied extensively in the literature [2, 5, 6, 15, 20, 34, 41, 42, 45], with Bloom filters perhaps the most widely employed variants in practice [35]. Learning-based approaches to classic algorithm design have recently attracted a great deal of attention, see e.g. [21, 31–33, 46]. For a comprehensive survey on learned data structures, we refer the reader to Ferragina and Vinciguerra [29].

Weighted Bloom Filters

Given information about the probability of inserting and querying each element, Bruck, Gao and Jiang [12] set out to find an optimal choice of the parameters k_x that limit the false positive rate (in expectation over both the input and the query distribution). The approach

[4] The daisy is one of our favorite flowers, especially when in full bloom, and is also a subsequence of "*d*yn*a*m*i*c *s*trech*y*" which describes the key properties of our data structure. It is also the nickname of the Danish queen, whose residence is not far from the place where this work was conceived. Daisy Bloom filters are not related to any celebrities.

is to solve an unconstrained optimization problem where the variables k_x can be any real number. In a post-processing step each k_x is rounded to the nearest non-negative integer. Unfortunately, this process does not lead to an optimal choice of parameters, and in fact, does not guarantee a non-trivial false positive rate. The issue is that the solution to the unconstrained problem may have many negative values of k_x, so even though the weighted sum $\sum_x p_x k_x$ is bounded, the post-processed sum $\sum_x p_x \max(k_x, 0)$ can be arbitrarily large. In particular, this is the case if at least one element is queried very rarely. This means that the weighted Bloom filter may consist only of 1s with high probability, resulting in a false positive probability of 1.

The above issue was noted by Wang, Ji, Dang, Zheng and Zhao [49] who attempt to correct the values for k_x, but their analysis still suffers from the same, more fundamental, problem: the existence of a very rare query element drives the false positive rate to 1. Wang et al. [49] also show an information-theoretical “approximate lower bound” on the number of bits needed for a weighted Bloom filter with given distributions $\mathcal{P}$ and $\mathcal{Q}$. The sense in which the lower bound is approximate is not made precise, and the lower bound is certainly not tight (for example, it can be negative).

Partitioned Learned Bloom Filters

There are several learned Bloom filter designs that assume that the filter has access to a learned model of the input set [18, 33, 37, 48]. The model is given a fixed input set S and a representative sample of elements in $\mathcal{U} \setminus S$ (the query distribution is not specified). Given a query element x, the model returns a *score* $s(x) \in [0, 1]$, which can be intuitively thought of as the model’s belief that $x \in S$. Based on this score, Vaidya, Knorr, Mitzenmacher and Kraska [48] choose a fixed number of k thresholds, partition the elements according to these thresholds, and build separate Bloom filters for each set of the partition. For fixed threshold values, they then formulate the optimization problem of setting the false positive rates f_i such that the total space of the data structure is minimized and the overall false positive rate is at most a given F.

As noted by Ferragina and Vinciguerra [29], a significant drawback in these constructions is that the guarantees they provide depend significantly on the query set given as input to the machine learning component and in particular, the set being representative for the whole query distribution. We avoid this issue by making the dependencies on q_x explicit and by bounding the average false positive probability even when just one element is queried. In addition, our data structure does not need to know the set S in advance (and hence, training can be done just once, in a pre-processing phase), employs only one data structure, and our guarantees are robust to approximate values for p_x and q_x.

1.3 Paper Organization

After some preliminaries, Section 3 shows our lower bound on the space usage. In Section 4, we discuss a space-efficient filter with constant time worst-case operations. Finally, Section 5 presents the analysis of the Daisy Bloom filter.

2 Preliminaries

For clarity, throughout the paper, we will distinguish between probabilities over the randomness of the input set, denoted by $\Pr_{\mathcal{P}_n}[\cdot]$, and probabilities over the internal randomness of the filter, denoted by $\Pr_A[\cdot]$. Joint probabilities are denoted by $\Pr_{\mathcal{P}_n, A}[\cdot]$. For the analysis, it will also make sense to partition the universe $\mathcal{U}$ into the following 5 parts:

$$\begin{aligned}
&\mathcal{U}_0 \triangleq \{x \in \mathcal{U} \mid q_x \le \varepsilon p_x\} \ , \\
&\mathcal{U}_1 \triangleq \{x \in \mathcal{U} \mid q_x > \varepsilon p_x \text{ and } p_x > 1/n\} \ , \\
&\mathcal{U}_2 \triangleq \{x \in \mathcal{U} \mid \varepsilon p_x < q_x \le \min\{p_x, \varepsilon/n\}\} \ , \\
&\mathcal{U}_3 \triangleq \{x \in \mathcal{U} \mid q_x > p_x \text{ and } \varepsilon/n \ge p_x\} \ , \\
&\mathcal{U}_4 \triangleq \{x \in \mathcal{U} \mid q_x > \varepsilon/n \text{ and } \varepsilon/n < p_x \le 1/n\} \ .
\end{aligned}$$

The high probability guarantees we obtain increase with $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$. Therefore, such bounds are meaningful for distributions in which the optimal size $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ of a filter is not too small. Similarly, we can assume that the size of the universe is polynomial in n, and so $\log(1/\varepsilon) = O(\log n)$ in the standard case in which $\varepsilon > n/|\mathcal{U}|$. Therefore, while in general $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ can be much smaller than $n \log_2(1/\varepsilon)$, we do require some mild dependency on n for the high probability bounds to be meaningful. Finally, we recall the following classic result in data compression:

▶ **Theorem 5** (Kraft's inequality [47]). *For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths $\ell_1, \ell_2, \dots, \ell_m$ must satisfy the inequality*

$$\sum_i D^{-\ell_i} \le 1 \ .$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

3 The Lower Bound

The goal of this section is to prove the lower bound from Theorem 2. As discussed, we prove a slightly stronger statement where we allow our algorithm to not produce a $(\mathcal{Q}, \varepsilon)$-filter for some input sets as long as the probability of sampling them is low. Formally, we show that:

▶ **Theorem 6.** *Let $\mathcal{T} \subseteq \mathbb{P}(\mathcal{U})$ be given such that $\Pr_{\mathcal{P}_n}[S \notin \mathcal{T}] \le \frac{1}{\log u}$. If A is an algorithm such that for all $S \in \mathcal{T}$, $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for S. Then the expected size of $A(S)$ must satisfy*

$$\mathbb{E}_{\mathcal{P}_n, A}[|A(S)|] \ge \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) - 1 - 6n \ ,$$

where S is sampled with respect to $\mathcal{P}_n$.

Proof. Each instance $\mathcal{I}$ of the data structure corresponds to a subset $\mathcal{U}_{\mathcal{I}} \subset \mathcal{U}$ on which the data structure answers YES. We denote the number of bits needed by such an instance by $|\mathcal{I}|$. For any set $S \in \mathcal{T}$, we have that $\mathcal{I} = A(S)$ satisfies that $S \subseteq \mathcal{U}_{\mathcal{I}}$ and

$$\mathbb{E}_A\left[\sum_{x \in \mathcal{U}_{\mathcal{I}} \setminus S} q_x\right] \le \varepsilon \ .$$

The goal is to prove that

$$\mathbb{E}_{\mathcal{P}_n, A}[|A(S)|] \ge n \cdot \left(\sum_{x \in \mathcal{U}_2} p_x \log\left(\frac{1}{\varepsilon} \cdot \frac{q_x}{p_x}\right) + \sum_{x \in \mathcal{U}_3} p_x \log\frac{1}{\varepsilon} + \sum_{x \in \mathcal{U}_4} p_x \log\frac{1}{n p_x}\right) - 1 - 6n \ .$$

We will lower bound $\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right]$ by using it to encode an ordered sequence of n elements drawn according to $\mathcal{P}_n$. Specifically, for any ordered sequence of n elements $\hat{S} \in \mathcal{U}^n$, we let $S \subseteq U$ be the set of distinct elements and let $\mathcal{I} = A(S)$ as above. We first note that to encode $\hat{S} \sim \mathcal{P}_n$, in expectation, we need at least the entropy number of bits, i.e.,

$$n \sum_{x \in \mathcal{U}} p_x \log \frac{1}{p_x} \; . \tag{1}$$

Now our encoding using $\mathcal{I}$ will depend on whether $S \in \mathcal{T}$ or not. First, we will use 1 bit to describe whether $S \in \mathcal{T}$ or not. For $(x_i)_{i \in [n]} \in \hat{S}$, we will denote b_i to be the number bits to encode x_i. If $S \notin \mathcal{T}$ then for all $i \in [n]$ we encode x_i using $b_i = \lceil \log(1/p_{x_i}) \rceil$ bits. If $S \in \mathcal{T}$ then for all $i \in [n]$ we encode x_i depending on which subset if $\mathcal{U}$ it belongs to:

1. If $x_i \in \mathcal{U}_0 \cup \mathcal{U}_1$, we encode x_i using $b_i = \lceil \log(4/p_{x_i}) \rceil$ bits.
2. If $x_i \in \mathcal{U}_2$, we encode x_i using $b_i = \left\lceil \log\left(4 \frac{\sum_{y \in \mathcal{U}_\mathcal{I} \cap \mathcal{U}_2} q_y}{q_{x_i}}\right)\right\rceil$ bits.
3. If $x_i \in \mathcal{U}_3$, we encode x_i using $b_i = \left\lceil \log\left(4 \frac{\sum_{y \in \mathcal{U}_\mathcal{I} \cap \mathcal{U}_3} p_y}{p_{x_i}}\right)\right\rceil$ bits.
4. If $x_i \in \mathcal{U}_4$, we encode x_i using $b_i = \lceil \log\left(4\, |\mathcal{U}_\mathcal{I} \cap \mathcal{U}_4|\right) \rceil$ bits.

It is clear from the construction that we satisfy the requirement for Theorem 5 thus there exists such an encoding. Now we will bound the expectation of the size of this encoding:

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)| + 1 + \sum_{i \in [n]} b_i\right] = \mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] + 1 + \mathbb{E}_{\mathcal{P}_n,A}\left[\sum_{i \in [n]} b_i\right] \; .$$

We will write $\mathbb{E}_{\mathcal{P}_n,A}\left[\sum_{i \in [n]} b_i\right] = \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \sum_{i \in [n]} b_i\right] + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \notin \mathcal{T}] \sum_{i \in [n]} b_i\right]$, and bound each term separately.

We start by bounding $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \notin \mathcal{T}] \sum_{i \in [n]} b_i\right]$.

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}_n,A}\left[[S \notin \mathcal{T}] \sum_{i \in [n]} b_i\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \notin \mathcal{T}] \sum_{i \in [n]} \lceil \log(1/p_{x_i}) \rceil\right] \\
&\leq \Pr_{\mathcal{P}_n}\left[S \notin \mathcal{T}\right] n + \mathbb{E}_{\mathcal{P}_n}\left[[S \notin \mathcal{T}] \log\left(\prod_{i \in [n]} 1/p_{x_i}\right)\right] \\
&= \Pr_{\mathcal{P}_n}\left[S \notin \mathcal{T}\right] n + \sum_{\hat{s} \in \mathcal{U}^n} [\hat{s} \in \mathcal{T}] \Pr_{\mathcal{P}_n}\left[\hat{S} = \hat{s}\right] \log \frac{1}{\Pr_{\mathcal{P}_n}\left[\hat{S} = \hat{s}\right]}
\end{aligned}$$

Now using Jensen's inequality we get that

$$\sum_{\hat{s} \in \mathcal{U}^n} [\hat{s} \in \mathcal{T}] \Pr_{\mathcal{P}_n}\left[\hat{S} = \hat{s}\right] \log \frac{1}{\Pr_{\mathcal{P}_n}\left[\hat{S} = \hat{s}\right]} \leq \Pr_{\mathcal{P}_n}\left[S \notin T\right] \log\left(\frac{1}{\Pr_{\mathcal{P}_n}[S \notin T] u^n}\right) \; .$$

Putting this together with the fact that $\Pr_{\mathcal{P}_n}\left[S \notin T\right] \leq \frac{1}{\log u}$, we get that,

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S \notin \mathcal{T}] \sum_{i \in [n]} b_i\right] \leq 2n \; .$$

Now we bound $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \sum_{i\in[n]} b_i\right] = \sum_{i\in[n]} \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right]$. We will bound $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right]$ depending on which subset of $\mathcal{U}$ that x_i belongs to.

If $x_i \in \mathcal{U}_0 \cup \mathcal{U}_1$, then we have that $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le \lceil \log(4/p_{x_i}) \rceil \le 3 + \log(1/p_{x_i})$.

If $x_i \in \mathcal{U}_2$, define $Z_2 = \mathcal{U}_{\mathcal{I}} \cap \mathcal{U}_2$. Then

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right] .$$

We know that $\sum_{y\in S\cap\mathcal{U}_2} q_y \le \varepsilon$ since $q_y \le \varepsilon/n$ for all $y \in \mathcal{U}_2$ and $|S| \le n$. We also know that $\mathbb{E}_A\left[\sum_{x\in Z_2\setminus S} q_x\right] \le \varepsilon$ for $S \in \mathcal{T}$. Now using Jensen's inequality we get that

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\, \mathbb{E}_A\left[\log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right]\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(\frac{\mathbb{E}_A\left[\sum_{y\in Z_2} q_y\right]}{q_{x_i}}\right)\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(\frac{2\varepsilon}{q_{x_i}}\right)\right] \le 1 + \mathbb{E}_{\mathcal{P}_n}\left[\log\left(\frac{\varepsilon}{q_{x_i}}\right)\right] .
\end{aligned}$$

If $x_i \in \mathcal{U}_3$, define $Z_3 = \mathcal{U}_{\mathcal{I}} \cap \mathcal{U}_3$. Then

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right] .$$

We know that $\sum_{y\in S\cap\mathcal{U}_3} p_y \le \varepsilon$ since $p_y \le \varepsilon/n$ for all $y \in \mathcal{U}_3$ and $|S| \le n$. We also know that $\mathbb{E}_A\left[\sum_{x\in Z_3\setminus S} p_x\right] \le \mathbb{E}_A\left[\sum_{x\in Z_3\setminus S} q_x\right] \le \varepsilon$ for $S \in \mathcal{T}$. Using Jensen's inequality we get that,

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\, \mathbb{E}_A\left[\log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right]\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(\frac{\mathbb{E}_A\left[\sum_{y\in Z_3} p_y\right]}{p_{x_i}}\right)\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(\frac{2\varepsilon}{p_{x_i}}\right)\right] \le 1 + \mathbb{E}_{\mathcal{P}_n}\left[\log\left(\frac{\varepsilon}{p_{x_i}}\right)\right] .
\end{aligned}$$

If $x_i \in \mathcal{U}_4$, define $Z_4 = \mathcal{U}_{\mathcal{I}} \cap \mathcal{U}_4$. Then $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(|Z_4|\right)\right]$. We know that $|Z_4| = |Z_4 \cap S| + |Z_4 \setminus S| \le n + \frac{n}{\varepsilon}\sum_{x\in Z_4\setminus S} q_x$ since $q_y > \varepsilon/n$ for all $y \in \mathcal{U}_4$ and $|S| \le n$. Using Jensen's inequality we get that, for $Z_4' = Z_4 \setminus S$:

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}] \log\left(n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\, \mathbb{E}_A\left[\log\left(n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right)\right]\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(\mathbb{E}_A\left[n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right]\right)\right] \\
&\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}] \log\left(2n\right)\right] \le 1 + \log(n) .
\end{aligned}$$

Combining it all we get an encoding that in expectation uses at most

$$\begin{aligned}
&\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] + 1 + 6n + \\
&\qquad \sum_{x\in(\mathcal{U}_0\cup\mathcal{U}_1)} p_x \log(1/p_x) + \sum_{x\in\mathcal{U}_2} p_x \log(\varepsilon/q_x) + \sum_{x\in\mathcal{U}_3} p_x \log(\varepsilon/p_x) + \sum_{x\in\mathcal{U}_4} p_x \log n .
\end{aligned}$$

bits to encode $\hat{S}$. Comparing this with Equation (1) we get that,

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] \geq n \cdot \left(\sum_{x\in\mathcal{U}_2} p_x \log\left(\frac{1}{\varepsilon}\cdot\frac{q_x}{p_x}\right) + \sum_{x\in\mathcal{U}_3} p_x \log\frac{1}{\varepsilon} + \sum_{x\in\mathcal{U}_4} p_x \frac{1}{np_x} \right) - 1 - 6n\ .$$

This proves the claim. ◀

4 Space-Efficient Filter

In this section, we show how one can use the k_x values proposed to design a space-efficient $\mathcal{Q}$-filter with worst-case constant time operations. Formally, we show that:

▶ **Theorem 7.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \leq \varepsilon$. Then there exists a filter with the following guarantees:*

- *there exists $\mathcal{T} \subseteq \mathbb{P}(U)$ where a set $S \in \mathcal{T}$ with high probability over the randomness of $\mathcal{P}_n$, such that the filter is a $(\mathcal{Q},\varepsilon)$-filter for any $S \in \mathcal{T}$,*
- *the filter uses $(1 + o_n(1)) \cdot LB(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits,*
- *the filter executes queries and insertions in worst case constant time and,*
- *the filter does not fail with high probability over its internal randomness.*

4.1 Construction

For $j \in \{1,\ldots,\lceil\log(1/\varepsilon)\rceil\}$, we let $\mathcal{U}^{(j)} \triangleq \{x \in \mathcal{U} \mid \lceil k_x\rceil = j\}$ denote the set of elements that hash to j locations in the Daisy Bloom filter and $P_j \triangleq \sum_{x\in\mathcal{U}^{(j)}} p_x$ denote the probability that we select an element from $\mathcal{U}^{(j)}$ in one sample from $\mathcal{P}$. Then $n_j \triangleq n \cdot P_j$ denotes the average number of elements from $\mathcal{U}^{(j)}$ that we expect to see in the input set. We distinguish between the sets $\{\mathcal{U}^{(j)}\}$ depending on their corresponding n_j. Specifically, we say that $\mathcal{U}^{(j)}$ is a *rare class* if $n_j < n/\log^c n$, for some constant $c > 2$, and otherwise we say that $\mathcal{U}^{(j)}$ is a *frequent class*. We further define $\mathcal{U}_r \subseteq \mathcal{U}$ to be the set of all elements that are in a rare class, i.e., $\mathcal{U}^{(j)} \subseteq \mathcal{U}$ if and only if $\mathcal{U}^{(j)}$ is a rare class.

Now let $\mathcal{F}(\varepsilon, n)$ be a (standard) filter implementation for at most n elements with false positive probability at most ε. We focus on implementations that execute all operations (queries and insertions) in worst-case constant time, and are space efficient: they require $(1 + o_n(1)) \cdot n\log(1/\varepsilon) + O(n)$ bits [1, 4–6]. We employ $\lceil\log(1/\varepsilon)\rceil + 1$ instantiations of $\mathcal{F}$, which we denote by $\mathcal{F}_1,\ldots,\mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ and $\mathcal{F}_r$. They are parametrized as follows: for $j \in \{1,\ldots,\lceil\log(1/\varepsilon)\rceil\}$, we further define $N_j \triangleq (1+1/\log n)\cdot n_j$ and instantiate $\mathcal{F}_j = \mathcal{F}(2^{-j}, N_j)$. We instantiate $\mathcal{F}_r$ as $\mathcal{F}_r = \mathcal{F}(F, N_r)$, where $N_r \triangleq \Theta(n/\log^{c-1} n)$.

Operations. We distinguish between elements that are in a frequent class and elements that are in a rare class. If an element is in a frequent class $\mathcal{U}^{(j)}$, then operations are forwarded to the corresponding filter $\mathcal{F}_j$. Otherwise, the operation is forwarded to $\mathcal{F}_r$. Since all the filters we employ perform operations in constant time in the worst case, the same holds for our construction[5].

[5] We note here that it is possible to combine the filters $\mathcal{F}_1,\ldots,\mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ into one single data structure. One could use, for example, the balls-into-bins implementation in [5], where elements are randomly assigned to one of $n/\Theta(\log n/(\log(1/\varepsilon)))$ buckets and the buckets explicitly store random strings of length $\log(1/\varepsilon)$ associated with the elements that hash into them. Combining $\mathcal{F}_1,\ldots,\mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ would then entail "superimposing" their buckets.

Space. We now bound the total number of bits that $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ and $\mathcal{F}_r$ require:

▶ **Lemma 8.** *The above filter requires* $(1 + o_n(1)) \cdot \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ *bits.*

Proof. Recall that $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ and $\mathcal{F}_r$ are instantiations of a filter $\mathcal{F}$ which requires $(1+f(n)) \cdot n \log(1/\varepsilon) + O(n)$ bits for a set of n elements and false positive probability ε, where the function $f(n)$ satisfies $f(n) = o_n(1)$ [1,4–6]. For simplicity, we choose the implementation in [5], where $f(n) = \Theta(\log\log n/\sqrt{\log n})$. Consequently, for $j \in \{1, \ldots, \lceil \log(1/\varepsilon) \rceil\}$, the space of $\mathcal{F}_j$ is:

$$(1 + f(N_j)) \cdot N_j \log(1/2^{-j}) + O(N_j) = (1 + f(N_j)) \cdot N_j \cdot j + O(N_j)$$

bits. Since $\mathcal{F}_j$ is instantiated only for frequent classes, it follows that $n \geq N_j \geq n/\log^c n$ and hence, $f(N_j) = \Theta(f(n))$ for all j with $\mathcal{U}^{(j)}$ a frequent class. Furthermore, by definition, we have that $j \leq k_x + 1$ for all $x \in \mathcal{U}^{(j)}$ and $N_j = (1 + 1/\log n) \cdot n_j = (1 + 1/\log n) \cdot n \sum_{x \in \mathcal{U}^{(j)}} p_x$. Therefore, the space that $\mathcal{F}_j$ requires can be upper bounded by

$$(1 + \Theta(f(n))) \cdot n \sum_{x \in \mathcal{U}^{(j)}} p_x k_x + O(N_j) \ .$$

Since $\sum_j N_j = (1 + 1/\log n) \cdot n$ we get that, in the worst case in which all the classes are frequent, the filters $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ require

$$(1 + o_n(1)) \cdot n \sum_{x \in \mathcal{U}} p_x k_x + O(n) = (1 + o_n(1)) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$$

bits. The space of the final filter $\mathcal{F}_r$ is upper bounded by $\Theta(n/\log^{c-1} n) \cdot \log(1/\varepsilon) = \Theta(n/\log^{c-2} n) = o(n)$ bits for any constant $c > 2$. The claim follows. ◀

4.2 Analysis

In this section, we show that the filter described above does not fail whp and that it achieves a false positive probability of at most 3ε with respect to $\mathcal{Q}$. In our construction, there are two sources of failure: when the number of elements which we insert into each filter exceeds the maximum capacity of the filter, and when the filters themselves fail as a consequence of their internal randomness. In the latter case, we note that the failure probability of $\mathcal{F}(\varepsilon, n)$ is guaranteed to be at most $1/\mathsf{poly}(n)$, where the degree of the polynomial is a constant of our choosing [1,4–6]. Since all of the instantiations we employ have maximum capacities which are $\Omega(n/\mathsf{polylog}(n))$, we conclude that each of these separate instantiations also fails with probability at most $1/\mathsf{poly}(n)$, and therefore, by a union bound over the $\lceil \log(1/\varepsilon) \rceil + 1 = O(\log n)$ instantiations, we get that some filter fails with probability at most $1/\mathsf{poly}(n)$. We now show that the maximum capacities we set for each filter suffice.

▶ **Lemma 9.** *Whp, at most* $N_r = \Theta(n/\log^{c-1} n)$ *elements from* $\mathcal{U}_r$ *are sampled in the input set.*

Proof. Let $\mathcal{U}^{(j)}$ be a rare class and let X_j denote the number of elements from $\mathcal{U}^{(j)}$ that we sample in the input set. By definition, the expected number of elements we see from $\mathcal{U}^{(j)}$ satisfies $\mathbb{E}_{\mathcal{P}}[X_j] = n_j < n/\log^c n$, for some constant $c > 2$. By Chernoff bound, we therefore get that:

$$\Pr[X_j > 6n/\log^c n] \leq 2^{-6n/\log^c n} \ .$$

There are at most $\lceil \log(1/\varepsilon) \rceil = O(\log n)$ possible rare classes, and so, by the union bound, the number of elements from $\mathcal{U}_r$ that we sample in the input set is at most $N_r = \Theta(n/\log^{c-1} n)$ whp. ◀

We now focus on sampling elements from a frequent class:

▶ **Lemma 10.** *Let $\mathcal{U}^{(j)}$ be a frequent class. Then, whp, at most N_j elements from $\mathcal{U}^{(j)}$ are sampled in the input set.*

Proof. Let X_j denote the number of elements from $\mathcal{U}^{(j)}$ that we sample in the input set and note that $\mathbb{E}_{\mathcal{P}}[X_j] = n_j \geq n/\log^c n$. By Chernoff:

$$\Pr[X_j > N_j] = \Pr[X_j > (1 + 1/\log n) \cdot n_j] \leq \exp(-\Theta(n/\log^{c-2} n)) \leq 1/\mathsf{poly}\, n \ .$$

This concludes our proof. ◀

Finally, we bound the false positive rate of the filter:

▶ **Lemma 11.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n \sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon$ and that the input set S does not make $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ and $\mathcal{F}_r$ fail. Then the filter described is a $(\mathcal{Q}, 3\varepsilon)$-filter on S.*

Proof. Fix an input set S and denote by $A'(S,x)$ the output of the filter when queried for an element x. We are interested in bounding $\Pr[A'(S,x) = \mathsf{YES}]$ for an element $x \notin S$. If $x \in \mathcal{U}_r$, then we forward the query operation to $\mathcal{F}_r$, which guarantees that $\Pr[A'(S,x) = \mathsf{YES}] \leq \varepsilon$. Therefore:

$$\sum_{x \in \mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x \in \mathcal{U}_r} q_x \cdot \varepsilon \ ,$$

Otherwise, if $x \notin \mathcal{U}_r$, the query is forwarded to $\mathcal{F}_j$, where $j = \lceil k_x \rceil$. In this case, $\Pr[A'(S,x) = \mathsf{YES}] \leq 2^{-j} \leq 2^{-k_x}$ and we get that

$$\sum_{x \in \mathcal{U}_0 \cup \mathcal{U}_2 \setminus \mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x \in \mathcal{U}_0 \setminus \mathcal{U}_r} q_x + \sum_{x \in \mathcal{U}_2 \setminus \mathcal{U}_r} p_x \varepsilon \leq \sum_{x \in \mathcal{U}_0 \cup \mathcal{U}_2 \setminus \mathcal{U}_r} p_x \cdot \varepsilon \ ,$$

and similarly,

$$\sum_{x \in \mathcal{U}_1 \cup \mathcal{U}_4 \setminus \mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x \in \mathcal{U}_1 \setminus \mathcal{U}_r} q_x + \sum_{x \in \mathcal{U}_4 \setminus \mathcal{U}_r} n p_x q_x \leq \sum_{x \in \mathcal{U}_1 \cup \mathcal{U}_4 \setminus \mathcal{U}_r} n p_x q_x \ .$$

Finally, we have that

$$\sum_{x \in \mathcal{U}_3 \setminus \mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x \in \mathcal{U}_3 \setminus \mathcal{U}_r} q_x \cdot \varepsilon \ .$$

Adding all of these quantities, we obtain the claim. ◀

4.3 Remarks

The filter construction assumes that we know, in advance, whether a class $\mathcal{U}^{(j)}$ is frequent and, if so, what is the value of its corresponding $P_j = \sum_{x \in \mathcal{U}^{(j)}} p_x$. This is because we employ fixed capacity filters which require us to provide an upper bound on the cardinality of the input set $S \cap \mathcal{U}^{(j)}$ at all points in time. We note that this assumption can be alleviated in two ways: on one hand, one can employ filters that do not require us to know the size of the input set in advance [8, 42]. This would incur an additional $\Theta(n \log \log n)$ bits in the space consumption of our filter (operations would remain constant time worst case).

On the other hand, one can estimate P_j for all frequent classes $\mathcal{U}^{(j)}$ if we are allowed to take $\mathsf{polylog}(n)$ samples from $\mathcal{P}$ before constructing the filter. Specifically, fix $j \in \{1, \ldots, \lceil \log(1/\varepsilon) \rceil\}$ and take $\ell = O(\log^{2c} n)$ samples from $\mathcal{P}$. Define Z_j to be the number of sampled elements that are in $\mathcal{U}^{(j)}$. Then, if $\mathcal{U}^{(j)}$ is indeed frequent, by the standard Chernoff bound we get that, whp, $Z_j > \log^c n$ elements and Z_j/ℓ is an unbiased estimator for P_j with the guarantee that $P_j = (1 \pm O(1/\log^{(c-1)/2} n)) \cdot Z_j/\ell$ whp. Note that we can tolerate such an estimate since we set the maximum capacity of each filter to be $N_j = (1+1/\log n) \cdot nP_j$. A similar argument can be used for estimating the lower bound $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) = n \cdot \sum_{x \in \mathcal{U}} p_x \cdot k_x$ whp. Specifically, by the definition of P_j, we have that

$$\sum_{x \in \mathcal{U}} p_x \cdot \lceil k_x \rceil = \sum_{j=1}^{\lceil \log(1/\varepsilon) \rceil} j \cdot P_j \ .$$

If $\mathcal{U}^{(j)}$ is a frequent class, then by the above argument we have an estimate of its P_j. Otherwise, we know that $P_j < 1/\log^c n$ and, since $j \le \lceil \log(1/\varepsilon) \rceil = O(\log n)$, get that

$$\sum_{j \text{ s.t. } \mathcal{U}^{(j)} \in \mathcal{U}_r} j \cdot P_j \le \sum_{j \text{ s.t. } \mathcal{U}^{(j)} \in \mathcal{U}_r} \lceil \log(1/\varepsilon) \rceil \cdot 1/\log^c n \le 1/\log^{c-2} n \ ,$$

which contributes a $o(n)$ term to the lower bound.

5 The Daisy Bloom Filter Analysis

In this section, we analyse the behaviour of the Daisy Bloom filter with the values k_x denoting the number of hash functions that we use to hash x into the array. Let X_i denote the number of hash functions that are employed when we sample in the i^{th} round, i.e., $X_i = k_x$ with probability p_x. Then $X = \sum_i X_i$ denotes the number of locations that are set in the Bloom filter (where the same location might be set multiple times). Moreover, $\mathbb{E}_{\mathcal{P}_n}[X] = n \cdot \sum_{x \in \mathcal{U}} p_x k_x = \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$, since the $\{X_i\}_i$ are identically distributed. We then set the length m of the Daisy Bloom filter array to

$$m \triangleq \mathbb{E}_{\mathcal{P}_n}[X] \,/\, \ln 2 \ .$$

The remainder of this section is dedicated to proving the following statement:

▶ **Theorem 12.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n \sum_{x \in \mathcal{U}} p_x q_x \le \varepsilon$. Then there exists $\mathcal{T} \subseteq \mathbb{P}(U)$ such that $S \in \mathcal{T}$ with high probability over the randomness of $\mathcal{P}_n$, and for all $S \in \mathcal{T}$ the Daisy Bloom Filter is a $(\mathcal{Q}, \varepsilon)$-filter for S. The Daisy Bloom filter uses $\log(e) \cdot LB(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits and executes all operations in at most $\lceil \log(1/\varepsilon) \rceil$ time in the worst case.*

The sets in $\mathcal{T}$ are the sets for which $X \approx \mathbb{E}_{\mathcal{P}_n}[X] = \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$. The reason we constrain ourselves to these sets, is that if $X \gg \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ then most bits will be set to 1 which will make the false positive rate large. We will bound the probability that $X \gg \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ by using Bernstein's inequality and here, the following observation becomes crucial:

▶ **Observation 13.** *For every $x \in \mathcal{U}$, $k_x \le \log(1/\varepsilon)$.*

Proof. For $x \in \mathcal{U}_0 \cup \mathcal{U}_1$, we have that $k_x = 0$ which is clearly less than $\log(1/\varepsilon)$. For $x \in \mathcal{U}_3$ we have that $k_x = \log(1/\varepsilon)$ and again the statement holds trivially. For $x \in \mathcal{U}_2$, we have that $q_x \le p_x$ and so $k_x = \log(1/\varepsilon \cdot q_x/p_x) \le \log(1/\varepsilon)$. For $x \in \mathcal{U}_4$, we have that $p_x > F/n$ and so $k_x = \log(1/(np_x)) < \log(1/\varepsilon)$. ◀

We are now ready to prove that the random variable X is concentrated around its expectation.

▶ **Lemma 14.** *For any $\delta > 0$,*

$$\Pr_{\mathcal{P}_n}\left[X > (1+\tau)\cdot\mathbb{E}_{\mathcal{P}_n}[X]\right] \le \exp\left(-\frac{\tau^2 \ln 2}{2(1+\tau/3)}\cdot\frac{m}{\log(1/\varepsilon)}\right)$$

Proof. The random variables $\{X_i\}_i$ are independent and $X_i \le b \triangleq \log(1/\varepsilon)$ for all i by Observation 13. We apply Bernstein's inequality [22]:

$$\Pr_{\mathcal{P}_n}\left[X - \mathbb{E}_{\mathcal{P}_n}[X] > t\right] \le \exp\left(-\frac{t^2/2}{n\mathrm{Var}_{\mathcal{P}_n}[X_1] + bt/3}\right) .$$

Note that $\mathrm{Var}_{\mathcal{P}_n}[X_i] \le \mathbb{E}_{\mathcal{P}_n}\left[X_i^2\right] \le b\cdot\mathbb{E}_{\mathcal{P}_n}[X_i]$. Setting $t = \tau\cdot\mathbb{E}_{\mathcal{P}_n}[X] = \tau n\cdot\mathbb{E}_{\mathcal{P}_n}[X_1]$, we get that

$$\begin{aligned}\Pr_{\mathcal{P}_n}\left[X > (1+\tau)\mathbb{E}_{\mathcal{P}_n}[X]\right] &\le \exp\left(-\frac{\tau^2}{2}\cdot\frac{n^2(\mathbb{E}_{\mathcal{P}_n}[X_1])^2}{nb\cdot\mathbb{E}_{\mathcal{P}_n}[X_1] + \tau/3\cdot nb\cdot\mathbb{E}_{\mathcal{P}_n}[X_1]}\right)\\ &= \exp\left(-\frac{\tau^2}{2(1+\tau/3)}\cdot\frac{n\mathbb{E}_{\mathcal{P}_n}[X_1]}{b}\right) .\end{aligned}$$

The claim follows by noticing that $n\mathbb{E}_{\mathcal{P}_n}[X_1] = m\ln 2$. ◀

We can now prove that as long as $X \le (1 + 1/(2\log(1/\varepsilon)))\cdot\mathbb{E}_{\mathcal{P}_n}[X]$, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter for S. We consider the fraction ρ of entries in the array that are set to 0 after we have inserted the elements of the set. We then show that ρ is close to 1/2 with high probability over the input set and the randomness of the hash functions. Conditioned on this, we then have that the probability that we make a mistake for x is at most 2^{-k_x+1}. The false positive rate is then derived similarly to that of Lemma 11.

▶ **Lemma 15.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \le \varepsilon$, and that $X \le (1 + 1/(2\log(1/\varepsilon)))\mathbb{E}_{\mathcal{P}_n}[X]$. Then, whp, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter on S.*

Proof. Let $\rho \in [0,1]$ denote the fraction of entries in the Daisy Bloom filter that are set to 0 after we have inserted the elements of the set. Recall that the random variable X denotes the total number of entries that are set in the Bloom filter, including multiplicities. In the worst case, all the entries to the filter are distinct, and we have X independent chances to set a specific bit to 1. Therefore

$$\mathbb{E}_h[\rho|X] \ge \left(1-\frac{1}{m}\right)^X \approx e^{-X/m} = 2^{-X/\mathbb{E}_{\mathcal{P}_n}[X]} .$$

Moreover, by applying a Chernoff bound for negatively associated random variables, we have that for any $0 < \gamma < 1$,

$$\Pr_A\left[\rho \le (1-\gamma)\cdot\left(1-\frac{1}{m}\right)^X \middle| X\right] \le \exp\left(-m\left(1-\frac{1}{m}\right)^X\cdot\gamma^2/2\right) \tag{2}$$

We now let B_δ denote the event that $\left(1-\frac{1}{m}\right)^X > (1-\delta)\cdot\frac{1}{2}$ and B_γ the event that $\rho > (1-\gamma)\left(1-\frac{1}{m}\right)^X$. We then choose $0 < \delta < 1$ and $0 < \gamma < 1$ such that B_δ and $B_{\delta'}$ imply that

$$\rho \ge 1 - 2^{1/\log(1/\varepsilon)}\cdot\frac{1}{2} .$$

Moreover, for our choices of δ and γ, we have that both B_δ and $B_\gamma|B_\delta$ occur with high probability.[6] We refer the reader to the full version [7] for δ and γ. Conditioned on B_δ and B_γ, we get that, for an $x \notin S$, since $k_x \leq \log(1/\varepsilon)$,

$$\Pr_A\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] = (1-\rho)^{k_x} \leq 2^{k_x/b} \cdot 2^{-k_x} \leq 2 \cdot 2^{-k_x}$$

We bound the false positive rate on each partition. For $x \in \mathcal{U}_0$, i.e., with $q_x \leq Fp_x$ and $k_x = 0$, we can upper bound the false positive rate as such

$$\sum_{x\in\mathcal{U}_0} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \leq \sum_{x\in\mathcal{U}_0} q_x \leq \sum_{x\in\mathcal{U}_0} p_x \cdot \varepsilon\ .$$

For $x \in \mathcal{U}_1$ with $p_x > 1/n$ and $k_x = 0$, we have the following

$$\sum_{x\in\mathcal{U}_1} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \leq \sum_{x\in\mathcal{U}_1} q_x < n \sum_{x\in\mathcal{U}_1} p_x q_x\ .$$

For $x \in \mathcal{U}_2$ with $k_x = \log(1/\varepsilon \cdot q_x/p_x)$, we have the following

$$\begin{aligned}\sum_{x\in\mathcal{U}_2} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] &\leq \sum_{x\in\mathcal{U}_2} q_x \cdot 2 \cdot 2^{-k_x} = \sum_{x\in\mathcal{U}_2} q_x \cdot 2 \cdot p_x/q_x \cdot \varepsilon \\ &= \sum_{x\in\mathcal{U}_2} p_x \cdot 2\varepsilon\ .\end{aligned}$$

For $x \in \mathcal{U}_3$ with $k_x = \log(1/\varepsilon)$,

$$\sum_{x\in\mathcal{U}_3} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \leq \sum_{x\in\mathcal{U}_3} q_x \cdot 2\varepsilon \leq 2\varepsilon\ .$$

For $x \in \mathcal{U}_4$ with $k_x = \log(1/(np_x))$,

$$\sum_{x\in\mathcal{U}_4} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \leq \sum_{x\in\mathcal{U}_4} q_x \cdot 2np_x = 2n \sum_{x\in\mathcal{U}_4} p_x q_x\ .$$

For the overall false positive rate, note that the total false positive rate in $\mathcal{U}_0$ and $\mathcal{U}_2$ is at most 2ε. Similarly for the false positive rate in $\mathcal{U}_3$. For the remaining partitions $\mathcal{U}_1$ and $\mathcal{U}_4$, we have that it is at most

$$2n \sum_{x\in\mathcal{U}_1\cup\mathcal{U}_4} p_x q_x\ .$$

From our assumption, this later term is at most 2ε as well. ◀

Combining the above with Lemma 14 we get that with probability $1 - \exp\left(-\frac{m}{\Theta(\log^3(1/\varepsilon))}\right)$ over the randomness of the input set, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter for S. This is exactly the statement of Theorem 12.

[6] We implicitly assume here that $2^{1/\log(1/\varepsilon)}. \leq 2$, i.e., $\varepsilon \leq 1/2$. Notice that this does not affect the overall result, since the false positive rate we obtain is $5 \cdot \varepsilon$.

References

1 Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 787–796. IEEE, 2010. See also arXiv:0912.5424v3.

2 Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 182–193, 2018. doi:10.1109/FOCS.2018.00026.

3 Michael A Bender, Martin Farach-Colton, Rob Johnson, Bradley C Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. In *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, 2011.

4 Michael A. Bender, Martin Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 1284–1297. ACM, 2022. doi:10.1145/3519935.3519969.

5 Ioana O. Bercea and Guy Even. A dynamic space-efficient filter with constant time operations. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*, volume 162 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.11.

6 Ioana O. Bercea and Guy Even. Dynamic dictionaries for multisets and counting filters with constant time operations. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures – 17th International Symposium, WADS 2021, Virtual Event, August 9-11, 2021, Proceedings*, volume 12808 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2021. doi:10.1007/978-3-030-83508-8_11.

7 Ioana O. Bercea, Jakob Bæk Tejs Houen, and Rasmus Pagh. Daisy bloom filters. *CoRR*, abs/2205.14894, 2022. doi:10.48550/arXiv.2205.14894.

8 Ioana Oriana Bercea and Guy Even. An extendable data structure for incremental stable perfect hashing. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 1298–1310. ACM, 2022. doi:10.1145/3519935.3520070.

9 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

10 Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*, pages 684–695. Springer, 2006.

11 Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003. doi:10.1080/15427951.2004.10129096.

12 Jehoshua Bruck, Jie Gao, and Anxiao Jiang. Weighted bloom filter. In *International Symposium on Information Theory (ISIT)*, pages 2304–2308. IEEE, 2006. doi:10.1109/ISIT.2006.261978.

13 Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2014.

14 Xinyuan Cao, Jingbang Chen, Li Chen, Chris Lambert, Richard Peng, and Daniel Sleator. Learning-augmented b-trees, 2023. arXiv:2211.09251.

15 Larry Carter, Robert W. Floyd, John Gill, George Markowsky, and Mark N. Wegman. Exact and approximate membership testers. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 59–65. ACM, 1978.

16 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

17 Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

18 Zhenwei Dai and Anshumali Shrivastava. Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier with application to real-time information filtering on the web. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

19 Niv Dayan, Ioana O. Bercea, Pedro Reviriego, and Rasmus Pagh. Infinifilter: Expanding filters to infinity and beyond. *Proc. ACM Manag. Data*, 1(2):140:1–140:27, 2023. `doi:10.1145/3589285`.

20 Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership. In *International Colloquium on Automata, Languages, and Programming*, pages 385–396. Springer, 2008.

21 Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations (ICLR)*, 2020.

22 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, USA, 2012.

23 Talya Eden, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, and Tal Wagner. Learning-based support estimation in sublinear time. In *International Conference on Learning Representations*, 2020.

24 Tomer Even, Guy Even, and Adam Morrison. Prefix filter: Practically and theoretically better than bloom. *Proc. VLDB Endow.*, 15(7):1311–1323, 2022. `doi:10.14778/3523210.3523211`.

25 Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.

26 Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, 8(3):281–293, 2000.

27 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ESA.2023.46`.

28 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR, 2020.

29 Paolo Ferragina and Giorgio Vinciguerra. Learned data structures. In Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita, editors, *Recent Trends in Learning From Data – Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL 2019)*, volume 896 of *Studies in Computational Intelligence*, pages 5–41. Springer, 2019. `doi:10.1007/978-3-030-43883-8_2`.

30 Paolo Ferragina and Giorgio Vinciguerra. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment*, 13(8):1162–1175, 2020.

31 Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. In *Proceedings International Conference on Management of Data (SIGMOD)*, pages 1189–1206, 2019.

32 Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

33 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 489–504. ACM, 2018.

34 Mingmou Liu, Yitong Yin, and Huacheng Yu. Succinct filters for sets of unknown sizes. *arXiv preprint*, 2020. `arXiv:2004.12465`.

35 Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. Optimizing bloom filter: Challenges, solutions, and comparisons. *IEEE Commun. Surv. Tutorials*, 21(2):1912–1949, 2019. `doi:10.1109/COMST.2018.2889329`.

36 Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. Online list labeling with predictions, 2023. `arXiv:2305.10536`.

37 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

38 Michael Mitzenmacher, Salvatore Pontarelli, and Pedro Reviriego. Adaptive cuckoo filters. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 36–47. SIAM, 2018.

39 Moni Naor and Noa Oved. Bet-or-pass: Adversarially robust bloom filters. In *Theory of Cryptography Conference*, pages 777–808. Springer, 2022.

40 Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.

41 Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *SODA*, pages 823–829. SIAM, 2005.

42 Rasmus Pagh, Gil Segev, and Udi Wieder. How to approximate a set without knowing its size in advance. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 80–89. IEEE, 2013.

43 Prashant Pandey, Michael A Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM international conference on Management of Data*, pages 775–787, 2017.

44 Prashant Pandey, Alex Conway, Joe Durie, Michael A. Bender, Martin Farach-Colton, and Rob Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1386–1399. ACM, 2021. `doi:10.1145/3448016.3452841`.

45 Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *International Computer Science Symposium in Russia*, pages 263–273. Springer, 2009.

46 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

47 MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

48 Kapil Vaidya, Eric Knorr, Michael Mitzenmacher, and Tim Kraska. Partitioned learned bloom filters. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

49 Xiujun Wang, Yusheng Ji, Zhe Dang, Xiao Zheng, and Baohua Zhao. Improved weighted bloom filter and space lower bound analysis of algorithms for approximated membership querying. In *Database Systems for Advanced Applications (DASFAA)*, volume 9050 of *Lecture Notes in Computer Science*, pages 346–362. Springer, 2015. `doi:10.1007/978-3-319-18123-3_21`.

Online Bin Covering with Frequency Predictions

Magnus Berg
University of Southern Denmark, Odense, Denmark

Shahin Kamali
York University, Toronto, Canada

Abstract

We study the bin covering problem where a multiset of items from a fixed set $S \subseteq (0, 1]$ must be split into disjoint subsets while maximizing the number of subsets whose contents sum to at least 1. We focus on the online discrete variant, where S is finite, and items arrive sequentially. In the purely online setting, we show that the competitive ratios of best deterministic (and randomized) algorithms converge to $\frac{1}{2}$ for large S, similar to the continuous setting. Therefore, we consider the problem under the prediction setting, where algorithms may access a vector of frequencies predicting the frequency of items of each size in the instance. In this setting, we introduce a family of online algorithms that perform near-optimally when the predictions are correct. Further, we introduce a second family of more robust algorithms that presents a tradeoff between the performance guarantees when the predictions are perfect and when predictions are adversarial. Finally, we consider a stochastic setting where items are drawn independently from any fixed but unknown distribution of S. Using results from the PAC-learnability of probabilities in discrete distributions, we introduce a purely online algorithm whose average-case performance is near-optimal with high probability for all finite sets S and all distributions of S.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Online learning algorithms; Theory of computation → Online algorithms

Keywords and phrases Bin Covering, Online Algorithms with Predictions, PAC Learning, Learning-Augmented Algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.10

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2401.14881

Funding *Magnus Berg*: Supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B and in part by the Innovation Fund Denmark, grant 9142-00001B, Digital Research Centre Denmark, project P40: Online Algorithms with Predictions.
Shahin Kamali: Supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number DGECR-2018-00059].

1 Introduction

Bin Covering is a classical NP-complete [5] optimization problem where the input is a multiset of items, each with a size between 0 and 1. The objective is to split the items into disjoint subsets, called *bins*, while maximizing the number of bins whose contents sum to at least 1 [22]. The problem is often considered a dual to the bin packing problem, which asks for minimizing the number of bins, subject to each bin having a sum of at most 1.

In the online setting [18, 14, 5], items arrive one by one, and whenever an item arrives, an algorithm has to irrevocably place the item in an existing bin or open a new bin to place the item in. The existing results mostly consider a continuous setting in which items take any real value from $(0, 1]$, and it is well known that a simple greedy strategy, *Dual-Next-Fit* (DNF), achieves an optimal competitive ratio of $\frac{1}{2}$ [5].

In this paper, we consider a discrete variant of Online Bin Covering, where item sizes belong to a finite, known set $S \subseteq (0, 1]$. We abbreviate this problem by DBC_S. The special case when $S = \{\frac{i}{k} \mid i = 1, \ldots, k\}$ has been studied in the previous work. For example, Csirik, Johnson, and Kenyon [15] developed online algorithms with good average-case performance based on the *Sum of Squares* algorithm for Online Discrete Bin Packing [17, 16]. In this paper, we study a more general setting where S may be *any* finite subset of $(0, 1]$.

For measuring and comparing the quality of online algorithms for the DBC_S problem, we rely on the classical *competitive analysis* framework [9, 23], where one measures the quality of an online algorithm by comparing the performance of the algorithm to the performance of an optimal offline algorithm optimizing for the best worst-case guarantee.

1.1 Previous Work

The possibilities for creating algorithms for Online Bin Covering are well-studied. In the continuous setting, where items can take any size in $(0, 1]$, Assmann et al. [5] proved that DNF is $\frac{1}{2}$-competitive, and Csirik and Totik [18] presented an impossibility result showing that this is best possible. Later, Epstein [20] proved that the same impossibility result holds for randomized algorithms as well. Online Bin Covering has been studied under the advice setting [10, 12], where algorithms can access an advice tape that has encoded information about the input sequence. The aim is to determine how much additional information, measured by the number of bits needed to encode the information, is necessary and sufficient to achieve a certain competitive ratio and how well algorithms can perform when they are given a certain amount of information. For example, it is known that $\Theta(\log \log n)$ bits of advice are necessary and sufficient to achieve algorithms with a competitive ratio strictly better than $\frac{1}{2}$ [10], and that $O(b + \log(n))$ bits is sufficient to create an asymptotically $\frac{2}{3}$-competitive algorithm [12], where b is the number of bits needed to encode a rational value.

In recent years, developments in machine learning have inspired questions about how online algorithms may benefit from machine-learned advice [24, 25], commonly referred to as *predictions*. Unlike the advice model, the predictions may be erroneous or even adversarial. Online algorithms with predictions is a rapidly growing field (see, e.g., [1]) that aims at deriving online algorithms that provide a tradeoff between *consistency* and *robustness*. The consistency of an online algorithm refers to its competitive ratio when predictions are error-free; ideally, the consistency of an algorithm is 1 or close to 1. On the other hand, robustness refers to the competitive ratio assuming adversarial predictions; ideally, the robustness of an algorithm is close to the competitive ratio of the best purely online algorithm (with no prediction). These ideal cases, however, are not always realizable simultaneously, and one often settle for a consistency/robustness trade-off [25, 2, 27, 11, 3], giving explicit bounds on an algorithm's consistency as a function of its robustness, and vice versa.

To the authors' knowledge, no previous work on Bin Covering with predictions exists. The related Bin Packing problem, however, is previously studied under the prediction setting [4, 2].

1.2 Contribution

Our contributions for DBC_S can be summarized as follows. Throughout, we let $k = |S|$. In the continuous setting, where items take *any* real value in $(0, 1]$, no improvements in the competitive ratio can be achieved via predictions that are of size independent of input length, even if the predictions are error-free. This follows from a result of [10] that states any algorithm with an advice of size $o(\log \log n)$ is no better than $\frac{1}{2}$-competitive. Due to this negative result, we relax the problem and assume items come from a fixed, finite set. This relaxed setting is also studied for the related bin packing problem [4].

Purely online setting. We establish the following result on purely online algorithms for DBC_{F_k}, where $F_k = \{\frac{i}{k} \mid i = 1, 2, \dots, k\}$, based on ideas from [18] and [20] (all missing proofs can be found in the full paper [8]).

▶ **Theorem 1.** *Let* $\textsc{Alg}$ *be any deterministic or randomized online algorithm for* DBC_{F_k}*, with* $k \geqslant 5$*. Then,* $\textsc{Alg}$*'s competitive ratio is at most* $\frac{1}{2} + \frac{1}{H_{k-1}}$*, where* $H_{k-1} = \sum_{i=1}^{k-1} \frac{1}{i}$*.*

A consequence of Theorem 1 is the well-known fact [18, 20] that the competitive ratio of any deterministic or randomized algorithm for Online Bin Covering is at most $\frac{1}{2}$. This shows that Online Bin Covering is still a hard problem, even after discretization.

Prediction setting. We study DBC_S, where predictions concerning the frequency of item sizes are available. We start with an impossibility result that establishes a consistency/robustness tradeoff for this prediction scheme (Theorem 2). We then present an online algorithm, named *Group Covering*, which is near-optimal when the predictions are error-free, for all finite sets $S \subseteq (0, 1]$ (Theorem 5). Further, we create a family of hybrid algorithms that accepts a parameter λ, quantifying one's trust in the predictions. We establish a consistency/robustness tradeoff that bounds the consistency and robustness of these hybrid algorithms as a function of λ (Theorems 9 and 10).

Stochastic setting. Motivated by the work of Csirik, Johnson, and Kenyon [15], we study the purely online problem under a stochastic setting, where item sizes follow an unknown distribution. Unlike [15], which assumes items are of sizes $\frac{i}{k}$, for $i = 1, 2, \dots, k$, we do not make any assumption about input set S. We use a PAC-learning bound [13, 26] to create a family of online algorithms without predictions, whose expected performance ratio [15] is near-optimal with high probability, for any finite set S, and any unknown distribution D of S (Theorem 12).

2 Preliminaries

2.1 Online Discrete Bin Covering

Fix a finite set $S = \{s_1, s_2, \dots, s_k\} \subseteq (0, 1]$. An instance for *S-Discrete Bin Covering* is a sequence $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ of items, where $a_i \in S$, for $i \in [n]$. The task of an algorithm $\textsc{Alg}$ is to place the items in σ into bins $B_1, B_2, \dots, B_t$, maximizing the number of bins, B, for which $\sum_{a \in B} a \geqslant 1$. For any bin, B, we call $\mathrm{lev}(B) = \sum_{a' \in B} a'$ the *level* of B. We assume that algorithms are aware of S. In the online setting, the items are presented one-by-one to $\textsc{Alg}$, and upon receiving an item a, $\textsc{Alg}$ has to place a in a bin. This decision is irrevocable. We abbreviate *Online S-Discrete Bin Covering* by DBC_S. Throughout, we assume that $k \geqslant 2$, and we set $F_k = \left\{\frac{i}{k} \mid \text{for } i = 1, 2, \dots, k\right\}$, and abbreviate DBC_{F_k} by DBC_k.

2.2 Performance Measures

Given an online maximization problem, Π, an online algorithm, $\textsc{Alg}$, for Π, and an instance, σ, of Π, we let $\textsc{Alg}[\sigma]$ be $\textsc{Alg}$'s solution on instance σ and $\textsc{Alg}(\sigma)$ be the profit of $\textsc{Alg}[\sigma]$. If $\textsc{Alg}$ is deterministic, then the *competitive ratio* of $\textsc{Alg}$ is

$$\textsc{cr}_{\textsc{Alg}} = \sup\{c \in (0, 1] \mid \exists b > 0 \colon \forall \sigma \colon \textsc{Alg}(\sigma) \geqslant c \cdot \textsc{Opt}(\sigma) - b\},$$

where $\textsc{Opt}$ is an offline optimal algorithm for Π. Further, $\textsc{Alg}$ is *c-competitive* if $c \leqslant \textsc{cr}_{\textsc{Alg}}$.

For a fixed finite set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$, and a fixed (unknown) distribution D of S, the *asymptotic expected ratio* [19, 15] of an online algorithm, $\textsc{Alg}$, is

$$\mathrm{ER}^{\infty}_{\textsc{Alg}}(D) = \liminf_{n\to\infty} \mathbb{E}_D \left[\frac{\textsc{Alg}(\sigma_n(D))}{\textsc{Opt}(\sigma_n(D))}\right], \tag{1}$$

where $\sigma_n(D)$ is a sequence of n independent identically distributed random variables, $\sigma_n(D) = \langle X_1, X_2, \ldots, X_n\rangle$[1], where $X_i \sim D$, for all $i = 1, 2, \ldots, n$.

When an algorithm, $\textsc{Alg}$, has access to predictions, the *consistency* of $\textsc{Alg}$, and the *robustness* of $\textsc{Alg}$, is $\textsc{Alg}$'s competitive ratio when the predictions are error-free and adversarial, respectively. Throughout, we let $[n] = \{1, 2, \ldots, n\}$.

3 Predictions Setting

In this section, we assume that algorithms are given a *frequency prediction*, which, for a fixed instance σ, and each item $s_i \in S$, predicts what fraction of items in σ are of size s_i.

Formally, given a finite set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$, and an instance, σ, of DBC_S, we let n_i^σ be the number of items of size s_i in σ, n^σ be the number of items in σ, and $f_i^\sigma = \frac{n_i^\sigma}{n^\sigma}$. We call f_i^σ the *frequency* of items of size s_i in σ, and set $\boldsymbol{f^\sigma} = (f_1^\sigma, f_2^\sigma, \ldots, f_k^\sigma)$. When there can be no confusion, we abbreviate n_i^σ, n^σ, f_i^σ, and $\boldsymbol{f^\sigma}$, by n_i, n, f_i, and $\boldsymbol{f}$, respectively.

Throughout, we abbreviate *Online S-Discrete Bin Covering with Frequency Predictions* by $\mathrm{DBC}_S^{\mathcal{F}}$. An instance for $\mathrm{DBC}_S^{\mathcal{F}}$ is a tuple $(\sigma, \hat{\boldsymbol{f}})$ consisting of a sequence of items, σ, and a vector of predicted frequencies $\hat{\boldsymbol{f}} = \left(\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_k\right)$.

It is well-known that probabilities in discrete distributions are PAC-learnable, as shown in [13]. That is, there exists a polynomial-time algorithm that learns the probabilities in discrete distributions to arbitrary precision with a confidence that is arbitrarily close to 1, given sufficiently many random samples (see [26] for a formal definition of PAC-learnability). This makes frequency predictions easily attainable when historical data is available.

3.1 A Consistency-Robustness Trade-Off for $\mathrm{DBC}_k^{\mathcal{F}}$

In the following, by a *wasteful* algorithm, we mean an algorithm that sometimes places an item, a, in a bin, B, for which $\mathrm{lev}(B) \geqslant 1$ before a was placed in B. Any wasteful algorithm can be trivially converted to an equally good (possibly better) algorithm that avoids placing items into already-covered bins. Therefore, in what follows, we assume that all algorithms, including $\textsc{Opt}$, are non-wasteful.

▶ **Theorem 2.** *Any $(1-\alpha)$-consistent deterministic algorithm for $\mathrm{DBC}_k^{\mathcal{F}}$ is at most 2α-robust.*

Proof. Let $\textsc{Alg}$ be any deterministic online algorithm for $\mathrm{DBC}_k^{\mathcal{F}}$. Consider the instance $(\sigma_1^n, \hat{\boldsymbol{f}})$, with $\hat{\boldsymbol{f}} = (\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_k)$, where

$$\sigma_1^n = \left\langle \left\langle \frac{k-1}{k} \right\rangle^n, \left\langle \frac{1}{k} \right\rangle^n \right\rangle \quad \text{and} \quad \hat{f}_i = \begin{cases} \frac{1}{2}, & \text{if } s_i \in \left\{\frac{1}{k}, \frac{k-1}{k}\right\} \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, $\hat{\boldsymbol{f}}$ is a perfect prediction for σ_1^n, and $\textsc{Opt}(\sigma_1^n) = n$. Hence, by the consistency of $\textsc{Alg}$, there exists a constant b, such that

$$\textsc{Alg}(\sigma_1^n, \hat{\boldsymbol{f}}) \geqslant (1-\alpha) \cdot \textsc{Opt}(\sigma_1^n) - b = (1-\alpha) \cdot n - b. \tag{2}$$

[1] The particular choice of notation for X_i's is due to the items being random variables.

Let $\mathcal{B}_i$, for $i = 1, 2$, be the collection of bins that $\textsc{Alg}$ places i items of size $\frac{k-1}{k}$ in. Then, $\textsc{Alg}(\sigma_1^n, \hat{\boldsymbol{f}}) \leqslant |\mathcal{B}_1| + |\mathcal{B}_2| + \frac{n-|\mathcal{B}_1|}{k}$. Since $\textsc{Alg}$ is non-wasteful, $n = |\mathcal{B}_1| + 2 \cdot |\mathcal{B}_2|$, and so, by Equation (2), we have that $(1-\alpha)\cdot(|\mathcal{B}_1| + 2\cdot|\mathcal{B}_2|) - b \leqslant |\mathcal{B}_1| + \frac{(k+2)\cdot|\mathcal{B}_2|}{k}$, which implies

$$\frac{n\cdot\left(1 - 2\cdot\alpha - \frac{2}{k}\right) - 2\cdot b}{1-\frac{2}{k}} \leqslant |\mathcal{B}_1|\,. \tag{3}$$

Hence, since $\textsc{Alg}$ is $(1-\alpha)$-consistent, it has created at least $\frac{n\cdot\left(1-2\cdot\alpha-\frac{2}{k}\right)-2\cdot b}{1-\frac{2}{k}}$ bins that contain exactly one item of size $\frac{k-1}{k}$ after processing the first n items.

Next, consider the instance $(\sigma_2^n, \hat{\boldsymbol{f}})$, with imperfect predictions, where $\sigma_2^n = \left\langle\frac{k-1}{k}\right\rangle^n$. Since the first n requests of σ_1^n and σ_2^n are identical, $\textsc{Alg}$ cannot distinguish the instances $(\sigma_1^n, \hat{\boldsymbol{f}})$ and $(\sigma_2^n, \hat{\boldsymbol{f}})$ until it has seen the first n items. Hence, since $\textsc{Alg}$ is deterministic, it distributes the first n items identically on the two instances. Given that $n = |\mathcal{B}_1| + 2\cdot|\mathcal{B}_2|$, Equation (3) implies that

$$\textsc{Alg}(\sigma_2^n, \hat{\boldsymbol{f}}) \leqslant |\mathcal{B}_2| = \frac{n - |\mathcal{B}_1|}{2} \leqslant \frac{1}{2}\cdot\left(n - \frac{n\left(1-2\cdot\alpha-\frac{2}{k}\right)-2\cdot b}{1-\frac{2}{k}}\right) = \frac{2\cdot n\cdot\alpha + 2\cdot b}{2-\frac{4}{k}}.$$

Since $\textsc{Opt}(\sigma_2^n) = \frac{n}{2}$, then, for all $n \in \mathbb{Z}^+$, $\frac{\textsc{Alg}(\sigma_2^n,\hat{\boldsymbol{f}})}{\textsc{Opt}(\sigma_2^n)} \leqslant \frac{\frac{2\cdot n\cdot\alpha+2\cdot b}{2-\frac{4}{k}}}{\frac{n}{2}} = \frac{4\cdot n\cdot\alpha+4\cdot b}{n\cdot\left(2-\frac{4}{k}\right)} \leqslant 2\cdot\alpha - \frac{2\cdot b}{n}$, and thus $\textsc{Alg}$ is at most $2\cdot\alpha$-robust. ◀

Note that the impossibility result of Theorem 2 holds even for the special case of $S = F_k$. In fact, since we only use items from $\{\frac{1}{k}, \frac{k-1}{k}\}$ in input sequences of the proof, Theorem 2 can be stated for all finite sets $S \subseteq (0,1]$, for which $\{\frac{1}{k}, \frac{k-1}{k}\} \subseteq S$.

3.2 A Near-Optimally Consistent Algorithm for $\mathrm{DBC}_S^{\mathcal{F}}$

In this section, inspired by the *Profile Packing* algorithm from [4], we present a family of algorithms named *Group Covering*, parameterized by a parameter, ε, that receives frequency predictions, and outputs a $(1-\varepsilon)$-approximation of the optimal solution, assuming predictions are error-free. In other words, the algorithm achieves a consistency that is arbitrarily close to optimal. For a fixed $\varepsilon > 0$, we let GC_ε be the Group Covering algorithm with parameter ε.

The Strategy of Group Covering

Fix a finite set $S = \{s_1, s_2, \dots, s_k\} \subseteq (0,1]$. A *non-wasteful bin type* is an ordered l-tuple $(a_1, a_2, \dots, a_l)$ of items, with $l \geqslant 1$ and $a_i \in S$, for all $i \in [l]$, such that a_1 was placed in the bin first, then a_2, and so on, and such that $\sum_{i=1}^{l-1} a_i < 1$. Observe that this definition implies an ordering of the items in bin types, which is essential for our purpose. For example, the bin type $(1/2, 1/2, \varepsilon)$ is wasteful, as the bin is already covered after placing the second item of size $1/2$, but the bin type $(1/2, \varepsilon, 1/2)$ is non-wasteful, as removing the top item will make the bin no longer covered. Note that non-covered bins are also constitute a non-wasteful bin type. We let $\mathcal{T}_S$ denote the collection of all possible non-wasteful bin types given S, and set $\tau_S = |\mathcal{T}_S|$ and $t_{\max} = \max_{t\in\mathcal{T}_S}\{|t|\}$. For example, if $S = \left\{\frac{1}{k}, \frac{k-1}{k}\right\}$ then,

$$\mathcal{T}_S = \left\{\left(\underbrace{\frac{1}{k},\frac{1}{k},\dots,\frac{1}{k}}_{i\text{ times}}\right) \mid i\in[k]\right\} \cup \left\{\left(\underbrace{\frac{1}{k},\frac{1}{k},\dots,\frac{1}{k}}_{i\text{ times}},\frac{k-1}{k}\right) \mid i\in[k-1]\right\} \cup \left\{\left(\frac{k-1}{k}\right),\left(\frac{k-1}{k},\frac{1}{k}\right),\left(\frac{k-1}{k},\frac{k-1}{k}\right)\right\},$$

$\tau_S = 2k+2$, and $t_{\max} = k$.

Given an instance of $\mathrm{DBC}_S^{\mathcal{F}}$, $(\sigma, \hat{\boldsymbol{f}})$, GC_ε works as follows. In its initialization phase (before any item is placed), it creates an optimal solution to the following multiset, σ_{sub}, created based on $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$ (which it knows) and the frequency prediction:

$$\sigma_{\text{sub}} = \langle \lfloor \hat{f}_1 \cdot m_{k,\varepsilon} \rfloor, \lfloor \hat{f}_2 \cdot m_{k,\varepsilon} \rfloor, \ldots, \lfloor \hat{f}_k \cdot m_{k,\varepsilon} \rfloor \rangle,$$

where $m_{k,\varepsilon} = m_\varepsilon + k$, and $m_\varepsilon = \lceil 3 \cdot \tau_S \cdot t_{\max} \cdot \varepsilon^{-1} \rceil$. In this optimal solution, we maintain a *placeholder* of size a for any item $a \in \sigma_{\text{sub}}$. A placeholder of size a is a virtual item of size a, which reserves space for an item of size a. We let $P_{\hat{\boldsymbol{f}},\varepsilon}$ be the copy of $\textsc{Opt}[\sigma_{\text{sub}}]$ containing placeholders. To finish the initialization, GC_ε opens the first *group*, $G^1_{\hat{\boldsymbol{f}},\varepsilon}$; a copy of $P_{\hat{\boldsymbol{f}},\varepsilon}$.

When an item, a, arrives, GC_ε searches for a placeholder of size a in the open groups, searching in $G^1_{\hat{\boldsymbol{f}},\varepsilon}$ first, then $G^2_{\hat{\boldsymbol{f}},\varepsilon}$ second, and so on. If such a placeholder exists, GC_ε replaces the placeholder with a. If no such placeholder exists, GC_ε checks whether $P_{\hat{\boldsymbol{f}},\varepsilon}$ contains such a placeholder, by checking whether $a \in \sigma_{\text{sub}}$. If so, then GC_ε opens a new group, $G^i_{\hat{\boldsymbol{f}},\varepsilon}$, i.e. a new copy of $P_{\hat{\boldsymbol{f}},\varepsilon}$, and it replaces a newly created placeholder with a. Otherwise, GC_ε places a in an *extra*-bin using DNF. Extra bins are reserved for items that GC_ε did not expect to receive any of (items whose predicted frequency is 0 and thus are not in σ_{sub}). Pseudocode for GC_ε are given in Algorithm 1.

Analysis of GC_ε

We say that a group, $G^i_{\hat{\boldsymbol{f}},\varepsilon}$, is *completed* if all its placeholders have been replaced by items, and let g_ε be the number of groups that GC_ε completes. Recall that, by construction, GC_ε first completes $G^1_{\hat{\boldsymbol{f}},\varepsilon}$, then $G^2_{\hat{\boldsymbol{f}},\varepsilon}$, and so on.

▶ **Lemma 3.** *Fix any finite set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$, any $\varepsilon \in (0, 1)$, and any instance $(\sigma, \hat{\boldsymbol{f}})$ for $\mathrm{DBC}_S^{\mathcal{F}}$, with $\hat{\boldsymbol{f}} = \boldsymbol{f}$. Then, $\left\lfloor \frac{n}{m_{k,\varepsilon}} \right\rfloor \leqslant g_\varepsilon \leqslant \left\lfloor \frac{n}{m_\varepsilon} \right\rfloor$.*

Throughout, we let $\mathbf{p}(N)$ be the profit of a solution N for an input σ. Observe that $\mathbf{p}\left(G^1_{\hat{\boldsymbol{f}},\varepsilon}\right) = \mathbf{p}\left(G^i_{\hat{\boldsymbol{f}},\varepsilon}\right)$, for all $i \in [g_\varepsilon]$, i.e. all completed groups have the same profit.

▶ **Lemma 4.** *Fix any set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$, any $\varepsilon \in (0, 1)$, and any instance, $(\sigma, \hat{\boldsymbol{f}})$, for $\mathrm{DBC}_S^{\mathcal{F}}$, with $\hat{\boldsymbol{f}} = \boldsymbol{f}$ and $n^\sigma > m^2_{k,\varepsilon} + m_{k,\varepsilon}$. Then, $g_\varepsilon \cdot \mathbf{p}\left(G^1_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant (1 - \varepsilon) \cdot \textsc{Opt}(\sigma)$.*

Proof. We show this by creating a solution, N, based on $\textsc{Opt}[\sigma]$, such that

(i) $\mathbf{p}(N) \geqslant \left(1 - \frac{\varepsilon}{3}\right) \cdot \textsc{Opt}(\sigma)$, and

(ii) $g_\varepsilon \cdot \mathbf{p}\left(G^1_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant \left(1 - \frac{2 \cdot \varepsilon}{3}\right) \cdot \mathbf{p}(N)$.

Since $\varepsilon \in (0, 1)$, it suffices to prove (i) and (ii), because (i) and (ii) imply that

$$g_\varepsilon \cdot \mathbf{p}\left(G^1_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant \left(1 - \frac{2 \cdot \varepsilon}{3}\right) \cdot \left(1 - \frac{\varepsilon}{3}\right) \cdot \textsc{Opt}(\sigma) \geqslant (1 - \varepsilon) \cdot \textsc{Opt}(\sigma).$$

Construction of N. Initially, let N be a copy of $\textsc{Opt}[\sigma]$. Since $\textsc{Opt}$ is non-wasteful, all bins in $\textsc{Opt}[\sigma]$ are filled according to non-wasteful bin types. For each non-wasteful bin type $t \in \mathcal{T}_S$, remove between 0 and $g_\varepsilon - 1$ bins of type t from N, such that the number of bins of type t becomes divisible by g_ε.

Proof of (i). Since $\textsc{Opt}(\sigma) \geqslant \frac{n^\sigma}{t_{\max}}$, Lemma 3 implies that

$$\begin{aligned}\mathbf{p}(N) &\geqslant \textsc{Opt}(\sigma) - (g_\varepsilon - 1) \cdot \tau_S \geqslant \textsc{Opt}(\sigma) - \frac{n^\sigma}{m_\varepsilon} \cdot \tau_S \\ &\geqslant \textsc{Opt}(\sigma) - \textsc{Opt}(\sigma) \cdot \frac{\tau_S \cdot t_{\max}}{m_\varepsilon} \geqslant \left(1 - \frac{\varepsilon}{3}\right) \cdot \textsc{Opt}(\sigma).\end{aligned}$$

Algorithm 1 GC$_{\varepsilon}$.

```
Input: a DBC$_S^{\mathcal{F}}$-instance. $(\sigma, \hat{\boldsymbol{f}})$
$j, l \leftarrow 1$
Compute $\tau_S$, $t_{\max}$, and $k = |S|$
$m_\varepsilon \leftarrow \lceil 3 \cdot \tau_S \cdot t_{\max} \cdot \varepsilon^{-1} \rceil$
$m_{k,\varepsilon} \leftarrow m_\varepsilon + k$
$\sigma_{\text{sub}} \leftarrow \langle \lfloor \hat{f}_1 \cdot m_{k,\varepsilon} \rfloor, \lfloor \hat{f}_2 \cdot m_{k,\varepsilon} \rfloor, \ldots, \lfloor \hat{f}_k \cdot m_{k,\varepsilon} \rfloor \rangle$
$P_{\hat{\boldsymbol{f}},\varepsilon} \leftarrow \emptyset$
for all $B \in \textsc{Opt}[\sigma_{\text{sub}}]$ do
    $B' \leftarrow \emptyset$                                          ▷ Create a new empty bin
    for all $a \in B$ do
        $B' \leftarrow B' \cup \{p_a\}$                          ▷ Add a placeholder of size $a$ to $B'$
    $P_{\hat{\boldsymbol{f}},\varepsilon} \leftarrow P_{\hat{\boldsymbol{f}},\varepsilon} \cup B'$          ▷ Add a copy of $B$ containing placeholders to $P_{\hat{\boldsymbol{f}},\varepsilon}$
$G^1_{\hat{\boldsymbol{f}},\varepsilon} \leftarrow P_{\hat{\boldsymbol{f}},\varepsilon}$                                          ▷ Open the first group
while receiving items, $a$, do
    not_placed ← true                          ▷ Marks whether $a$ still has to be placed
    for $i = 1, 2, \ldots, l$ do                    ▷ Go through open groups chronologically
        if not_placed then                ▷ To avoid trying to place $a$ multiple times
            if $\exists B \in G^i_{\hat{\boldsymbol{f}},\varepsilon} : p_a \in B$ then                      ▷ Search for $p_a$ in $G^i_{\hat{\boldsymbol{f}},\varepsilon}$
                $B \leftarrow B \setminus \{p_a\} \cup \{a\}$                ▷ Swap out placeholder, $p_a$, for $a$
                not_placed ← false                         ▷ $a$ has been placed in a bin
    if not_placed then                          ▷ Checking whether $a$ has been placed
        if $\lfloor \hat{f}_a \cdot m_{k,\varepsilon} \rfloor \neq 0$ then                  ▷ Checking whether $a \in \sigma_{\text{sub}}$
            $l \leftarrow l + 1$
            $G^l_{\hat{\boldsymbol{f}},\varepsilon} \leftarrow \textsc{Opt}[\sigma_{\text{sub}}]$                                       ▷ Open a new group
            Determine $B \in G^l_{\hat{\boldsymbol{f}},\varepsilon}$ such that $p_a \in B$, and $B \leftarrow B \setminus \{p_a\} \cup \{a\}$
        else                                                                   ▷ $a \notin \sigma_{\text{sub}}$
            $B^E_j \leftarrow B^E_j \cup \{a\}$                      ▷ Place $a$ in a *extra* bin using DNF
            if lev$(B^E_j) \geqslant 1$ then
                $j \leftarrow j + 1$
                $B^E_j \leftarrow \emptyset$
```

Proof of (ii). Since the number of occurrences of each bin type in N is divisible by g_ε, we may consider N as g_ε identical copies of a smaller covering $\overline{N}$. Since we do not add any items when creating N, and thus $\overline{N}$, we have $n_i^{\overline{N}} \leqslant \left\lfloor \frac{n_i^\sigma}{g_\varepsilon} \right\rfloor$, for all $i \in [k]$, where $n_i^{\overline{N}}$ denotes the number of items of size i in $\overline{N}$. Then, for all $i \in [k]$, we can write

$$n_i^{\overline{N}} \leqslant \left\lfloor \frac{n_i^\sigma}{g_\varepsilon} \right\rfloor \leqslant \left\lfloor \frac{n_i^\sigma}{\left\lfloor \frac{n^\sigma}{m_{k,\varepsilon}} \right\rfloor} \right\rfloor \leqslant \left\lfloor \frac{n_i^\sigma}{\frac{n^\sigma}{m_{k,\varepsilon}} - 1} \right\rfloor = \left\lfloor \frac{n_i^\sigma}{\frac{n^\sigma - m_{k,\varepsilon}}{m_{k,\varepsilon}}} \right\rfloor = \left\lfloor n_i^\sigma \cdot \frac{m_{k,\varepsilon}}{n^\sigma - m_{k,\varepsilon}} \right\rfloor .$$

Given that $\frac{m_{k,\varepsilon}}{n^\sigma - m_{k,\varepsilon}} = \frac{m_{k,\varepsilon}}{n^\sigma} + \frac{m_{k,\varepsilon}^2}{n^\sigma \cdot (n^\sigma - m_{k,\varepsilon})}$, and that $n^\sigma > m_{k,\varepsilon}^2 + m_{k,\varepsilon}$, we may conclude

$$n_i^{\overline{N}} \leqslant \left\lfloor \frac{n_i^\sigma \cdot m_{k,\varepsilon}}{n^\sigma} + \frac{m_{k,\varepsilon}^2}{n^\sigma - m_{k,\varepsilon}} \right\rfloor \leqslant \left\lfloor \frac{n_i^\sigma \cdot m_{k,\varepsilon}}{n^\sigma} \right\rfloor + 1 = \lfloor f_i \cdot m_{k,\varepsilon} \rfloor + 1.$$

Hence, $\overline{N}$ contains at most one more item of size s_i than $G^j_{\hat{\boldsymbol{f}},\varepsilon}$, for all $i \in [k]$, and all $j \in [g_\varepsilon]$. Then, for all $j \in [g_\varepsilon]$, the following holds:

$$\mathbf{p}\left(G^{j}_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant \mathbf{p}\left(\overline{N}\right) - k. \tag{4}$$

Next, we devise a lower bound for $\mathbf{p}\left(\overline{N}\right)$. Since $\textsc{Opt}(\sigma) \geqslant \frac{n^{\sigma}}{t_{\max}}$,

$$\begin{aligned}\mathbf{p}\left(\overline{N}\right) &= \frac{\mathbf{p}(N)}{g_{\varepsilon}} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot \textsc{Opt}(\sigma)}{g_{\varepsilon}} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot n^{\sigma}}{t_{\max}\cdot g_{\varepsilon}} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot n^{\sigma}}{t_{\max}\cdot \frac{n^{\sigma}}{m_{\varepsilon}}}\\ &= \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot m_{\varepsilon}}{t_{\max}} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot \frac{3\cdot\tau_S\cdot t_{\max}}{\varepsilon}}{t_{\max}} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot 3\cdot \tau_S}{\varepsilon} \geqslant \frac{\left(1-\frac{\varepsilon}{3}\right)\cdot k}{\frac{\varepsilon}{3}}.\end{aligned}$$

Hence, $k \leqslant \frac{\frac{\varepsilon}{3}\cdot\mathbf{p}\left(\overline{N}\right)}{1-\frac{\varepsilon}{3}}$, and so, by Equation (4), $\mathbf{p}\left(G^{j}_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant \mathbf{p}\left(\overline{N}\right) - \frac{\frac{\varepsilon}{3}\cdot\mathbf{p}\left(\overline{N}\right)}{1-\frac{\varepsilon}{3}} \geqslant \left(1-\frac{2\cdot\varepsilon}{3}\right)\cdot\mathbf{p}\left(\overline{N}\right)$. Since $\mathbf{p}(N) = g_{\varepsilon}\cdot\mathbf{p}\left(\overline{N}\right)$ and $\mathbf{p}\left(G^{j}_{\hat{\boldsymbol{f}},\varepsilon}\right) = \mathbf{p}\left(G^{1}_{\hat{\boldsymbol{f}},\varepsilon}\right)$, for all $j \in [g_{\varepsilon}]$, we conclude $g_{\varepsilon}\cdot\mathbf{p}\left(G^{1}_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant g_{\varepsilon}\cdot\left(1-\frac{2\cdot\varepsilon}{3}\right)\cdot\mathbf{p}\left(\overline{N}\right) = \left(1-\frac{2\cdot\varepsilon}{3}\right)\cdot\mathbf{p}(N)$, which establishes (ii). ◀

Given Lemma 4, it is straightforward to deduce the following theorem, which is the main result of this section.

▶ **Theorem 5.** *For any set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0,1]$, and any $\varepsilon \in (0,1)$, there exists a constant, b, such that for all instances $(\sigma, \hat{\boldsymbol{f}})$, with $\boldsymbol{f} = \hat{\boldsymbol{f}}$, it holds that $GC_{\varepsilon}(\sigma, \hat{\boldsymbol{f}}) \geqslant (1-\varepsilon)\cdot \textsc{Opt}(\sigma) - b$. That is, GC_{ε} is a $(1-\varepsilon)$-consistent algorithm for $DBC_S^{\mathcal{F}}$.*

While the above theorem shows that $\mathrm{GC}_{\varepsilon}$ is almost optimally consistent, the same cannot be said about its robustness. Consider the instance $(\sigma^n, \hat{\boldsymbol{f}})$ where $\sigma^n = \left\langle\frac{1}{k}\right\rangle^n$ and $\hat{\boldsymbol{f}}$ predicts that half of the items are of size $\frac{1}{k}$, and half of the items are of size $\frac{k-1}{k}$, a wrong prediction for σ^n. Based on the predictions $\hat{\boldsymbol{f}}$, $\mathrm{GC}_{\varepsilon}$ creates $\left\lfloor\frac{m_{k,\varepsilon}}{2}\right\rfloor$ bins that contain placeholders for one item of size $\frac{1}{k}$, and one item of size $\frac{k-1}{k}$. Since no item of size $\frac{k-1}{k}$ appears in the input, $\mathrm{GC}_{\varepsilon}$ never covers a bin, and since $\textsc{Opt}(\sigma^n) = \left\lfloor\frac{n}{k}\right\rfloor$, $\mathrm{GC}_{\varepsilon}$ is not robust. In the next section, we introduce a strategy for improving the robustness of $\mathrm{GC}_{\varepsilon}$.

3.3 Robustifying GC$_{\varepsilon}$

For each purely online algorithm, $\textsc{Alg}$ (e.g. $\textsc{dnf}$), we create a family of *hybrid algorithms* that combines $\mathrm{GC}_{\varepsilon}$ with $\textsc{Alg}$ to improve the robustness of $\mathrm{GC}_{\varepsilon}$. Formally, for any algorithm, $\textsc{Alg}$, we create the family $\{\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}\}_{\lambda,\varepsilon}$, of hybrid algorithms, parametrized by $\varepsilon \in (0,1)$ and a *trust level*, $\lambda \in \mathbb{Q}^{+}$. Throughout, we assume that λ is given as a fraction, $\lambda = \frac{\kappa}{\ell}$, for some $\kappa \in \mathbb{N}$ and $\ell \in \mathbb{Z}^{+}$. For any item $a \in S$, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ maintains a counter for the number of items of size a in the input observed so far. Upon receiving an item a, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ counts the number of occurrences of a, denoted c_a, and if $c_a \pmod{\ell} \leqslant \ell - \kappa - 1$, it uses $\textsc{Alg}$ to place a in a bin that only $\textsc{Alg}$ places items into, and otherwise, it uses $\mathrm{GC}_{\varepsilon}$ to place a in a bin that only $\mathrm{GC}_{\varepsilon}$ places items into. The pseudo-code for $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ is given in Algorithm 2.

For the analysis of $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$, we associate, to any instance σ of DBC_S, a $(\ell+1)$-tuple, $(\sigma_1, \sigma_2, \ldots, \sigma_{\ell}, \sigma_e)$ called the *ℓ-splitting of* σ, which is created as follows. Process the items one-by-one, in the order they appear in σ; when processing an item a, place it in σ_{i+1} if $c_a \pmod{\ell} \equiv i$, where c_a is the number of items of size a previously recorded. After processing all items in σ, we compute the number of items of size s_i, for any $s_i \in S$, in each σ_j, for all $i \in [k]$ and all $j \in [\ell]$. If there are equally many items of size s_i in all σ_j, we are done. If, on the other hand, there exists some $i \in [k]$ and some $j \in [\ell]$ such that $\sigma_1, \sigma_2, \ldots, \sigma_j$ contains one more item of size s_i than $\sigma_{j+1}, \sigma_{j+2}, \ldots, \sigma_{\ell}$, then we remove one item of size s_i from all of $\sigma_1, \sigma_2, \ldots, \sigma_j$, and place it in σ_e instead. The pseudo-code for this process is given in the full paper [8].

Algorithm 2 $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$.

```
Input: An instance for $\mathrm{DBC}^{\mathcal{F}}_{S}$, $(\sigma, \hat{\boldsymbol{f}})$
Determine $\kappa, \ell \in \mathbb{Z}^{+}$ such that $\lambda = \frac{\kappa}{\ell}$
Run Lines 2-13 of $\mathrm{GC}_{\varepsilon}$ (see Algorithm 1), given the prediction $\hat{\boldsymbol{f}}$
Run initialization part of $\textsc{Alg}$, if such exists
for all $i \in [k]$ do
    $c_{s_i} \leftarrow 0$
while receiving items, $a$, do
    $j \leftarrow c_a \pmod{\ell}$                                  ▷ $a \in \sigma_{j+1}$
    if $j \leqslant \ell - \kappa - 1$ then
        Ask $\textsc{Alg}$ to place $a$
    else                                                            ▷ $\ell - \kappa \leqslant j \leqslant \ell - 1$
        Ask $\mathrm{GC}_{\varepsilon}$ to place $a$                ▷ See Lines 14-30 in Algorithm 1
    $c_a \leftarrow c_a + 1$
```

By construction, the ℓ-splitting of σ decomposes σ into ℓ smaller instances, σ_i for $i \in [\ell]$, that all contain the same multiset of items, but possibly in different orders, and an *excess* instance σ_e, which contain the remaining items from σ. By construction, $n^{\sigma_e} \leqslant (\ell - 1) \cdot k$.

Bounding the Performance of the Optimal Packing

In what follows, we present an upper bound for the number of bins covered by $\textsc{Opt}$. Throughout, given ℓ instances, $\sigma_1, \sigma_2, \ldots, \sigma_\ell$, we set $\bigcup_{i=1}^{\ell} \sigma_i = \langle \sigma_1, \sigma_2, \ldots, \sigma_\ell \rangle$.

▶ **Observation 6.** *Let $\sigma_1, \sigma_2, \ldots, \sigma_\ell$ be any instances for DBC_S, then $\sum_{i=1}^{\ell} \textsc{Opt}(\sigma_i) \leqslant \textsc{Opt}\left(\bigcup_{i=1}^{\ell} \sigma_i\right)$.*

▶ **Lemma 7.** *Let $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$ be any finite set, let σ by any instance of DBC_S, and let $(\sigma_1, \sigma_2, \ldots, \sigma_\ell, \sigma_e)$ be the ℓ-splitting of σ. Then, $\textsc{Opt}(\sigma) \leqslant \sum_{i=1}^{\ell} \textsc{Opt}(\sigma_i) + (\ell - 1) \cdot (k + \tau_S)$.*

Proof. We split this proof into two parts, by showing that

(i) $\textsc{Opt}\left(\bigcup_{i=1}^{\ell} \sigma_i\right) \leqslant \sum_{i=1}^{\ell} \textsc{Opt}(\sigma_i) + (\ell - 1) \cdot \tau_S$, and

(ii) $\textsc{Opt}(\sigma) \leqslant \textsc{Opt}\left(\bigcup_{i=1}^{\ell} \sigma_i\right) + (\ell - 1) \cdot k$.

Proof of (i). We use a similar strategy as in the proof of Theorem 5. To this end, let N be the solution obtained by removing at most $\ell - 1$ bins of each non-wasteful bin type from a copy of $\textsc{Opt}\left[\bigcup_{i=1}^{\ell} \sigma_i\right]$ (recall that $\textsc{Opt}$ is non-wasteful) such that the number of each bin type in N is divisible by ℓ. Then, $\mathbf{p}(N) \geqslant \textsc{Opt}\left(\bigcup_{i=1}^{\ell} \sigma_i\right) - (\ell - 1) \cdot \tau_S$. Therefore, it suffices to compare the profit of $\bigcup_{i=1}^{\ell} \textsc{Opt}[\sigma_i]$ to $\mathbf{p}(N)$. Since $\sigma_1, \sigma_2, \ldots, \sigma_\ell$ all contain the same multiset of items (but possibly in a different order), it holds that $\textsc{Opt}(\sigma_i) = \textsc{Opt}(\sigma_j)$, for all $i, j \in [\ell]$. Further, by construction, N is the union of ℓ identical smaller coverings, $\overline{N}$, for which $n_i^{\overline{N}} \leqslant n_i^{\sigma_i}$, for all $i \in [k]$. Therefore, $\textsc{Opt}(\sigma_i) \geqslant \mathbf{p}\left(\overline{N}\right)$, for all $i \in [k]$, and we can write $\sum_{i=1}^{\ell} \textsc{Opt}(\sigma_i) = \ell \cdot \textsc{Opt}(\sigma_1) \geqslant \ell \cdot \mathbf{p}\left(\overline{N}\right) = \mathbf{p}(N)$, which completes the proof of (i).

Proof of (ii). Since $n^{\sigma_e} \leqslant (\ell - 1) \cdot k$, we can write $\textsc{Opt}\left(\bigcup_{i=1}^{\ell} \sigma_i\right) \geqslant \textsc{Opt}(\sigma) - (\ell - 1) \cdot k$. Adding $(\ell - 1) \cdot k$ to both sides establishes (ii) and thus completes the proof. ◀

A Bound on the Performance of GC_ε

We compare the number of bins covered by GC_ε on a subset of the instances in the ℓ-splitting of an instance, σ, to that of $\textsc{Opt}$ on σ. To this end, observe that if σ is a DBC_S-instance, where $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0,1]$, and $(\sigma_1, \sigma_2, \ldots, \sigma_\ell, \sigma_e)$ is the ℓ-splitting of σ, then $n_j^{\sigma_i} = \left\lfloor \frac{n_j^\sigma}{\ell} \right\rfloor$, for all $j \in [k]$ and all $i \in [\ell]$.

▶ **Lemma 8.** *Fix any set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0,1]$, any $\varepsilon \in (0,1)$, and any instance $(\sigma, \hat{\boldsymbol{f}})$ of DBC_S, for which $\boldsymbol{f} = \hat{\boldsymbol{f}}$, and let $(\sigma_1, \sigma_2, \ldots, \sigma_\ell, \sigma_e)$ be the ℓ-splitting of σ, for some $\ell \in \mathbb{Z}^+$. Then, for any $j \in \mathbb{Z}^+$, with $j \leqslant \ell$, there exists a constant b such that $GC_\varepsilon\left(\left(\bigcup_{i=\ell-j+1}^{\ell} \sigma_i\right), \hat{\boldsymbol{f}}\right) \geqslant \frac{j \cdot (1-\varepsilon) \cdot \textsc{Opt}(\sigma)}{\ell} - b$.*

Proof. Let $\tilde{\sigma_j} = \bigcup_{i=\ell-j+1}^{\ell} \sigma_i$, and set $b = m_{k,\varepsilon}^2 + m_{k,\varepsilon} + k \cdot \ell$. If $n^\sigma \leqslant b$, the right-hand side is non-positive, and the left-hand side is non-negative, and the lemma's statement follows.

Hence, assume that $n^\sigma > b$. Let $C = \mathrm{GC}_\varepsilon[\sigma, \hat{\boldsymbol{f}}]$, and let g_ε be the number of groups, $G^i_{\hat{\boldsymbol{f}},\varepsilon}$, that GC_ε completes on instance $(\sigma, \hat{\boldsymbol{f}})$. By Lemma 4, we have $g_\varepsilon \cdot \mathbf{p}\left(G^1_{\hat{\boldsymbol{f}},\varepsilon}\right) \geqslant (1-\varepsilon) \cdot \textsc{Opt}(\sigma)$. Since $G^i_{\hat{\boldsymbol{f}},\varepsilon}$ is only dependent on ε, S, and $\hat{\boldsymbol{f}}$, GC_ε creates the same groups, $G^i_{\hat{\boldsymbol{f}},\varepsilon}$, on instance $(\sigma, \hat{\boldsymbol{f}})$ as on instance $(\tilde{\sigma_j}, \hat{\boldsymbol{f}})$. In the following, we prove a lower bound for the number of groups that GC_ε completes on instance $(\tilde{\sigma_j}, \hat{\boldsymbol{f}})$, as a function of g_ε.

Since C completely covers g_ε copies of $G^i_{\hat{\boldsymbol{f}},\varepsilon}$, then $n_i^\sigma \geqslant g_\varepsilon \cdot \lfloor f_i^\sigma \cdot m_{k,\varepsilon} \rfloor$ for all $i \in [k]$. Moreover, given that each σ_i contains exactly $\left\lfloor \frac{n_i^\sigma}{\ell} \right\rfloor$ items of size s_i, we have

$$n_i^{\tilde{\sigma}_j} \geqslant j \cdot \left\lfloor \frac{n_i^\sigma}{\ell} \right\rfloor \geqslant \frac{j \cdot n_i^\sigma}{\ell} - j \geqslant \frac{j \cdot g_\varepsilon}{\ell} \cdot \lfloor f_i^\sigma \cdot m_{k,\varepsilon} \rfloor - j \geqslant \left\lfloor \frac{j \cdot g_\varepsilon}{\ell} \right\rfloor \cdot \lfloor f_i^\sigma \cdot m_{k,\varepsilon} \rfloor - j.$$

This implies that, GC_ε fills in all placeholders for items of size s_i in $\left\lfloor \frac{j \cdot g_\varepsilon}{\ell} \right\rfloor$ groups, except at most j, on instance $(\tilde{\sigma}_j, \hat{\boldsymbol{f}})$, for all $i \in [k]$. Hence,

$$\mathrm{GC}_\varepsilon(\tilde{\sigma}_j, \hat{\boldsymbol{f}}) \geqslant \left\lfloor \frac{j \cdot g_\varepsilon}{\ell} \right\rfloor \cdot \mathbf{p}\left(G^i_{\hat{\boldsymbol{f}},\varepsilon}\right) - k \cdot j \geqslant \left(\frac{j \cdot g_\varepsilon}{\ell} - 1\right) \cdot \mathbf{p}\left(G^i_{\hat{\boldsymbol{f}},\varepsilon}\right) - k \cdot j.$$

Since $\mathbf{p}\left(G^i_{\hat{\boldsymbol{f}},\varepsilon}\right) \leqslant m_{k,\varepsilon}$, we conclude the following, which completes the proof:

$$\mathrm{GC}_\varepsilon(\tilde{\sigma}_j, \hat{\boldsymbol{f}}) \geqslant \frac{j \cdot g_\varepsilon}{\ell} \cdot \mathbf{p}\left(G^i_{\hat{\boldsymbol{f}},\varepsilon}\right) - k \cdot j - m_{k,\varepsilon} \geqslant \frac{j \cdot (1-\varepsilon) \cdot \textsc{Opt}(\sigma)}{\ell} - b. \qquad ◀$$

A Trust-Parametrized Family of Hybrid Algorithms

In what follows, we wrap up the analysis of $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ by stating and proving the main results of this section. By construction, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ (see Algorithm 2) distributes the items that arrive between GC_ε and $\textsc{Alg}$ in a way determined by λ. Whenever λ becomes close to 1, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ assigns a larger fraction of items to GC_ε, and when λ gets close to 0, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ assigns more items to $\textsc{Alg}$. In particular, $\textsc{Hyb}^{1,\varepsilon}_{\textsc{Alg}} = \mathrm{GC}_\varepsilon$, and $\textsc{Hyb}^{0,\varepsilon}_{\textsc{Alg}} = \textsc{Alg}$. Clearly, $\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}$ cannot create a perfect ℓ-splitting online, since it cannot correctly identify the items that are placed in σ_e. It can, however, get sufficiently close.

▶ **Theorem 9.** *For any finite set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0,1]$, any purely online $DBC_S^{\mathcal{F}}$-algorithm, $\textsc{Alg}$, any $c \leqslant \textsc{cr}_{\textsc{Alg}}$, any $\varepsilon \in (0,1)$, and any $\lambda \in \mathbb{Q}^+$, there exists a constant $b \in \mathbb{Z}^+$, such that for all instances $(\sigma, \hat{\boldsymbol{f}})$, the following holds, assuming $\boldsymbol{f} = \hat{\boldsymbol{f}}$:*

$$\textsc{Hyb}^{\lambda,\varepsilon}_{\textsc{Alg}}(\sigma, \hat{\boldsymbol{f}}) \geqslant (\lambda \cdot (1-\varepsilon) + (1-\lambda) \cdot c) \cdot \textsc{Opt}(\sigma) - b.$$

Proof. Let b_{ALG} be the additive constant of ALG, $b_{\text{GC}_\varepsilon} = m_{k,\varepsilon}^2 + m_{k,\varepsilon} + k \cdot \ell$. Then, we set $b = b_{\text{ALG}} + b_{\text{GC}_\varepsilon} + (\ell - 1) \cdot (k + \tau_S)$. If $n^\sigma \leqslant b$, the result follows trivially. Hence, assume that $n^\sigma > b$.

Let $(\sigma_1, \sigma_2, \ldots, \sigma_\ell, \sigma_e)$ be the ℓ-splitting of σ, and let σ_e^{ALG} and $\sigma_e^{\text{GC}_\varepsilon}$ be the collection of instances from σ_e that ALG and GC_ε receive, respectively. Then, by definition of $\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}$,

$$\begin{aligned}\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}[\sigma, \hat{\boldsymbol{f}}] &= \text{ALG}\left[\left(\bigcup_{i=1}^{\ell-\kappa} \sigma_i\right) \cup \sigma_e^{\text{ALG}}\right] \cup \text{GC}_\varepsilon\left[\left(\bigcup_{i=\ell-\kappa+1}^{\ell} \sigma_i\right) \cup \sigma_e^{\text{GC}_\varepsilon}, \hat{\boldsymbol{f}}\right] \\ &\geqslant \text{ALG}\left(\bigcup_{i=1}^{\ell-\kappa} \sigma_i\right) + \text{GC}_\varepsilon\left(\left(\bigcup_{i=\ell-\kappa+1}^{\ell} \sigma_i\right), \hat{\boldsymbol{f}}\right).\end{aligned}$$

Set $b' = b_{\text{ALG}} + b_{\text{GC}_\varepsilon}$. Then, by c-competitiveness of ALG and Lemma 8, we can write

$$\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}(\sigma, \hat{\boldsymbol{f}}) \geqslant c \cdot \text{OPT}\left(\bigcup_{i=1}^{\ell-\kappa} \sigma_i\right) + \lambda \cdot (1 - \varepsilon) \cdot \text{OPT}(\sigma) - b'.$$

Since $\text{OPT}(\sigma_i) = \text{OPT}(\sigma_j)$ for all $i, j \in [\ell]$ then, by Observation 6, we have $\sum_{i=1}^{\ell-\kappa} \text{OPT}(\sigma_i) \leqslant \text{OPT}\left(\bigcup_{i=1}^{\ell-\kappa} \sigma_i\right)$. Therefore, from the above inequality, we can conclude

$$\begin{aligned}\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}(\sigma, \hat{\boldsymbol{f}}) &\geqslant c \cdot \left(\sum_{i=1}^{\ell-\kappa} \text{OPT}(\sigma_i)\right) + \lambda \cdot (1 - \varepsilon) \cdot \text{OPT}(\sigma) - b' \\ &= (1 - \lambda) \cdot c \cdot \left(\sum_{i=1}^{\ell} \text{OPT}(\sigma_i)\right) + \lambda \cdot (1 - \varepsilon) \cdot \text{OPT}(\sigma) - b'.\end{aligned}$$

Combining Lemma 7 and the above bound for $\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}(\sigma, \hat{\boldsymbol{f}})$, we can conclude the following, which completes the proof:

$$\begin{aligned}\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}(\sigma, \hat{\boldsymbol{f}}) &\geqslant (1 - \lambda) \cdot c \cdot (\text{OPT}(\sigma) - (\ell - 1) \cdot (k + \tau_S)) + \lambda \cdot (1 - \varepsilon) \cdot \text{OPT}(\sigma) - b' \\ &\geqslant ((1 - \lambda) \cdot c + \lambda \cdot (1 - \varepsilon)) \cdot \text{OPT}(\sigma) - b.\end{aligned}$$ ◀

The above theorem gives an explicit formula for the consistency of $\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}$ as a function of the trust-level, λ, $\varepsilon \in (0, 1)$, and the performance guarantee of ALG. A similar proof can be used to establish a guarantee on the robustness of $\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}$.

▶ **Theorem 10.** *For any finite set $S = \{s_1, s_2, \ldots, s_k\} \subseteq (0, 1]$, any purely online algorithm, ALG, for DBC_S, any $c \leqslant \text{CR}_{\text{ALG}}$, and any ε, there exists a constant $b \in \mathbb{Z}^+$, such that for all instances $(\sigma, \hat{\boldsymbol{f}})$, $\text{HYB}_{\text{ALG}}^{\lambda,\varepsilon}(\sigma, \hat{\boldsymbol{f}}) \geqslant (1 - \lambda) \cdot c \cdot \text{OPT}(\sigma) - b$.*

4 Stochastic Setting

In this section, we consider a setting for DBC_S where item sizes are generated independently at random from an unknown distribution. This setting has already been studied for the more restricted DBC_k problem, where Csirik, Johnson and Kenyon used variants of the Bin Packing algorithm "Sum-of-Squares", first introduced in [17, 16], to develop algorithms for DBC_k. Rather than designing algorithms that perform well in the worst case, they aimed to design algorithms that perform well on average. Specifically, they develop an algorithm, called SS^*, with $\text{ER}_{SS^*}^{\infty}(D) = 1$ (see Equation (1) for the definition of $\text{ER}_{SS^*}^{\infty}(D)$), for all discrete distributions D of F_k, with rational probabilities.

In this section, we use a PAC-learning bound for learning frequencies in discrete distributions to derive a family of algorithms called *purely online group covering* ($\{\text{POGC}_\varepsilon^\delta\}_{\varepsilon,\delta}$). These algorithms are parametrized by two real numbers $\varepsilon, \delta \in (0,1)$, satisfying that, for *all* finite sets $S = \{s_1, s_2, \dots, s_k\} \subseteq (0,1]$, there exists a constant $b \in \mathbb{R}^+$, such that for all (unknown) distributions $D = \{p_1, p_2, \dots, p_k\}$ of S, allowing irrational probabilities, the following holds:

$$P\left(\text{POGC}_\varepsilon^\delta(\sigma_n(D)) \geqslant (1-\varepsilon) \cdot \textsc{Opt}(\sigma_n(D)) - b\right) \geqslant 1-\delta, \tag{5}$$

where $\sigma_n(D)$ is defined in the preliminaries. Observe that this guarantee is true, even for adversarial S and D. Clearly, Equation (5) implies that

$$P(\textsc{er}^{\infty}_{\text{POGC}_\varepsilon^\delta}(D) \geqslant 1-\varepsilon) \geqslant 1-\delta. \tag{6}$$

The guarantee from Equation (5) is, however, stronger than Equation (6), in that the additive term in Equation (5) is constant, whereas the additive term for $\text{POGC}_\varepsilon^\delta$ in Equation (6) may be a function of n. As pointed out in [6], having only constant loss before giving a multiplicative performance guarantee is a desirable property.

We formalize the strategy of $\text{POGC}_\varepsilon^\delta$ in Algorithm 3. In words; the algorithm works by defining a "sample size", Φ, as a function of k, ε and δ. Intuitively, observing Φ items from the input prefix is sufficient to make predictions about the frequency of items with respect to D that are ε-accurate with confidence $1-\delta$. We formalize this in Proposition 11. In the process of learning D, $\text{POGC}_\varepsilon^\delta$ places the first Φ items using DNF while observing the item frequencies. After placing the first Φ item, $\text{POGC}_\varepsilon^\delta$ uses the observed frequencies to make an estimate - prediction - about the item frequencies and applies $\text{GC}_{\frac{\varepsilon}{2}}$ to place the remaining items.

Algorithm 3 $\text{POGC}_\varepsilon^\delta$.

```
Input: A $\text{DBC}_S$-instance, $\sigma$
$ss \leftarrow 0$                                              ▷ Sample size
Compute $\tau_S$, $t_{\max}$, and $k = |S|$
$m_{\frac{\varepsilon}{2}} \leftarrow \lceil 6 \cdot \tau_S \cdot t_{\max} \cdot \varepsilon^{-1} \rceil$
$m_{k,\frac{\varepsilon}{2}} \leftarrow m_{\frac{\varepsilon}{2}} + k$
$\Phi \leftarrow \max\left\{16 \cdot k \cdot (m_{k,\frac{\varepsilon}{2}}+1)^2, 32 \cdot (m_{k,\frac{\varepsilon}{2}}+1)^2 \cdot \ln\left(\frac{2}{1-\sqrt{1-\delta}}\right)\right\}$
for all $i \in [k]$ do
    $c_{s_i} \leftarrow 0$                                     ▷ Number of items of size $s_i$
while receiving items, $a$, and $ss < \Phi$ do
    $c_a \leftarrow c_a + 1$
    Place $a$ in a DNF-marked bin using DNF
    $ss \leftarrow ss + 1$
for $i = 1, 2, \dots, k$ do
    $\hat{f}_i^\Phi = \frac{c_{s_i}}{\Phi}$
$\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}} = \left(\hat{f}_1^\Phi, \hat{f}_2^\Phi, \dots, \hat{f}_k^\Phi\right)$
Run Lines 2-13 of $\text{GC}_{\frac{\varepsilon}{2}}$ (see Algorithm 1), given the prediction $\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}$
while receiving items, $a$, do
    Place $a$ using $\text{GC}_{\frac{\varepsilon}{2}}$                ▷ See Lines 14-30 in Algorithm 1
```

Before formalizing and proving the claim from Equation (5), we review a PAC-learning bound for learning frequencies in discrete distributions [13].

Sampling Complexity of Learning Frequencies

We refer to [13] for a proof of the following well-known fact that establishes an upper bound for the sampling complexity of PAC-learning frequencies:

▶ **Proposition 11** ([13]). *For any finite set $S = \{s_1, s_2, \dots, s_k\} \subseteq (0,1]$, there exists an algorithm, $\mathcal{A}$, and a map $\Phi_{\mathcal{A}} \colon \mathbb{R}^+ \times (0,1) \to \mathbb{Z}^+$, such that for any $\gamma \in \mathbb{R}^+$, any $\delta \in (0,1)$, any (unknown) discrete distribution $D = \{p_1, p_2, \dots, p_k\}$ of S, and any $n \geqslant \Phi_{\mathcal{A}}(\gamma, \delta)$, letting $\{X_i\}_{i=1}^n$ be a sequence of independent identically distributed random variables, with $X_i \sim D$,*

$$P\left(L^1(\mathcal{A}(X_1, X_2, \dots, X_n), D) \leqslant \gamma\right) \geqslant 1 - \delta,$$

where L^1 is the usual L^1-distance. For learning frequencies in discrete distributions, $\mathcal{A}$ is the algorithm which outputs the predicted distribution:

$$\mathcal{A}(X_1, X_2, \dots, X_n) = \left\{ \hat{p}_i \;\middle|\; i \in [k] \text{ and } \hat{p}_i = \frac{1}{n} \cdot \sum_{j=1}^{n} \mathbb{1}_{\{s_i\}}(X_j) \right\},$$

and, for any $\gamma \in \mathbb{R}^+$ and $\delta \in (0,1)$, the map $\Phi_{\mathcal{A}}$ is given by

$$\Phi_{\mathcal{A}}(\gamma, \delta) = \max\left\{ \frac{4 \cdot k}{\gamma^2}, \frac{8}{\gamma^2} \cdot \ln\left(\frac{2}{\delta}\right) \right\}.$$

4.1 Analysis of POGC$_\varepsilon^\delta$

We formalize and prove the claim from Equation (5):

▶ **Theorem 12.** *For all finite sets $S = \{s_1, s_2, \dots, s_k\} \subset (0,1]$, and all $\varepsilon, \delta \in (0,1)$, there exists a constant $b \in \mathbb{Z}^+$, such that for all discrete distributions $D = \{p_1, p_2, \dots, p_k\}$ of S, and all $n \in \mathbb{Z}^+$, the following holds:*

$$P\left(POGC_\varepsilon^\delta(\sigma_n(D)) \geqslant (1 - \varepsilon) \cdot \textsc{Opt}(\sigma_n(D)) - b\right) \geqslant 1 - \delta,$$

where $\sigma_n(D) = \langle X_1, X_2, \dots, X_n \rangle$, and $\{X_i\}_{i=1}^n$ is a sequence of independent identically distributed random variables with $X_i \sim D$, for all $i \in [n]$.

Proof. Set $\Phi = \max\left\{16 \cdot k \cdot (m_{k,\frac{\varepsilon}{2}} + 1)^2, 32 \cdot (m_{k,\frac{\varepsilon}{2}} + 1)^2 \cdot \ln\left(\frac{2}{1 - \sqrt{1-\delta}}\right)\right\}$, and $b = \max\{2 \cdot \Phi, m_{k,\frac{\varepsilon}{2}}^2 + m_{k,\frac{\varepsilon}{2}} + \Phi\}$, and observe that b is independent of the input length n. By similar arguments as in the proof of Lemma 8, we assume that $n \geqslant b$. For ease of notation, we set $\tilde{\varepsilon} = \frac{\varepsilon}{2}$.

Throughout this proof, we split $\sigma_n(D)$ into two subsequences, σ_a and σ_s. Formally, we set $\sigma_a = \langle X_1, X_2, \dots, X_\Phi \rangle$, and $\sigma_s = \langle X_{\Phi+1}, X_{\Phi+2}, \dots, X_n \rangle$. By construction, POGC$_\varepsilon^\delta$ uses DNF on the first Φ items while counting the number of items of each size. After observing the first Φ items, it creates the predicted distribution $\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}} = \mathcal{A}(X_1, X_2, \dots, X_\Phi)$, by Lines 13-15 in Algorithm 3. By construction of Φ and Proposition 11, we can write

$$P\left(L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, D) \leqslant \frac{1}{2 \cdot (m_{k,\tilde{\varepsilon}} + 1)}\right) \geqslant \sqrt{1 - \delta}.$$

Therefore, by construction of $\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}$ and the definition of L^1, the following holds:

$$P\left(\sum_{i=1}^{k} \left|\hat{f}_i^\Phi - p_i\right| \leqslant \frac{1}{2 \cdot (m_{k,\tilde{\varepsilon}} + 1)}\right) \geqslant \sqrt{1 - \delta}.$$

Denote by $\boldsymbol{f}^{\boldsymbol{\sigma_s}}$ the true frequencies of $\sigma_s = \langle X_{\Phi+1}, X_{\Phi+2}, \ldots, X_n \rangle$. Since $n \geqslant 2 \cdot \Phi$, we know that $|\sigma_s| \geqslant \Phi$, and so, by similar arguments as above,

$$P\left(\sum_{i=1}^{k} |f_i^{\sigma_s} - p_i| \leqslant \frac{1}{2 \cdot (m_{k,\tilde{\varepsilon}} + 1)}\right) \geqslant \sqrt{1-\delta}.$$

Let $E_{\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}}$ be the event $\sum_{i=1}^{k} \left|\hat{f}_i^{\Phi} - p_i\right| \leqslant \frac{1}{2 \cdot (m_{k,\tilde{\varepsilon}}+1)}$, and $E_{\boldsymbol{f}^{\boldsymbol{\sigma_s}}}$ be the event $\sum_{i=1}^{k} |f_i^{\sigma_s} - p_i| \leqslant \frac{1}{2 \cdot (m_{k,\tilde{\varepsilon}}+1)}$. Since $E_{\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}}$ and $E_{\boldsymbol{f}^{\boldsymbol{\sigma_s}}}$ are independent, we have $P\left(E_{\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}} \textbf{ and } E_{\boldsymbol{f}^{\boldsymbol{\sigma_s}}}\right) \geqslant 1 - \delta$ Therefore, with probability at least $1 - \delta$, we have

$$L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) = \sum_{i=1}^{k} \left|\hat{f}_i^{\Phi} - f_i^{\sigma_s}\right| \leqslant \sum_{i=1}^{k} \left|\hat{f}_i^{\Phi} - p_i\right| + \sum_{i=1}^{k} |f_i^{\sigma_s} - p_i| < \frac{1}{m_{k,\tilde{\varepsilon}}}. \tag{7}$$

This means that the predictions $\text{POGC}_{\varepsilon}^{\delta}$ creates are very close to the true frequencies of the remainder of the instance, σ_s, with high probability.

Next, by construction of $\text{POGC}_{\varepsilon}^{\delta}$, we deduce that $\text{POGC}_{\varepsilon}^{\delta}(\sigma_n(D)) \geqslant \text{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}})$. Then, as long as we can verify that the inequality

$$\text{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}) \geqslant (1-\varepsilon) \cdot \text{OPT}(\sigma_s), \tag{8}$$

holds whenever $L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) < \frac{1}{m_{k,\tilde{\varepsilon}}}$, we deduce that

$$\begin{aligned}\text{POGC}_{\varepsilon}^{\delta}(\sigma_n(D)) &\geqslant \text{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}) \\ &\geqslant (1-\varepsilon) \cdot \text{OPT}(\sigma_s) \\ &\geqslant (1-\varepsilon) \cdot \text{OPT}(\sigma_n(D)) - 2 \cdot \Phi.\end{aligned}$$

Since $P(L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) < \frac{1}{m_{k,\tilde{\varepsilon}}}) \geqslant 1 - \delta$, by Equality 7, we can write

$$P\left(\text{POGC}_{\varepsilon}^{\delta}(\sigma_n(D)) \geqslant (1-\varepsilon) \cdot \text{OPT}(\sigma_n(D)) - 2 \cdot \Phi\right) \geqslant 1 - \delta,$$

which completes the proof.

It remains to prove that Equation (8) holds whenever $L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) < \frac{1}{m_{k,\tilde{\varepsilon}}}$. To this end, assume that $L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) < \frac{1}{m_{k,\tilde{\varepsilon}}}$. Let $g_{\tilde{\varepsilon}}$ be the number of groups that $\text{GC}_{\tilde{\varepsilon}}$ would complete on instance $(\sigma_s, \boldsymbol{f}^{\boldsymbol{\sigma_s}})$, that is, with perfect predictions. Moreover, let $P_{\sigma_s,\tilde{\varepsilon}} = \text{OPT}[\langle \lfloor f_1^{\sigma_s} \cdot m_{k,\tilde{\varepsilon}} \rfloor, \ldots, \lfloor f_k^{\sigma_s} \cdot m_{k,\tilde{\varepsilon}} \rfloor \rangle]$, and $P_{\Phi,\tilde{\varepsilon}} = \text{OPT}[\langle \lfloor \hat{f}_1^{\Phi} \cdot m_{k,\tilde{\varepsilon}} \rfloor, \ldots, \lfloor \hat{f}_k^{\Phi} \cdot m_{k,\tilde{\varepsilon}} \rfloor \rangle]$, where items have been replaced with placeholders.

First, we compare the number of items of size s_i in $P_{\sigma_s,\tilde{\varepsilon}}$ compared to $P_{\Phi,\tilde{\varepsilon}}$. To this end, for all $i \in [k]$, set $\mu_i = \left|\lfloor \hat{f}_i^{\Phi} \cdot m_{k,\tilde{\varepsilon}} \rfloor - \lfloor f_i^{\sigma_s} \cdot m_{k,\tilde{\varepsilon}} \rfloor\right|$. Then,

$$\mu_i \leqslant \left|\hat{f}_i^{\Phi} \cdot m_{k,\tilde{\varepsilon}} - f_i^{\sigma_s} \cdot m_{k,\tilde{\varepsilon}}\right| + 1 = \left|\hat{f}_i^{\Phi} - f_i^{\sigma_s}\right| \cdot m_{k,\tilde{\varepsilon}} + 1.$$

Since $L^1(\hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}, \boldsymbol{f}^{\boldsymbol{\sigma_s}}) < \frac{1}{m_{k,\tilde{\varepsilon}}}$, we get that $\sum_{i=1}^{k} \left|\hat{f}_i^{\Phi} - f_i^{\sigma_s}\right| < \frac{1}{m_{k,\tilde{\varepsilon}}}$, which implies that $\left|\hat{f}_i^{\Phi} - f_i^{\sigma_s}\right| < \frac{1}{m_{k,\tilde{\varepsilon}}}$, for all $i \in [k]$. Therefore, we have $\mu_i < 2$ for all $i \in [k]$, and since $\mu_i \in \mathbb{N}$, we get that $\mu_i \in \{0, 1\}$, for all $i \in [k]$.

Next, we lower bound $\text{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}})$, as a function of $\mathbf{p}(P_{\Phi,\tilde{\varepsilon}})$ and $g_{\tilde{\varepsilon}}$. Since $\text{GC}_{\tilde{\varepsilon}}$ would complete $g_{\tilde{\varepsilon}}$ groups on instance $(\sigma_s, \boldsymbol{f}^{\boldsymbol{\sigma_s}})$, then, for all $i \in [k]$, σ_s contains at least $g_{\tilde{\varepsilon}} \cdot \lfloor f_i^{\sigma_s} \cdot m_{k,\tilde{\varepsilon}} \rfloor$ items of size s_i. Since $\mu_i \in \{0, 1\}$ for all $i \in [k]$, then, on instance $(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}})$, $\text{GC}_{\tilde{\varepsilon}}$ fills all placeholders of size s_i in $g_{\tilde{\varepsilon}}$ groups, except at most $g_{\tilde{\varepsilon}}$. Hence,

$$\text{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}) \geqslant g_{\tilde{\varepsilon}} \cdot \mathbf{p}(P_{\Phi,\tilde{\varepsilon}}) - g_{\tilde{\varepsilon}} \cdot k.$$

For the rest of this proof, we use an argument as in the proof of Theorem 5. To this end, let N be the covering obtained by creating a copy of $\textsc{Opt}[\sigma_s]$, from which we have removed a number of bins of type $t \in \mathcal{T}_S$, such that the number of bins of type t is divisible by $g_{\tilde{\varepsilon}}$, for all $t \in \mathcal{T}_S$. By similar arguments as in Lemma 4, we get that $\mathbf{p}(N) \geqslant (1 - \frac{\tilde{\varepsilon}}{3}) \cdot \textsc{Opt}(\sigma_s)$.

Next, observe that N is comprised of $g_{\tilde{\varepsilon}}$ identical coverings $\overline{N}$. Since $n \geqslant b$, we can write $|\sigma_s| \geqslant m_{k,\tilde{\varepsilon}}^2 + m_{k,\tilde{\varepsilon}}$. Hence, by a similar argument as in the proof of Lemma 4, we have $n_i^{\overline{N}} \leqslant n_i^{P_{\sigma_s,\tilde{\varepsilon}}} + 1 \leqslant n_i^{P_{\Phi,\tilde{\varepsilon}}} + 2$, for all $i \in [k]$, and thus $\mathbf{p}(P_{\Phi,\tilde{\varepsilon}}) \geqslant \mathbf{p}\big(\overline{N}\big) - 2 \cdot k$. Moreover, as in Lemma 4, it holds that $k \leqslant \frac{\frac{\tilde{\varepsilon}}{3} \cdot \mathbf{p}(\overline{N})}{1-\frac{\tilde{\varepsilon}}{3}}$, and we can write

$$\mathbf{p}(P_{\Phi,\tilde{\varepsilon}}) \geqslant \mathbf{p}\big(\overline{N}\big) - 2 \cdot \frac{\frac{\tilde{\varepsilon}}{3} \cdot \mathbf{p}\big(\overline{N}\big)}{1 - \frac{\tilde{\varepsilon}}{3}} \geqslant (1 - \tilde{\varepsilon}) \cdot \mathbf{p}\big(\overline{N}\big) \,.$$

Conclusively, from the above-established inequalities, we can conclude the following, which completes the proof:

$$\begin{aligned}
\mathrm{GC}_{\tilde{\varepsilon}}(\sigma_s, \hat{\boldsymbol{f}}^{\boldsymbol{\Phi}}) &\geqslant g_{\tilde{\varepsilon}} \cdot (\mathbf{p}(P_{\Phi,\tilde{\varepsilon}}) - k) \geqslant g_{\tilde{\varepsilon}} \cdot \left((1 - \tilde{\varepsilon}) \cdot \mathbf{p}\big(\overline{N}\big) - \frac{\frac{\tilde{\varepsilon}}{3} \cdot \mathbf{p}\big(\overline{N}\big)}{1 - \frac{\tilde{\varepsilon}}{3}} \right) \\
&\geqslant g_{\tilde{\varepsilon}} \cdot \left(1 - \frac{5}{3} \cdot \tilde{\varepsilon}\right) \cdot \mathbf{p}\big(\overline{N}\big) \geqslant \left(1 - \frac{5}{3} \cdot \tilde{\varepsilon}\right) \cdot \left(1 - \frac{\tilde{\varepsilon}}{3}\right) \cdot \textsc{Opt}(\sigma_s) \\
&= (1 - 2 \cdot \tilde{\varepsilon}) \cdot \textsc{Opt}(\sigma_s) = (1 - \varepsilon) \cdot \textsc{Opt}(\sigma_s).
\end{aligned}$$ ◀

5 Concluding Remarks

We studied the power of frequency predictions in improving the performance of online algorithms for the discrete bin cover problem. In particular, we showed that when input is adversarially generated, frequency predictions (from historical data) can help design algorithms with adjustable trade-offs between consistency and robustness. Specifically, one can achieve near-optimal solutions, assuming predictions are error-free. On the other hand, when input is generated stochastically, we showed that frequencies could be learned from an input prefix of constant length to achieve solutions that are arbitrarily close to optimal with arbitrarily high confidence. An interesting variant of the problem concerns inputs generated adversarially but permuted randomly. This setting is in line with recent work on the analysis of algorithms with random order input (see, e.g., [21, 7]). We expect that our algorithm for the stochastic setting can still be applied to this setting to achieve close to optimal solutions with high confidence, although a different analysis is needed.

References

1 Algorithms with predictions. URL: `https://algorithms-with-predictions.github.io/`. Accessed: 2024-01-26.

2 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 52:1–52:15. Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ITCS.2020.52`.

3 Spyros Angelopoulos and Shahin Kamali. Contract scheduling with predictions. *J. Artif. Intell. Res.*, 77:395–426, 2023.

4 Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. *Journal of Artificial Intelligence Research*, 78:1111–1141, 2023. `doi:10.1613/jair.1.14820`.

5 Susan F. Assmann, David S. Johnson, Daniel J. Kleitman, and Josepth Y.-T. Leung. On a dual version of the one-dimensional packing problem. *Journal of Algorithms*, 5:502–525, 1984. doi:10.1016/0196-6774(84)90004-X.

6 Siddhartha Banerjee and Daniel Freund. Uniform loss algorithms for online stochastic decision-making with applications to bin packing. In *Abstracts of the 2020 SIGMETRICS*, pages 1–2. ACM, 2020. doi:10.1145/3393691.3394224.

7 Magnus Berg, Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online minimum spanning trees with weight predictions. In *Algorithms and Data Structures (WADS)*, pages 136–148. Springer, Cham, 2023. doi:10.1007/978-3-031-38906-1_10.

8 Magnus Berg and Shahin Kamali. Online bin covering with frequency predictions, 2024. arXiv:2401.14881.

9 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA, 1998.

10 Joan Boyar, Lene M. Favrholdt, Shahin Kamali, and Kim S. Larsen. Online bin covering with advice. *Algorithmica*, 83:795–821, 2021. doi:10.1007/s00453-020-00728-0.

11 Joan Boyar, Lene M. Favrholdt, Shahin Kamali, and Kim S. Larsen. Online interval scheduling with predictions. In *18th International Symposium on Algorithms and Data Structures (WADS)*, volume 14079, pages 193–207, 2023.

12 Andrej Brodnik, Bengt J. Nilsson, and Gordana Vujovic. Online bin covering with exact parameter advice. *arXiv:2309.13647*, 2023. doi:10.48550/arXiv.2309.13647.

13 Clément L. Canonne. A short note on learning discrete distributions. *arXiv:2002.11457*, 2020. doi:10.48550/arXiv.2002.11457.

14 János Csirik, J. B. G. Frenk, Martine Labbé, and Shuzhong Zhang. Two simple algorithms for bin covering. *Acta Cybernetica*, 14:13–25, 1999.

15 János Csirik, David S. Johnson, and Claire Kenyon. Better approximation algorithms for bin covering. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 557–566. SIAM, 2001.

16 János Csirik, David S. Johnson, Claire Kenyon, James B. Orlin, Peter W. Shor, and Richard R. Weber. On the sum-of-squares algorithm for bin packing. *Journal of the ACM*, 53:1–65, 2006. doi:10.1145/1120582.1120583.

17 János Csirik, David S. Johnson, Claire Kenyon, Peter W. Shor, and Richard R. Weber. A self organizing bin packing heuristic. In *Algorithms Engeneering and Experimentation (ALENEX)*, pages 250–269. Springer, 1999. doi:10.1007/3-540-48518-X_15.

18 János Csirik and Vilmos Totik. Online algorithms for a dual version of bin packing. *Discrete Applied Mathematics*, 21:163–167, 1988. doi:10.1016/0166-218X(88)90052-2.

19 János Csirik and Gerhard H. Woeginger. On-line packing and covering problems. In *Online Algorithms*, pages 147–177. Springer, Berlin, Heidelberg, 2005. doi:10.1007/BFb0029568.

20 Leah Epstein. Online variable sized covering. *Information and Computation*, 171:294–305, 2001. doi:10.1006/inco.2001.3087.

21 Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1253–1264. IEEE, 2021. doi:10.1109/FOCS52979.2021.00122.

22 Klaus Jensen and Roberto Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theoretical Computer Science*, 306:543–551, 2003. doi:10.1016/S0304-3975(03)00363-3.

23 Dennis Komm. *An Introduction to Online Computation: Determinism, Randomization, Advice*. Springer Cham, Switzerland, 2016. doi:10.1007/978-3-319-42749-2.

24 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68:1–25, 2021. doi:10.1145/3447579.

25 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 9684–9693. Curran Associates, Inc., 2018.

26 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning Theory: From Theory to Algorithms*. Cambridge University Press, Cambridge, England, 2014. `doi:10.1017/CBO9781107298019`.

27 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, pages 8042–8053. Curran Associates, Inc., 2020.

Subexponential Algorithms in Geometric Graphs via the Subquadratic Grid Minor Property: The Role of Local Radius

Gaétan Berthe
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Marin Bougeret
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Daniel Gonçalves
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Jean-Florent Raymond
Univ Lyon, EnsL, CNRS, LIP, F-69342, Lyon Cedex 07, France

Abstract

We investigate the existence in geometric graph classes of subexponential parameterized algorithms for cycle-hitting problems like Triangle Hitting (TH), Feedback Vertex Set (FVS) or Odd Cycle Transversal (OCT). These problems respectively ask for the existence in a graph G of a set X of at most k vertices such that $G - X$ is triangle-free, acyclic, or bipartite. It is know that subexponential FPT algorithms of the form $2^{o(k)}n^{\mathcal{O}(1)}$ exist in planar and even H-minor free graphs from bidimensionality theory [Demaine et al. 2005], and there is a recent line of work lifting these results to geometric graph classes consisting of intersection of similarly sized "fat" objects ([Fomin et al. 2012], [Grigoriev et al. 2014], or disk graphs [Lokshtanov et al. 2022], [An et al. 2023]).

In this paper we first identify sufficient conditions, for any graph class $\mathcal{C}$ included in string graphs, to admit subexponential FPT algorithms for any problem in $\mathcal{P}$, a family of bidimensional problems where one has to find a set of size at most k hitting a fixed family of graphs, containing in particular FVS. Informally, these conditions boil down to the fact that for any $G \in \mathcal{C}$, the *local radius of* G (a new parameter introduced in [Lokshtanov et al. 2023]) is polynomial in the clique number of G and in the maximum matching in the neighborhood of a vertex. To demonstrate the applicability of this generic result, we bound the local radius for two special classes: intersection graphs of axis-parallel squares and of contact graphs of segments in the plane. This implies that any problem $\Pi \in \mathcal{P}$ (in particular, FVS) can be solved in:

- $2^{\mathcal{O}(k^{3/4}\log k)}n^{\mathcal{O}(1)}$-time in contact segment graphs,
- $2^{\mathcal{O}(k^{9/10}\log k)}n^{\mathcal{O}(1)}$ in intersection graphs of axis-parallel squares

On the positive side, we also provide positive results for TH by solving it in:

- $2^{\mathcal{O}(k^{3/4}\log k)}n^{\mathcal{O}(1)}$-time in contact segment graphs,
- $2^{\mathcal{O}(\sqrt{d}t^2(\log t)k^{2/3}\log k)}n^{\mathcal{O}(1)}$-time in $K_{t,t}$-free d-DIR graphs (intersection of segments with d slopes)

On the negative side, assuming the ETH we rule out the existence of algorithms solving:

- TH and OCT in time $2^{o(n)}$ in 2-DIR graphs and more generally in time $2^{o(\sqrt{\Delta n})}$ in 2-DIR graphs with maximum degree Δ, and
- TH, FVS, and OCT in time $2^{o(\sqrt{n})}$ in $K_{2,2}$-free contact-2-DIR graphs of maximum degree 6.

Observe that together, these results show that the absence of large $K_{t,t}$ is a necessary and sufficient condition for the existence of subexponential FPT algorithms for TH in 2-DIR.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Fixed parameter tractability; Theory of computation → Computational geometry

Keywords and phrases geometric intersection graphs, subexponential FPT algorithms, cycle-hitting problems, bidimensionality

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.11

Related Version *Full Version*: `https://arxiv.org/abs/2306.17710`

Funding *Jean-Florent Raymond*: Supported by the project GRALMECO (ANR-21-CE48-0004).

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 11; pp. 11:1–11:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper we consider fundamental NP-hard cycle-hitting problems like Triangle Hitting (TH), Feedback Vertex Set (FVS), and Odd Cycle Transversal (OCT) where, given a graph G and an integer k, the goal is to decide whether G has a set of at most k vertices hitting all its triangles (resp. cycles for FVS, and odd cycles for OCT). We consider these problems from the perspective of parameterized complexity, where the objective is to answer in time $f(k)n^{\mathcal{O}(1)}$ for some computable function f, and with n denoting the order of G. It is known (see for instance [12]) that these three problems can be solved on general graphs in time $c^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ (for some constant c) and that, under the Exponential Time Hypothesis (ETH), the contribution of k cannot be improved to a subexponential function (i.e., there are no algorithms with running times of the form $c^{o(k)}n^{\mathcal{O}(1)}$ for these problems). However, it was discovered that some problems admit subexponential time algorithms in certain classes of graphs, and there is now a well established set of techniques to design such algorithms. Let us now review these techniques and explain why they do not apply on the problems we consider here.

Subexponential FPT algorithms in sparse graphs. Let us start with the bidimensionality theory, which gives an explanation on the so-called *square root phenomenon* arising for planar and H-minor free graphs [14] for bidimensional[1] problems, where a lot of graph problems admit ETH-tight $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ algorithms. What we call a *graph parameter* here is a function p mapping any (simple) graph to a natural number and that is invariant under isomorphism. The classical win-win strategy to decide if $p(G) \leq k$ for a minor-bidimensional[2] parameter (like $p = \mathsf{fvs}$, the size of a minimum feedback vertex set of G) is to first reduce to the case where $\boxplus(G) = \mathcal{O}(\sqrt{k})$ (where $\boxplus(G)$ denotes the maximum k such that the (k,k)-grid is contained as a minor in G), and then use an inequality of the form $\mathsf{tw}(G) \leq f(\boxplus(G))$ to bound the treewidth obtained through the following property.

▶ **Definition 1** ([4]). *Given $c < 2$, a graph class $\mathcal{G}$ has the* subquadratic grid minor property *for c (*SQGM *for short), denoted $\mathcal{G} \in SQGM(c)$, if $\mathsf{tw}(G) = \mathcal{O}(\boxplus(G)^c)$ for all $G \in \mathcal{G}$. We write $\mathcal{G} \in SQGM$ if there exists $c < 2$ such that $\mathcal{G} \in SQGM(c)$.*

While in general every graph G satisfies the inequality $\mathsf{tw}(G) \leq \boxplus(G)^c$ for some $c < 10$ [11], the SQGM property additionally require that $c < 2$. Thus, for any $\mathcal{G} \in SQGM(c)$ and $G \in \mathcal{G}$ such that $\boxplus(G) = \mathcal{O}(\sqrt{k})$, we get $\mathsf{tw}(G) \leq \boxplus(G)^c = \mathcal{O}\left(k^{c/2}\right) = o(k)$. For instance planar graphs and more generally H-minor free graph [15] are known to have a treewidth linearly bounded from above by the size of their largest grid minor. In other words, these classes belong to $SQGM(1)$. The conclusion is that the SQGM property allows subexponential parameterized algorithms for minor-bidimensional problems (if the considered problem has a $2^{\mathcal{O}(\mathsf{tw}(G))}n^{\mathcal{O}(1)}$-time algorithm) on sparse graph classes. Notice that these techniques have been extended to contraction-bidimensional problems [4].

Extension to geometric graphs. Consider now a geometric graph class $\mathcal{G}$, meaning that any $G \in \mathcal{G}$ represents the interactions of some specified geometric objects. We consider here (Unit) Disk Graphs which correspond to intersection of (unit) disks in the plane, d-DIR graphs (where the vertices correspond to segments with d possible slopes in $\mathbb{R}^2$), and contact-segment

[1] Informally: yes-instances are minor-closed and a solution on the (r,r)-grid has size $\Omega(r^2)$.
[2] See definition in [20].

graphs (where each vertex corresponds to a segment in $\mathbb{R}^2$, and any intersection point between two segments must be an endpoint of one of them). We refer to Subsection 2.2 for formal definitions. Classes of geometric graphs represented in the plane form an appealing source of candidates to obtain subexponential parameterized algorithms as there is an underlying planarity in the representation. However these graphs are no longer sparse as they may contain large cliques, and thus cannot have the SQGM property. Indeed, if G is a clique of size a, then $\mathsf{tw}(G) = a - 1$ but $\boxplus(G) \leq \sqrt{|G|} = \sqrt{a}$. To overcome this, let us introduce the following notion where the bound on treewidth is allowed to depend on an additional parameter besides $\boxplus(G)$.

▶ **Definition 2.** *Given a graph parameter p and a real $c < 2$, a graph class $\mathcal{G}$ has the* almost subquadratic grid minor property *(*ASQGM *for short) for p and c if there exists a function f such that $\mathsf{tw}(G) = \mathcal{O}(f(p(G))\boxplus(G)^c)$. The class $\mathcal{G}$ has ASQGM(p) if there exists $c < 2$ such that $\mathcal{G}$ has the ASQGM property for p and c. The notation is naturally extended to more than one parameter.*

This notion was used implicitly in earlier work (e.g., [20]) but we chose to define it explicitly in order to highlight the contribution f of the parameter p to the treewidth, which is particularly relevant when it can be shown to be small (typically, polynomial). Let us now explain how ASQGM can be used to obtain subexponential parameterized algorithms on geometric graphs.

It was shown in [19] that FVS can be solved in time $2^{\mathcal{O}(k^{3/4}\log k)}n^{\mathcal{O}(1)}$ in map graphs, a superclass of planar graphs where arbitrary large cliques may exist, as follows. Let $\omega(G)$ denote the order of the largest clique in a graph G. The first ingredient is to prove that map graphs have *ASQGM*(ω), and more precisely that $\mathsf{tw}(G) = \mathcal{O}(\omega(G)\boxplus(G))$. Then, if $\omega(G) \geq k^{\epsilon}$ for some ϵ, the presence of such large clique allows to have subexponential branchings (as a solution of FVS must take almost all vertices of a clique). When $\omega(G) < k^{\epsilon}$, then the ASQGM property gives that $\mathsf{tw}(G) \leq k^{\epsilon}\boxplus(G) \leq k^{\frac{1}{2}+\epsilon}$ (as before we can immediately answer no if $\boxplus(G) > \mathcal{O}(\sqrt{k})$). By appropriately choosing ϵ the authors of [19] obtain the mentioned running time. The same approach also applies to unit disk graphs and has since been improved to $2^{\sqrt{k}\log k}n^{\mathcal{O}(1)}$ in [17] using a different technique, and finally improved to an optimal $2^{\sqrt{k}}(n + m)$ in [2] for similarly sized fat objets (which typically includes unit squares, but not disks, squares, nor segments).

There is also a line of work aiming at establishing ASQGM property for different classes of graphs and parameters, with for example [20] proving that (1) string graphs have ASQGM when the parameter p is the number of times a string is intersected (assuming at most two strings intersect at the same point), and that (2) intersection graphs of “fat” and convex objects have ASQGM when the parameter $p(G)$ is the minimal order of a graph H not subgraph of G (generalizing the degree when H is a star).

When *ASQGM*(ω) does not hold. A natural next step for FVS and TH is to consider classes that are not *ASQGM*(ω). Observe (see Figure 1) that neither disk graphs, nor contact-2-DIR graphs are in *ASQGM*(ω), and thus constitute natural candidates.

New ideas allowed the authors of [23] to obtain subexponential parameterized algorithms on disk graphs, in particular for TH and FVS. The first idea is a preliminary branching step (working on general graphs) which given an input (G, k) first reduces to the case where we are given a set M of size $\mathcal{O}(k^{1+\epsilon})$ such that $G - M$ is a forest and, for any $v \in M$, $N(v) \setminus M$ is an independent set (corresponding to Corollary 7, but where we consider a generic problem instead of FVS). The second idea is related to neighborhood complexity. If for a graph class $\mathcal{G}$ there is a constant c such that for every $G \in \mathcal{G}$ and every $X \subseteq V(G)$,

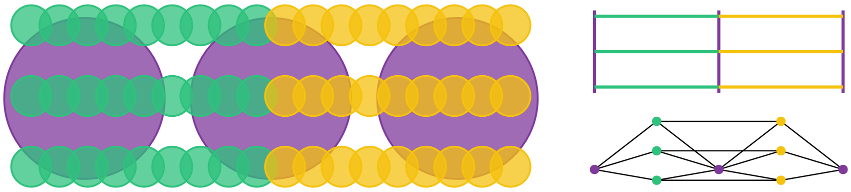


Figure 1 Left: a representation of a disk graph. Right: a contact 2-DIR graph and the corresponding graph. In these graphs (where the left one is from [19]), $\omega(G)$ is constant, $\mathsf{tw}(G) \geq t$ (where $t = 3$ here) as it contains $K_{t,t}$ as a minor, and $\boxplus(G) = \mathcal{O}(\sqrt{t})$ as they have a feedback vertex set of size at most t.

$|\{N(v) \cap X : v \in V(G)\}| \leq c|X|$, then we say that $\mathcal{G}$ has *linear neighborhood complexity* with *ratio* c. The following theorem was originally formulated using ply (the maximum number of disks containing a fixed point) instead of clique number, but it is known [7] that these two values are linearly related in disk graphs.

▶ **Theorem 3** (Theorem 1.1 in [23]). *Disk graphs with bounded clique number have linear neighborhood complexity.*

For TH, these two ideas are sufficient to obtain a subexponential parameterized algorithm. For FVS, [23] provides the following corollary.

▶ **Corollary 4** (Corollary 1.1 in [23] restricted to FVS). *Let G be a disk graph with a (non-necessarily minimal) feedback vertex set $M \subseteq V(G)$ such that for all $v \in M$, $N(v) \setminus M$ is an independent set, and such that for all $v \in V(G) \setminus M$, $N(v) \setminus M$ is non-empty. Then, the treewidth of G is $\mathcal{O}(\sqrt{|M|}\omega(G)^{2.5})$.*

As they use this corollary after a branching process reducing the clique number to k^{ϵ} and as their (approximated) feedback vertex set M has size $|M| = k^{1+\epsilon'}$, they obtain a sublinear treewidth and thus a subexponential parameterized algorithm for FVS (and several variants of FVS) running in time $2^{\mathcal{O}(k^{13/14} \log k)}n^{\mathcal{O}(1)}$. Recently this running time has been improved to $2^{\mathcal{O}(k^{7/8} \log k)}n^{\mathcal{O}(1)}$ when the representation is given and $2^{\mathcal{O}(k^{9/10} \log k)}n^{\mathcal{O}(1)}$ otherwise [1]. We point out that it is likely that the algorithms of [23] and [1] solving FVS in disk graphs with the respective running times $2^{\mathcal{O}(k^{13/14} \log k)}n^{\mathcal{O}(1)}$ and $2^{\mathcal{O}(k^{9/10} \log k)}n^{\mathcal{O}(1)}$, can be adapted[3] to the setting of square graphs, the later matching our bound.

Subexponential FPT algorithms via kernels. Another approach to obtain $2^{o(k)}n^{\mathcal{O}(1)}$ algorithms is to obtain small kernels (meaning computing in polynomial time an equivalent instance (G', k') with $|G'|$ typically in $\mathcal{O}(k)$), and then use a $2^{o(n)}$ time algorithm. For FVS such a $2^{o(n)}$-time algorithm is known in string graphs from [9] or [25], and was recently generalized to induced-minor-free graph classes [22]. However, as far as we are aware, the existence of a subquadratic kernel in this graph class is currently open.

1.1 Our contribution

Our objective is to study the existence of subexponential parameterized algorithms for hitting problems like FVS and TH in different types of intersection graphs. Our algorithmic results are summarized in Table 1.

[3] Regarding the algorithm of [1], it would be true if their lemma to bound the number of what they call "deep vertices" can be extended to square graphs.

Table 1 Summary of our results. All algorithms are robust, i.e., they do not need a representation.

Upper bounds				
Restriction	of class	Problem	Time complexity	Section
none	square graphs	$\Pi \in \mathcal{P}$	$2^{\mathcal{O}(k^{9/10} \log k)} n^{\mathcal{O}(1)}$	Section 3
contact	segment graphs		$2^{\mathcal{O}(k^{7/8} \log k)} n^{\mathcal{O}(1)}$	Full version
		TH	$2^{\mathcal{O}(k^{3/4} \log k)} n^{\mathcal{O}(1)}$	
$K_{t,t}$-free	d-DIR graphs		$2^{\mathcal{O}(k^{2/3} (\log k) \sqrt{d} t^2 \log t)} n^{\mathcal{O}(1)}$	
	string graphs		$2^{\mathcal{O}_t(k^{2/3} \log k)} n^{\mathcal{O}(1)}$	

Lower bounds (under ETH)				
Restriction	of class	Problem	Lower bound	Section
none	2-DIR	TH, OCT	$2^{o(n)}$	Section 4
Maximum degree Δ, for $\Delta \geq 6$			$2^{o\left(\sqrt{\Delta n}\right)}$	Full version
$K_{2,2}$-free contact, max degree 6		TH, FVS, OCT	$2^{o\left(\sqrt{n}\right)}$	

Positive results via ASQGM. In Section 3 we explain how the local radius (hereafter denoted lr), introduced recently in [24] in the context of approximation, can be used to get subexponential FPT algorithms for any problem in $\mathcal{P}$, a family of bidimensional problems where one has to find a set of size at most k hitting a fixed family of graphs. This class contains in particular FVS, and Pseudo Forest Del (resp. P_t-Hitting) where given a graph G, the goal is to remove a set S of at most k vertices of G such that each connected component of $G - S$ contains at most one cycle (resp. does not contain a path on t vertices as a subgraph). We point out that these three problems are also in the list of problems mentioned in [24] that admit EPTAS in disk graphs. We first provide sufficient conditions for graph class to admit subexponential FPT algorithms for any problem in $\mathcal{P}$, after the preprocessing step of Corollary 7 (introduced for disk graphs in [23]) has been performed. These conditions mainly boil down to having $ASQGM(\omega, \mu^{\mathrm{N}^\star})$, where $\mu^{\mathrm{N}^\star}$ is, informally, the maximum size of matching in the neighborhood of a vertex. Then, we use the framework of [4] to show that string graphs have $ASQGM(\omega, \mathrm{lr})$. Thus, the message of Section 3 is that in order to obtain a subexponential FPT algorithm for a problem $\Pi \in \mathcal{P}$ in a given subclass of string graphs, the only challenge is to bound lr by a polynomial of ω and $\mu^{\mathrm{N}^\star}$. Finally, we provide such bounds for square graphs (intersection of axis-parallel squares) and contact-segment graphs.

We point out that in our companion paper [5] we prove that FVS admits an algorithm running in time $2^{\mathcal{O}(k^{10/11} \log k)} n^{\mathcal{O}(1)}$ for pseudo-disk graphs. As square and segment graphs are in particular pseudo-disk graphs, this generalizes the graph class where subexponential parameterized algorithms exist, but to the price of a worst running time. Moreover, our result in [5] is obtained via kernelization techniques which require a representation of the input graph (i.e., this algorithm is not robust), and the reduction rules behind the kernel are tailored for FVS and not applicable for any problem $\Pi \in \mathcal{P}$.

Negative results. An interesting difference between disk graphs and d-DIR graphs is that Theorem 3 (about the linear neighborhood complexity) no longer holds for d-DIR graphs, because of the presence of large bicliques. Thus, it seems that $K_{t,t}$ is an important subgraph differentiating the two settings and this fact is confirmed by the two following results. First we show (see sketch in Section 4 and full proof in the full version of the paper) in that assuming the ETH, there is no algorithm solving TH and OCT in time $2^{o(n)}$ on n-vertex 2-DIR graphs and more generally in time $2^{o(\sqrt{\Delta n})}$ in 2-DIR graphs with maximum degree Δ. We note that the result for OCT was already proved in [26] as a consequence of algorithmic lower bounds for homomorphisms problems in string graphs. In our second negative result, we prove that assuming the ETH, the problems TH, OCT, and FVS cannot be solved in time $2^{o(\sqrt{n})}$ on n-vertex $K_{2,2}$-free contact-2-DIR graphs. Notice that that our $2^{o(\sqrt{n})}$ lower-bounds match those known for the same problems in planar graphs [10].

Positive results for TH. In the full version of the paper we observe that, for any hereditary graph class with sublinear separators, the preliminary branching step in Corollary 7 of [23] directly leads to a subexponential parameterized algorithm for TH. This implies the $2^{c_t k^{2/3} \log k} n^{\mathcal{O}(1)}$ algorithm for $K_{t,t}$-free string graphs. Recall that according to our negative result in the full version of the paper, the $K_{t,t}$-free assumption is necessary. To improve the constant c_t in special cases, we provide in the full version of the paper bounds on the neighborhood complexity of two subclasses that may be of independent interest: $K_{t,t}$-free d-DIR graphs have linear neighborhood complexity with ratio $\mathcal{O}(dt^3 \log t)$, and contact-segment graphs have linear neighborhood complexity. These bounds lead to improved running times for TH in the corresponding graph classes (see Table 1).

Due to space constraints, the proofs of the statements marked with the ✂ symbol have been deferred to the full version [6].

2 Preliminaries

2.1 Basics

In this paper logarithms are binary and all graphs are simple, loopless and undirected. Unless otherwise specified we use standard graph theory terminology, as in [16] for instance. Given a graph G, we denote by $\omega(G)$ the maximum order of a clique in G. We denote by $d_G(v)$ the degree of $v \in V(G)$, or simply $d(v)$ when G is clear from the context. The *distance* between two vertices of a graph is the minimum length (in number of edges) of a path linking them, and the *diameter* of a graph is the maximum distance between two of its vertices. The *radius* of a graph is the smallest integer $r \geq 0$ such that there exists a vertex v such that every vertex in the graph is at a distance at most r from v. A *t-bundle* [24] is a matching of size t plus a vertex connected to the $2t$ vertices of the matching. We say that B is a t-bundle of a graph G if $G[B]$ is a t-bundle plus possibly some extra edges. A set $S \subseteq V(G)$ is a t-bundle hitting set of G if $S \cap B \neq \emptyset$ for any t-bundle B of G. We denote by $\boxplus(G)$ the maximum k such that the (k, k)-grid is contained as a minor in G. We denote by $\mathsf{tw}(G)$ the treewidth of G, and $\mu(G)$ the size of a maximum matching of G.

In Section 3 we provide subexponential parameterized algorithms for a class of problems $\mathcal{P}$ that we will now define. We restrict our attention to *hitting problems*, where for a fixed graph family $\mathcal{F}$, the input is a graph G and an integer k, and the goal is to decide if there exists $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S \in \mathcal{F}$. A general setting where our results hold is described by the class $\mathcal{P}$ defined below and inspired by the problems tackled in [24] .

▶ **Definition 5.** *We denote by $\mathcal{P}$ the class of all hitting problems Π such that:*

1. *Π is bidimensional ;*
2. *there is an integer $c_\Pi > 0$ such that for any solution S in a graph G, and any c_Π-bundle B of G, $S \cap B \neq \emptyset$; and*
3. *Π can be solved on a graph G in time $\mathsf{tw}(G)^{\mathcal{O}(\mathsf{tw}(G))}$.*

▷ Claim 6. FVS, Pseudo Forest Del and P_t-Hitting for $t \leq 5$ belong to $\mathcal{P}$.

Proof. It is well known that these three problems are bidimensional. For the second condition, one can check that c_Π is equal to 1 for FVS (as a 1-bundle is a triangle) and equal to 2 for Pseudo Forest Del and P_t-Hitting when $t \leq 5$. For the last condition, as FVS corresponds to hit all K_3 as minor and Pseudo Forest Del correspond to hit all $\{H_0, H_1, H_2\}$ as a minor (with H_i is formed by two triangles sharing i vertices), these two problems can be solved in $\mathsf{tw}(G)^{\mathcal{O}(\mathsf{tw}(G))}$ by [3]. For P_t-Hitting the result holds by [13]. ◁

2.2 Graph classes

A summary of graph classes considered in this article is presented in Figure 2.

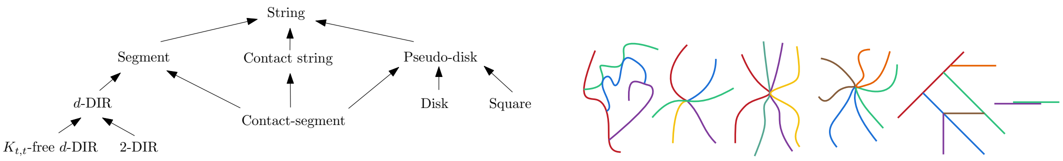


Figure 2 Left: inclusion between graph classes. Right: from left to right, four representations of contact string graphs, then a representation of 3-DIR contact-segment graph, and finally on the right an example of an intersection between segments not allowed in a representation of a contact-segment graph.

In this article, we are mainly concerned with geometric graphs described by the intersection or contact of objects in the Euclidean plane. The most general class we consider are *string graphs*, which are intersection graphs of strings (a.k.a. Jordan arcs). Intersection graphs of segments in $\mathbb{R}^2$ are called *segment graphs*. If a segment graph can be represented with at most d different slopes, we call it a *d-DIR graph*.[4] These classes of intersection graphs admit *contact* subclasses, where the representations should not contain *crossings*. That is, two strings either intersect tangentially, or they intersect at an endpoint of one of them. In a segment contact representation, any point belonging to two segments must be an endpoint of at least one of these segments. If a point belongs to several strings or segments, the above property must hold for any pair of them. This defines *contact string graphs*, *contact-segment graphs* and *contact d-DIR graphs*.

2.3 Preliminary branching steps

Our algorithms make use of the following preprocessing branching which was formulated in [23] for FVS for disk graphs. Here we restate it for any problem in $\mathcal{P}$ and for any graph class where the maximum clique can be approximated in polynomial time. A proof of this statement (included in the full version of the paper for completeness) can be obtained by closely following that in [23].

▶ **Corollary 7** (✂). *Let $\Pi \in \mathcal{P}$. Let $\mathcal{G}$ be a hereditary graph class where the maximum clique can be α-approximated for some constant factor $\alpha \geq 1$ in polynomial time. There exists a $2^{\mathcal{O}\left(\frac{k}{p}\log k\right)} n^{\mathcal{O}(1)}$-time algorithm that, given an instance (G, k) of Π and an integer $p \in [6\alpha c_\Pi, k]$, where $G \in \mathcal{G}$, returns a collection $\mathcal{C}$ of size $2^{\mathcal{O}\left(\frac{k}{p}\log k\right)}$ of tuples (G', M, k') such that:*

[4] In general two d-DIR graphs may require different sets of slopes in their representation but in the case $d = 2$ it is known that the segments can be assumed to be axis-parallel, which we will do.

1. *For any $(G', M, k') \in \mathcal{C}$, (G', k') is an instance of Π where G' is an induced subgraph of G, $\omega(G') \leq p$, and $k' \leq k$;*
2. *M is a c_Π-bundle hitting set of G' with $|M| = \mathcal{O}(pk)$, and for any $v \in M$, $\mu(G'[N(v) \setminus M]) < c_\Pi$; and*
3. *(G, k) is a yes-instance of Π if and only if there exists $(G', M, k') \in \mathcal{C}$ such that (G', k') is a yes-instance of Π.*

3 Positive results via ASQGM

3.1 From $ASQGM(\omega, \mu^{\mathrm{N}^\star})$ to subexponential algorithms

In this section we provide subexponential paramterized algorithms for problems of $\mathcal{P}$ in any class that has the $ASQGM(\omega, \mu^{\mathrm{N}^\star})$ property.

▶ **Definition 8.** *Given a graph G, a* subneighborhood function *of G is any function $\mathrm{N}^\star : V(G) \to 2^{V(G)}$ such that for any $v \in V(G)$, $\mathrm{N}^\star(v) \subseteq N(v)$. Moreover, if for any $u \in V(G)$, $|\{v \in V(G),\ u \in \mathrm{N}^\star(v)\}| \leq c$ for some $c \in \mathbb{N}$ then we say that $\mathrm{N}^\star$ has c-*bounded occurrences.

Given a subneighborhood function $\mathrm{N}^\star$, we define $\mu^{\mathrm{N}^\star}(v)$ as the maximum number of edges of a matching in $G[\mathrm{N}^\star(v)]$. We denote by $\mu^{\mathrm{N}^\star}(G)$ the maximum of $\mu^{\mathrm{N}^\star}$ over $V(G)$.
For example in square graphs, we will fix a representation $\mathcal{S}$, and define $\mathrm{N}^\star(v)$ as the set of neighbors of v whose square is smaller than the one of v.

The main theorem from this subsection is the following. Recall that $\mathcal{P}$ encompasses fundamental algorithmic problems such as FVS, Pseudo Forest Del and P_t-Hitting for $t \leq 5$ (Claim 6).

▶ **Theorem 9.** *Let Π be a problem of $\mathcal{P}$ and $\mathcal{C}$ be a hereditary graph class such that:*
- *maximum clique can be $\mathcal{O}(1)$-approximated in polynomial time in $\mathcal{C}$;*
- *for any $G \in \mathcal{C}$, there exists a subneighborhood function $\mathrm{N}^\star$ that has $\mathcal{O}(\omega(G)^{c_1})$-bounded occurrences for some $c_1 \in \mathbb{N}$; and*
- *$\mathcal{C}$ has the $ASQGM(\omega, \mu^{\mathrm{N}^\star})$ property, i.e., there exists a multivariate polynomial P such that for all $G \in \mathcal{C}$, we have $\mathbf{tw}(G) = \mathcal{O}(P(\omega(G), \mu^{\mathrm{N}^\star}(G)) \cdot \boxplus(G))$.*

Then, Π admits a parameterized subexponential algorithm on $\mathcal{C}$. More precisely, for $\epsilon > 0$ such that $P(k^\epsilon, k^{(c_1+2)\epsilon}) = \mathcal{O}(k^{\frac{1}{2}-\epsilon})$, Π admits a parameterized subexponential algorithm on $\mathcal{C}$ running in time $2^{\mathcal{O}(k^{1-\epsilon}\log(k))}$. This algorithm does not need a representation except if one is required for finding the $\mathcal{O}(1)$-approximation of a maximum clique.

▶ **Lemma 10.** *Let Π be a problem of $\mathcal{P}$. Consider a graph G and $\mathrm{N}^\star$ a c-bounded occurrences subneighborhood function of G. Let $M \subseteq V(G)$ be a c_Π-bundle hitting set of G such that for any vertex $v \in M$, $\mu(G[N(v)] - M) < c_\Pi$. Then for every positive integer $\tau \geq c_\Pi$, there exists a set $B \subseteq V(G)$ of size $|B| = \frac{c|M|}{\tau - c_\Pi + 1}$ such that $\mu^{\mathrm{N}^\star}(G - B) \leq \tau$.*

Proof. Let τ a positive integer with $\tau \geq c_\Pi$, and let us define $B = \{v \in V(G) : \mu^{\mathrm{N}^\star}(v) \geq \tau\}$ the set of vertices with "big" $\mu^{\mathrm{N}^\star}$ in G. Let us first prove that for any $v \in B$, $|\mathrm{N}^\star(v) \cap M| \geq \mu^{\mathrm{N}^\star}(v) - c_\Pi + 1$. Let $E' \subseteq E(G)$ be a maximum matching in $G[\mathrm{N}^\star(v)]$ with $|E'| = \mu^{\mathrm{N}^\star}(v)$. Observe that we cannot have c_Π edges $e \in E'$ such that $V(e) \cap M = \emptyset$ as if $v \notin M$, then vertices of E' together with v would form a c_Π-bundle not hit by M, a contradiction, and if $v \in M$, this would contradict the hypothesis $\mu(G[N(v)] - M) < c_\Pi$. Thus, there is at least $|E'| - c_\Pi + 1$ edges of E' intersecting M, leading to the desired inequality. Thus, we get

$$|B|\tau \leq \sum_{v \in B} \mu^{\mathrm{N}^\star}(v) \leq \sum_{v \in B} (|\mathrm{N}^\star(v) \cap M| + c_\Pi - 1).$$

Moreover, as for any $v \in V(G)$ there are at most c vertices u such that $v \in \mathrm{N}^\star(u)$, we get $\sum_{v \in B} |\mathrm{N}^\star(v) \cap M| \leq c|M|$ by the pigeonhole principle (if the inequality was false, then there would exists $v \in M$ with $|\{u : v \in \mathrm{N}^\star(u)\}| > c$). This leads to $|B| = \frac{c|M|}{\tau - c_\Pi + 1}$. ◀

We are now ready to describe the general algorithm to solve Π.

Proof of Theorem 9. Given an instance (G,k) of Π, we first use Corollary 7 with $p = k^\epsilon$ to obtain in time $2^{\mathcal{O}(k^{1-\epsilon}\log(k))}$ the set of $2^{\mathcal{O}(k^{1-\epsilon}\log(k))}$ triples (G_2, M, k_2) with $k_2 \leq k$, $|M| = \mathcal{O}(k^{1+\epsilon})$, and $\omega(G_2) \leq k^\epsilon$.

In order to solve Π on (G,k), it is now enough to solve it on these instances (G_2, k_2). Observe that applying the Lemma 10 to such (G_2, k_2, M) triple with $\tau \geq c_\Pi$ gives a set B of size at most $\frac{c|M|}{\tau - c_\Pi + 1} = \mathcal{O}(\frac{\omega(G_2)^{c_1} k^{1+\epsilon}}{\tau - c_\Pi + 1}) = \mathcal{O}(\frac{k^{1+\epsilon+\epsilon c_1}}{\tau - c_\Pi + 1})$ such that $G_3 = G_2 \setminus B$ verifies $\mu^{\mathrm{N}^\star}(G_3) \leq \tau$.

By assumption on the $ASQGM$ property we then have $\mathsf{tw}(G_3) = \mathcal{O}(P(k^\epsilon, \tau) \boxplus(G))$. Moreover $\mathsf{tw}(G_2) \leq \mathsf{tw}(G_3) + |B| = \mathcal{O}(P(k^\epsilon, \tau) \boxplus(G)) + \mathcal{O}\left(\frac{k^{1+\epsilon+\epsilon c_1}}{\tau - c_\Pi + 1}\right)$ as removing a vertex decreases the treewidth by at most 1. We set $\tau = k^{(c_1+2)\epsilon}$. By assumption we have $P(k^\epsilon, k^{(c_1+2)\epsilon}) = \mathcal{O}(k^{\frac{1}{2}-\epsilon})$. As Π is bidimensionnal, there exists c_1 such that if $\boxplus(G) > c_1\sqrt{k}$, then (G,k) is a no-instance.

Thus, as $\mathsf{tw}(G_2) = \mathcal{O}\left(k^{\frac{1}{2}-\epsilon} \boxplus(G)\right) + \mathcal{O}\left(\frac{k^{1+\epsilon+\epsilon c_1}}{\tau}\right) = \mathcal{O}\left(k^{\frac{1}{2}-\epsilon} \boxplus(G)\right) + \mathcal{O}(k^{1-\epsilon})$, observe that if $\boxplus(G) \leq c_1\sqrt{k}$, then there exists a constant c such that $\mathsf{tw}(G_2) \leq ck^{1-\epsilon}$. Thus, we use the treewidth approximation of [21] on G_2 with $\ell = ck^{1-\epsilon}$ to obtain in $2^{\mathcal{O}(\ell)} n^{\mathcal{O}(1)}$ either a $2\ell + 1$ treewidth decomposition, or conclude that $\mathsf{tw}(G_2) > \ell$. In the later case, this implies that $\boxplus(G) > c_1\sqrt{k}$, and thus we can conclude that (G,k) is a no instance. Otherwise, by definition of problems in $\mathcal{P}$ we can solve Π in time $\mathsf{tw}^{\mathcal{O}(\mathsf{tw}(G_2))})$, which gives the claimed overall time complexity of $2^{\mathcal{O}(k^{1-\epsilon}\log(k))} \times \mathsf{tw}(G_2)^{\mathcal{O}(\mathsf{tw}(G_2))} = 2^{\mathcal{O}(k^{1-\epsilon}\log(k))}$. ◀

3.2 From $ASQGM(\omega, \mathrm{lr})$ to $ASQGM(\omega, \mu^{\mathrm{N}^\star})$

To be able to use Theorem 9, we need to deal with graph classes that have the $ASQGM(\omega, \mu^{\mathrm{N}^\star})$ property. This section provides a general framework for obtaining this property via *local radius*. The local radius was originally introduced by Lokshtanov et al. [24] for disks graphs in the context of approximation algorithms. Here we first extend this definition to string graphs. To that end, we will see string graphs as graphs admitting a *thick representation*. In such a representation every vertex v of the considered graph G corresponds to a subset $\mathcal{D}_v$ of the plane that is homeomorphic to a disk, two intersecting such regions have an intersection with non-empty interior, and the number of maximal connected regions $\mathbb{R}^2 \setminus \bigcup_{v \in V(G)} \partial\mathcal{D}_v$ is finite.

To turn a string representation into a thick one, it simply suffices to thicken each string by a small enough amount so that no new intersections occur. On the other hand, note that any thick representation can be turned into a string representation by replacing each connected subset of the plane $\mathcal{D}_u$ by a string that almost completely fills its interior. Note that a thick representation is not necessarily a pseudo-disk representation as here, the intersection of two regions, $\mathcal{D}_u \cap \mathcal{D}_v$, may not be connected, or it may also be that $\mathcal{D}_u \setminus \mathcal{D}_v$ is not connected. Thick representations allow us to extend the definition of *local radius* to all string graphs. The next definition is illustrated Figure 4.

▶ **Definition 11.** *Let G be a string graph and $\mathcal{S}$ be a thick representation of it. Let $\mathcal{X}$ be the set of all maximal connected region $\mathcal{R}$ of $\mathbb{R}^2 \setminus \bigcup_{D \in \mathcal{S}} \partial D$, contained in at least one object of $\mathcal{S}$. We define the arrangement graph of $\mathcal{S}$, denoted $A_\mathcal{S}$, by:*

- *adding one vertex of each region of $\mathcal{X}$*
- *adding an edge between two vertices if the boundaries of their regions share a common arc.*

Moreover, for each $v \in G$, we denote $\mathcal{R}_{\mathcal{S}}(v) \subseteq \mathcal{X}$ the set of regions included in $\mathcal{D}_v$ (recall that $\mathcal{D}_v$ is the region associated to v), and $V_{\mathcal{S}}(v) \subseteq V(A_{\mathcal{S}})$ the set of vertices associated to the regions of $\mathcal{R}_{\mathcal{S}}(v)$ (implying $|V_{\mathcal{S}}(v)| = |\mathcal{R}_{\mathcal{S}}(v)|$). Finally, we denote $A_{\mathcal{S}}(v) = A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$.

▶ **Definition 12** (from [24], extended here to string graphs)**.** *Let G be a string graph.*

- *Given a thick representation $\mathcal{S}$ of G,*
  - *for any $v \in V(G)$, we define $\operatorname{lr}_{\mathcal{S}}(v)$ as the radius of the graph $A_{\mathcal{S}}(v)$*
  - *we define $\operatorname{lr}_{\mathcal{S}}(G) = \min_{v \in V(G)} \operatorname{lr}_{\mathcal{S}}(v)$*
- *the local radius $\operatorname{lr}(G)$ of G is the minimum over all thick representation $\mathcal{S}$ of G of $\operatorname{lr}_{\mathcal{S}}(G)$.*

In order to show ASQGM we use the framework of Baste and Thilikos [4] (originally designed for the classic SQGM property), that we recall now.

▶ **Definition 13** (Contractions [4])**.** *Given a non-negative integer c, two graphs H and G, and a surjection $\sigma : V(G) \to V(H)$ we write $H \leq_{\sigma}^{c} G$ if*

- *for every $x \in V(H)$, the graph $G[\sigma^{-1}(x)]$ has diameter at most c and*
- *for every $x, y \in V(H)$, $xy \in E(H) \iff G[\sigma^{-1}(x) \cup \sigma^{-1}(y)]$ is connected.*

We say that H is a c-diameter contraction of G if there is a surjection σ such that $H \leq_{\sigma}^{c} G$ and we write this $H \leq^{c} G$. Moreover, if σ is such that for every $x \in V(H)$, $|\sigma^{-1}(x)| \leq c'$, then we say that H is a c'-size contraction of G, and we write $H \leq^{(c')} G$. If there exists an integer c such that $H \leq^{c} G$, then we say that H is a contraction of G.

▶ **Definition 14** ((c_1, c_2)-extension [4])**.** *Given a class of graph $\mathcal{G}$ and two non-negative integers c_1 and c_2, we define the (c_1, c_2)-extension of $\mathcal{G}$, denoted by $\mathcal{G}^{(c_1,c_2)}$, as the class containing every graph H such that there exist a graph $G \in \mathcal{G}$ and a graph J that satisfy $G \leq^{(c_1)} J$ and $H \leq^{c_2} J$ (see Figure 3).*

$$\begin{tikzcd}
G \in \mathcal{G} & J \arrow[l, "c_1\text{-size contraction}"'] \arrow[r, "c_2\text{-diameter contraction}"] & H \in \mathcal{G}^{(c_1,c_2)}
\end{tikzcd}$$

Figure 3 A graphical representation of the definition of $\mathcal{G}^{(c_1,c_2)}$.

▶ **Lemma 15** (implicit in the proof of [4, Theorem 15])**.** *For every integers c_1, c_2 and $G \in \mathcal{P}^{(c_1,c_2)}$, with $\mathcal{P}$ the class of planar graphs, we have $\mathsf{tw}(G) = \mathcal{O}(c_1 c_2 \boxplus(G))$.*

The main result of this section is the following.

▶ **Theorem 16.** *String graphs have the ASQGM$(\omega, \operatorname{lr})$ property, more precisely for a string graph G we have $\mathsf{tw}(G) = \mathcal{O}(\omega(G) \operatorname{lr}(G) \boxplus(G))$.*

Proof. Let G be a string graph, and $\mathcal{S}$ a thick representation such that $\operatorname{lr}_{\mathcal{S}}(G) = \operatorname{lr}(G)$. Let us define a graph J as follows, Figure 4 is a representation of the construction. For any maximal connected region $\mathcal{R}$ of $\mathbb{R}^2 \setminus \bigcup_{D \in \mathcal{S}} \partial D$, we add to J a clique $K_{\mathcal{R}}$ of size $\operatorname{ply}(\mathcal{R})$. Then, for any pair of regions $\{\mathcal{R}_1, \mathcal{R}_2\}$ that share a common arc, we add all edges between $K_{\mathcal{R}_1}$ and $K_{\mathcal{R}_2}$. For any $v \in V(G)$, we associate a set $X(v) \subseteq V(J)$ such that for any $\mathcal{R} \in \mathcal{R}_{\mathcal{S}}(v)$, $|X(v) \cap K_{\mathcal{R}}| = 1$, and such that $X(v) \cap X(u) = \emptyset$ for any $u \neq v$. Notice that the condition $X(v) \cap X(u) = \emptyset$ is possible as $|K_{\mathcal{R}}| = \operatorname{ply}(\mathcal{R})$, and thus any vertex v can take its "private" vertex in $X(v) \cap \mathcal{R}$ for any $\mathcal{R} \in \mathcal{R}_{\mathcal{S}}(v)$.

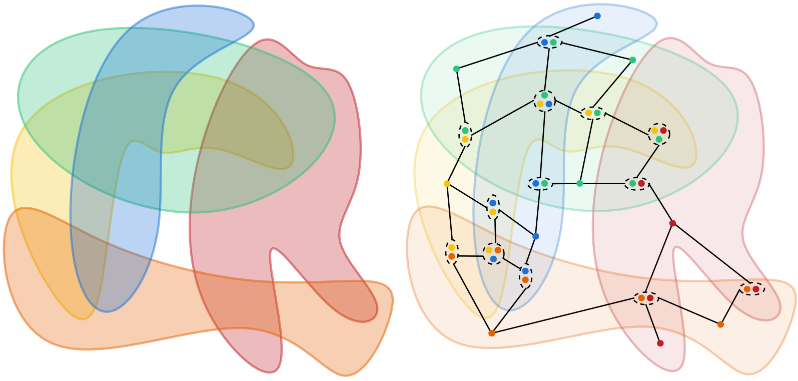


Figure 4 Left: thick representation of a string graph G. Right: Illustrates both $A_{\mathcal{S}}$ and the graph J used in the proof of Theorem 16. To visualise $A_{\mathcal{S}}$, consider that each black dotted ellipse is a single vertex (we have $|V(A_{\mathcal{S}})| = 23$). Moreover, if v is the vertex represented in red, we have $|V_{\mathcal{S}}(v)| = 6$ and $\mathrm{lr}_{\mathcal{S}}(v) = 2$. To visualise J: for each maximal connected region $\mathcal{R}$ of $\mathbb{R}^2 \setminus \bigcup_{D \in \mathcal{S}} \partial D$, the clique $K_{\mathcal{R}}$ with more than one vertex is represented by a black dotted ellipse around the clique. For readability only one edge is represented between two cliques instead of the complete bipartite graph.

Let us prove that G is a $\mathrm{lr}(G)$-diameter contraction of J by defining a surjection $\sigma : V(J) \to V(G)$ as follows. For any $v \in V(G)$, we define $\sigma^{-1}(v) = X(v)$ (informally we contract all vertices in $X(v)$). As for any $v \in V(G)$, $J[X(v)]$ is isomorphic to $A_{\mathcal{S}}(v)$, we immediately have $\mathrm{diam}(J[\sigma^{-1}(v)]) = \mathrm{lr}(G)$. Moreover, it is straightforward to check that for every $x, y \in V(G)$, $xy \in E(G) \iff J[\sigma^{-1}(x) \cup \sigma^{-1}(y)]$ is connected. Now, observe that $A_{\mathcal{S}}$ (which is planar) is a $\mathrm{ply}(\mathcal{S})$-size contraction of J using $\sigma' : V(J) \to V(A_{\mathcal{S}})$ such that for any $v \in V(A_{\mathcal{S}})$, v corresponding to a region $\mathcal{R}$ of the plane delimited by the boundaries of the objects of $\mathcal{S}$, $\sigma'^{-1}(v) = K_R$. As $\mathrm{ply}(\mathcal{S}) \leq \omega(G)$, we get the desired result. ◀

The following corollary is immediate from Theorem 9 and Theorem 16.

▶ **Corollary 17.** *Given an hereditary graph class $\mathcal{C}$ which is a subclass of string graphs such that*

- *maximum clique can be $\mathcal{O}(1)$-approximated in polynomial time,*
- *for any $G \in \mathcal{C}$, there exists a subneighborhood function $\mathrm{N}^\star$ that has $\mathcal{O}(\omega(G)^{c_1})$-bounded occurrences for some $c_1 \in \mathbb{N}$, and*
- *there exists a multivariate polynomial such that for any $G \in \mathcal{C}$, $\mathrm{lr}(G) = P(\omega(G), \mu^{\mathrm{N}^\star}(G))$*

Then, any problem $\Pi \in \mathcal{P}$ admits a parameterized subexponential algorithm on $\mathcal{C}$. More precisely, let $P'(\omega(G), \mu^{\mathrm{N}^\star}(G)) = \omega(G)P(\omega(G), \mu^{\mathrm{N}^\star}(G))$. For any $\epsilon > 0$ such that $P'(k^{\epsilon}, k^{(c_1+2)\epsilon}) = \mathcal{O}(k^{\frac{1}{2}-\epsilon})$, FVS can be solved in time $\mathcal{O}^*(k^{\mathcal{O}(k^{1-\epsilon})})$. This algorithm does not need a representation except if one is required for finding the $\mathcal{O}(1)$-approximation of a maximum clique.

3.3 Upper bounding the local radius for square graphs

Again we provided in the previous section a generic result (Corollary 17) but so far it might not be clear to the reader which graph classes may satisfy its requirements. To demonstrate the applicability of this result, we show here that square graphs do. This requires to define an appropriate $\mathrm{N}^\star$ and prove that $\mathrm{lr}(G) = \omega(G)^{O(1)} \cdot \mu^{\mathrm{N}^\star}(G)^{\mathcal{O}(1)}$. A second application is for contact-segment graphs, but due to space constraints we had to move the proof to the full version [6].

We say that a graph G is a *square graph* if it is the intersection graph of some collection of (closed) axis-parallel squares in the plane. In the following by *square* we always mean closed and axis-parallel square. By slightly altering the sizes and positions of the squares in a collection we can obtain a collection where exactly the same pairs of squares intersect and, in addition, all the side lengths of the squares are different from each other and no two sides squares are aligned. Furthermore this can easily be performed in polynomial time. From now on we will assume that all the representations we consider satisfy this property.

The first requirement of Corollary 17 is provided by following lemma from [8], which describes an EPTAS for the clique problem in the more general case of the intersection graph of a fixed convex geometric shape with a central symmetry, while allowing rescaling.

▶ **Theorem 18** ([8]). *There is a polynomial-time 2-approximation of maximum clique in intersection graphs of squares, even when no representation is provided.*

▶ **Definition 19.** *Given a square representation* $\mathcal{S} = \{\mathcal{D}_v\}_{v \in V(G)}$ *of a graph* G*, we denote* $\ell_{\mathcal{S}}(\mathcal{D}_v)$ *the length of a side of the square* $\mathcal{D}_v$*,* $N^-_{\mathcal{S}}(v)$ *(resp.* $N^+_{\mathcal{S}}(v)$*) the set of vertices* u *such that* $u \in N_G(v)$ *and* $\ell_{\mathcal{S}}(\mathcal{D}_u) < \ell_{\mathcal{S}}(\mathcal{D}_v)$ *(resp.* $>$*). When* $\mathcal{S}$ *is clear from the context, we will instead write* ℓ*,* N^- *and* N^+*.*

As the lengths of all sides differ, $\{N^+(v), N^-(v)\}$ is a partition of $N(v)$ for every vertex v.

▶ **Lemma 20.** *Given a square representation* $\mathcal{S}$ *of a graph* G*,* N^- *is a* $\mathcal{O}(\omega(G))$*-occurrences bounded subneighborhood function.*

Proof. N^- is clearly a subneighborhood function. For $v \in V(G)$, observe that a square larger than $\mathcal{D}_v$ has to contain one of the four corners of $\mathcal{D}_v$ if the two squares intersect. But a corner of $\mathcal{D}_v$ cannot be contained in more than $\omega(G)$ squares. Hence there are at most $4\omega(G)$ vertices $u \in V(G)$ such that $v \in N^-(u)$, and so N^- is $4\omega(G)$-occurrences bounded. ◀

We will prove that choosing $N^* = N^-$ allows us to bound the local radius.

▶ **Definition 21.** *Given a square graph* G *with representation* $\mathcal{S}$*, for any* $v \in G$*, we define* $H(v)$ *as a minimum vertex cover of* $G[N^-(v)]$*,* $I(v) = N^-(v) \setminus H(v)$*, and* $X(v) = H(v) \cup N^+(v)$*.*

▷ Claim 22. For every vertex v of a square graph G with representation $\mathcal{S}$, the following properties hold:

1. $I(v)$ is an independent set of G;
2. $|H(v)| \leq 2\,\mu^{\mathrm{N}^\star}(G)$;
3. $|N^+(v)| = \mathcal{O}(\omega(G))$ (as in the proof of Lemma 20);
4. $|X(v)| = \mathcal{O}(\mu^{\mathrm{N}^\star}(G) + \omega(G))$; and
5. $\{X(v), I(v)\}$ is a partition of $N(v)$.

▶ **Definition 23.** *For a curve* $\mathcal{C} : [0,1] \to \mathbb{R}^2$ *such that for* $t \in [0,1]$*,* $\mathcal{C}(t) = (x(t), y(t))$*, we say that* $\mathcal{C}$ *is* monotonic *if the functions* x *and* y *are monotonic. For* $k \geq 2$ *we say that* $\mathcal{C}$ *is* k*-monotonic if it is the composition*[5] *of* k *monotonic curves.*

Recall in the next Lemma that $\mathcal{D}_{I(v)}$ denotes the union of all squares in $I(v)$.

[5] A curve $\mathcal{C}(t) = (x(t), y(t))$ is the *composition* of k curves $(\mathcal{C}_i(t) = (x_i(t), y_i(t)))_{i \in \{1,\dots,k\}}$ if $(x(0), y(0)) = (x_1(0), y_1(0))$, $(x(1), y(1)) = (x_k(1), y_k(1))$, $(x_i(1), y_i(1)) = (x_{i+1}(0), y_{i+1}(0))$ for every $i \in \{1, \dots, k-1\}$ and the set of points $\{(x(t), y(t)\}, t \in [0,1]\}$ is the union of the $\{(x_i(t), y_i(t)\}, t \in [0,1]\}$ for $i \in \{1, \dots, k\}$.

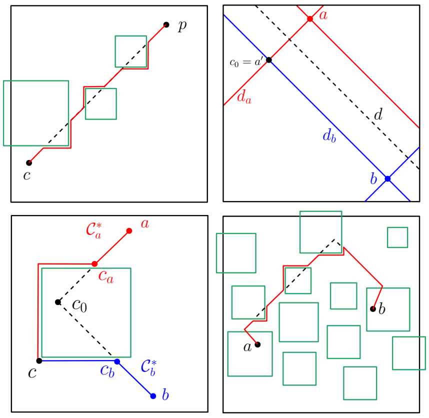


Figure 5 Illustrations of the construction used in the proof of the Lemma 24. Squares of $I(v)$ are represented in green. Top left: construction used for the Claim 25. Top right: construction used for the Claim 26. Bottom left: construction used for Claim 27. Observe that in this situation c_a and c_b are next to opposite sides of the square containing c_0, that $\mathcal{C}_a^*$ can be extended in an counterclockwise direction, and $\mathcal{C}_b^*$ in a clockwise direction, which ensure the existence of a common point c of their monotonic extensions. Bottom right: an example of a 4-monotonic curve between a and b obtained by the construction of Lemma 24. Observe that only two squares of $I(v)$ are crossed.

▶ **Lemma 24.** *Let G be a square graph and $\mathcal{S}$ a representation. Let $v \in V(G)$ and a, b two points contained in $\mathcal{D}_v$. There exists a 4-monotonic curve $\mathcal{C}$ contained in $\mathcal{D}_v$ joining the point a to the point b, and crossing at most twice a boundary of the squares of $I(v)$.*

Proof. In what follows, what we call a *diagonal line (resp. half line)* any line (resp. half line) having an angle +45° or −45° with the horizontal axis, and a *diagonal of a point p in the plan* a diagonal half line whose endpoint is p.

The first step for the creation of the curve is to reduce to the case where the point a and b are outside $\mathcal{D}_{I(v)}$. If this is not the case, for example if a in contained in a square $s = \mathcal{D}_u$ with $u \in I(v)$, we create a rectilinear curve from a toward the outside of s, in a direction such that the intersection of the curve with the boundary of s is still in $\mathcal{D}_v$ (see the construction in Figure 5 for an example of such reduction). As such curve is monotonic and crosses the boundary of a square of $I(v)$ exactly once, after the reduction we are in the case where we want to construct a 2-monotonic curve between two points of $\mathcal{D}_v \setminus \mathcal{D}_{I(v)}$ such that no square of $I(v)$ is crossed. In what follow we suppose we have reduced to this case and we still denote a and b the two points of $\mathcal{D}_v \setminus \mathcal{D}_{I(v)}$ we want to join by a curve.

▷ Claim 25. Given two points $c, p \in \mathcal{D}_v \setminus \mathcal{D}_{I(v)}$ on the same diagonal line, there is a monotonic curve included in $\mathcal{D}_v \setminus \mathcal{D}_{I(v)}$ between c and p.

Proof. The construction is represented in Figure 5. The curve is created by starting from the point c, then by following the diagonal line toward p. When encountering a square $s = \mathcal{D}_u$ of a vertex $u \in I(v)$, it is always possible of getting around s in order to join back the diagonal on the other side, and doing so in a direction such that the curve is still monotonic and contained in $\mathcal{D}_v$. ◁

▷ Claim 26. There are diagonals d_a of a and d_b of b intersecting on a point $c_0 \in \mathcal{D}_v$.

Proof. Consider the line d parallel to the top left to bottom right diagonal of $\mathcal{D}_v$ (see Figure 5), at equal distances of the points a and b. By symmetry of the square and of the variables a and b, we can suppose that d goes from top left to bottom right, is above the diagonal of $\mathcal{D}_v$, and that a is above d. The symmetric a' of the point a relatively to d is inside $\mathcal{D}_v$ and is contained in a diagonal of both a and b. ◁

Now, if $c_0 \in \mathcal{D}_v \setminus \mathcal{D}_{I(v)}$, composing the two curves toward c_0 given by the previous claim gives the wanted result.

It remains to deal with the case where c_0 lies in some square $s = \mathcal{D}_u$ for $u \in I(v)$. Let c_a be a point of d_a between a and the square s, at an infinitely small distance outside of s. Claim 25 gives a monotonic curve $\mathcal{C}_a^*$ from a to c_a. In the same way we define c_b and $\mathcal{C}_b^*$.

▷ Claim 27. There exists a point $c \in \mathcal{D}_v \setminus \mathcal{D}_{I(v)}$ such that $\mathcal{C}_a^*$ and $\mathcal{C}_b^*$ can be extended to c while still being monotonic and contained in $\mathcal{D}_v \setminus \mathcal{D}_{I(v)}$.

Proof. We can assume that d_a and d_b are perpendicular as otherwise the points a and b are on the same diagonal and so Claim 25 gives the wanted result by taking $c = b$. Observe that if c_a and c_b are arbitrarily close to the same side of s, then prolonging $\mathcal{C}_a^*$ toward c_b would keep the curve monotonic, as $\mathcal{C}_a^*$ was already going toward d_b as d_a and d_b intersect in s. So taking $c = c_b$ would give the wanted result.

Otherwise if c_a and c_b are at arbitrarily small distance from two different sides, observe that the curve $\mathcal{C}_a^*$ can be extended running alongside the boundary of s until crossing 2 corners. The same is true for $\mathcal{C}_b^*$ so the only situation where those extensions do not cross each other would be if c_a and c_b are next to opposite side of s, and that the orientations of d_a and d_b force the extensions of $\mathcal{C}_a*$ and $\mathcal{C}_b^*$ to go in the same direction around s. However, this is impossible: as d_a and d_b cross each other inside of s, one extension will go clockwise around s and the other counterclockwise (see Figure 5). This ensures that $\mathcal{C}_a^*$ and $\mathcal{C}_b^*$ can be extended around s while still being monotonic in order for them to join on a point c while staying outside of $\mathcal{D}_{I(v)}$. ◁

Composing the two curves obtained by the above claim gives a path as wanted. ◀

We are now ready to prove the main combinatorial statement of this section.

▶ **Lemma 28.** *Let G be a square graph. There exists a subneighborhood function* $\mathrm{N}^\star$ *which is $\omega(G)$-occurrences bounded and such that* $\mathrm{lr}(G) = \mathcal{O}(\mu^{\mathrm{N}^\star}(G) + \omega(G))$.

Proof. Let $\mathcal{S}$ be a square representation of G, and let $\mathrm{N}^\star$ as defined in Definition 19, which is $\omega(G)$-occurrences bounded according to Lemma 20. Let us now prove that $\mathrm{lr}_{\mathcal{S}}(G) = \mathcal{O}(|X(v)|)$. This will imply the required result as $\mathrm{lr}(G) \leq \mathrm{lr}_{\mathcal{S}}(G)$ and $|X(v)| = \mathcal{O}(\mu^{\mathrm{N}^\star}(G) + \omega(G))$ by Claim 22. To that end, let us bound the diameter of $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$. Let u, v be two vertices of $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$, and let us bound the distance between these two vertices. Remember that any vertex in $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$ corresponds to an inclusion-wise maximal rectangular region of the plane included in $\mathcal{D}_v$, and delimited by edges of squares of $\mathcal{S}$. Let a and b be points in the regions of u and v respectively. Notice that to any curve inside $\mathcal{D}_v$ we can associate a path in $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$ by considering the sequence of regions visited by $\mathcal{C}$, and associate to each of the region its corresponding vertex in $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$ (see Figure 6). Thus, we will upper bound the distance from u to v in $A_{\mathcal{S}}[V_{\mathcal{S}}(v)]$ by constructing a curve $\mathcal{C}$ from a to b, and by counting the length of the sequence of regions visited by $\mathcal{C}$.

We use for $\mathcal{C}$ the 4-monotonic curve between a and b defined in Lemma 24. Observe the following property π_0: any monotonic curve inside $\mathcal{D}_v$ crosses at most $4|X(v)|$ sides of squares in $X(v)$. Indeed, as each square in $X(v)$ has at most 4 sides intersecting $\mathcal{D}_v$, and any

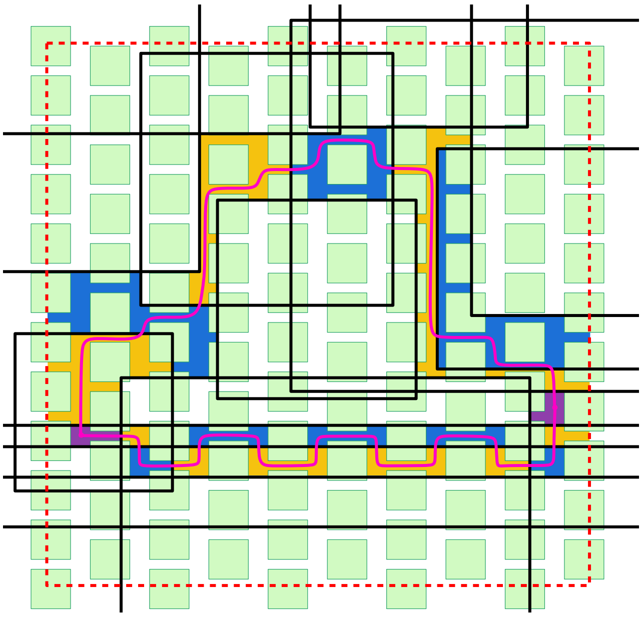


Figure 6 Examples of paths in the configuration graph, with $\mathcal{D}_v$ represented with a dashed red square, $I(v)$ by green squares and the sides of the squares of $X(v)$ in black. Here we can see two curves between the two purple regions, $\mathcal{C}_1$ (that goes up and then down) and $\mathcal{C}_2$, and the path in $A_\mathcal{S}(v)$ associated to each curve as in the proof of Lemma 28, where the regions traversed by the paths are alternatively colored blue and yellow. Notice that $\mathcal{C}_1$ is 2-monotone, whereas $\mathcal{C}_2$ is c-monotone, where c could be made arbitrary large by creating more and smaller squares in $I(v)$. As c is large, there is a side of a square in $X(v)$ crossed many times (eight) by $\mathcal{C}_2$, and thus we do not use curve like $\mathcal{C}_2$ in the proof.

side, as a vertical or horizontal segment intersecting in $\mathcal{D}_v$, can be crossed at most one time by a monotonic curve. Observe also that, each time $\mathcal{C}$ leaves its current region, $\mathcal{C}$ must cross a side of a square in $N(v)$. However, the total number of crossings between $\mathcal{C}$ and a side of a square in $N(v)$ is at most $16|X(v)| + 4$, as each of the four monotonic part of $\mathcal{C}$ crosses at most $4|X(v)|$ sides of squares in $X(v)$ (by π_0), and $\mathcal{C}$ crosses at most 4 sides of squares in $I(v)$ (the worst case being when $a \neq a'$, and $\mathcal{C}_{a\to a'}$ crosses the corner of the square in $I(v)$ containing a, and same for b, b'). Thus, the curve $\mathcal{C}$ goes from a region to the next one at most $16|X(v)| + 4$ times, implying that the diameter of $A_\mathcal{S}[V_\mathcal{S}(v)]$, and so the local radius $\mathrm{lr}_\mathcal{S}(G)$, are in $\mathcal{O}(|X(v)|)$. ◀

As announced in the introduction of the section, we are now able to apply Corollary 17.

▶ **Theorem 29.** *Any problem $\Pi \in \mathcal{P}$ can be solved in time $2^{\mathcal{O}(k^{9/10}\log(k))}n^{\mathcal{O}(1)}$ in square graphs, even when no representation is given.*

Proof. Let $\Pi \in \mathcal{P}$. According to Theorem 18, Lemma 28, we can apply Corollary 17 with $c_1 = 1$, and $P(x, y) = x + y$. This implies that for any ϵ such that $k^\epsilon(k^\epsilon + k^{3\epsilon}) = \mathcal{O}(k^{\frac{1}{2}-\epsilon})$, Π can be solved in $\mathcal{O}^*(k^{\mathcal{O}(k^{1-\epsilon})})$ in square graphs. Taking $\epsilon = \frac{1}{10}$ leads to the claimed complexity. ◀

4 ETH based hardness results

Let us here sketch the lower bounds. Full proofs are provided in the full version.

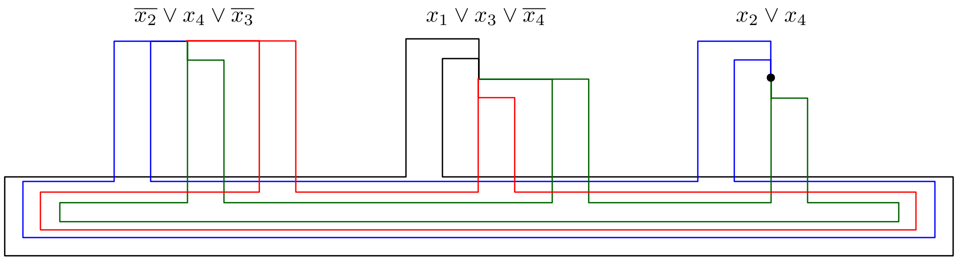

$\overline{x_2} \vee x_4 \vee \overline{x_3}$
$x_1 \vee x_3 \vee \overline{x_4}$
$x_2 \vee x_4$

Figure 7 The construction for the formula $(\overline{x_2} \vee x_4 \vee \overline{x_3}) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_2 \vee x_4)$. The zero-length segments at each corner of the k-polygons are not represented, while that added for the clause with two variables is depicted with a black dot.

▶ **Theorem 30.** *Under the ETH, TH and OCT cannot be solved in time $2^{o(n)}$ on n-vertex 2-DIR graphs.*

Sketch of Proof. Let φ be a 3-SAT instance with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. In these clauses, we do not have 3 literals all positive or all negative. We can ensure this by adding only few variables and few clauses.

Let us now construct a 2-DIR graph G from the formula φ. In this graph, each variable x_i is represented by a polygon with k_i vertical segments, k_i horizontal segments, and with also $2k_i$ trivial segments (i.e. points) that are placed in each corner of the polygon, where k_i is some number linear in the number of clauses containing x_i. See Figure 7 for an illustrative example. There, one can see that these polygons form concentric rectangles, from which small parts escape from above. These escaping parts allow interactions with other polygons, corresponding to variables from a same clause.

The idea of the reduction is that, φ is satisfiable if and only if G has a TH (resp. OCT) of size $K = \sum_{1 \le i \le n} k_i$. Furthermore, such hitting set will be of the following form. For the polygon corresponding to x_i, the hitting set will be either formed by the k_i vertical segments, or by the k_i horizontal segments. This is ensured by the triangles induced at each corner of the polygon. Furthermore, the choice of vertical or horizontal segments, depends on the interactions among polygons, and will correspond to a valuation of the variable x_i. ◀

In the full version [6] we also provide a refined bound of Theorem 30 depending on the maximum degree, and another negative result in $K_{2,2}$-free contact 2-DIR graphs.

5 Discussion

In this paper we gave subexponential FPT algorithms for cycle-hitting problems in intersection graphs. A general goal is to characterize the geometric graph classes that admit subexponential FPT algorithms for the problems we considered. In particular, an interesting open problem is whether FVS admits a subexponential parameterized algorithm in 2-DIR graphs.

References

1 Shinwoo An, Kyungjin Cho, and Eunjin Oh. Faster algorithms for cycle hitting problems on disk graphs. In *Algorithms and Data Structures: 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 – August 2, 2023, Proceedings*, pages 29–42, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-38906-1_3.

2 Shinwoo An and Eunjin Oh. Feedback vertex set on geometric intersection graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

3 Julien Baste, Ignasi Sau, and Dimitrios M Thilikos. Hitting minors on bounded treewidth graphs. iv. an optimal algorithm. *arXiv preprint*, 2019. `arXiv:1907.04442`.

4 Julien Baste and Dimitrios M Thilikos. Contraction bidimensionality of geometric intersection graphs. *Algorithmica*, 84(2):510–531, 2022.

5 Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond. Feedback vertex set for pseudo-disk graphs in subexponential fpt-time, 2024. To appear on Arxiv.

6 Gaétan Berthe, Marin Bougeret, Daniel Gonçalves, and Jean-Florent Raymond. Subexponential parameterized algorithms in square graphs and intersection graphs of thin objects, 2024. `arXiv:2306.17710`.

7 Marthe Bonamy, Edouard Bonnet, Nicolas Bousquet, Pierre Charbit, and Stéphan Thomassé. Eptas for max clique on disks and unit balls. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 568–579. IEEE, 2018.

8 Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.FSTTCS.2020.17`.

9 Édouard Bonnet and Paweł Rzążewski. Optimality program in segment and string graphs. *Algorithmica*, 81:3047–3073, 2019.

10 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.

11 Julia Chuzhoy and Zihan Tan. Towards tight (er) bounds for the excluded grid theorem. *Journal of Combinatorial Theory, Series B*, 146:219–265, 2021.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

13 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Information and Computation*, 256:62–82, 2017. `doi:10.1016/j.ic.2017.04.009`.

14 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.

15 Erik D Demaine and MohammadTaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

16 Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173(33):12, 2005.

17 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete & Computational Geometry*, 62:879–911, 2019.

18 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1563–1575, USA, 2012. Society for Industrial and Applied Mathematics.

19 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Excluded grid minors and efficient polynomial-time approximation schemes. *Journal of the ACM (JACM)*, 65(2):1–44, 2018.

20 Alexander Grigoriev, Athanassios Koutsonas, and Dimitrios M Thilikos. Bidimensionality of geometric intersection graphs. In *SOFSEM 2014: Theory and Practice of Computer Science: 40th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 26-29, 2014, Proceedings 40*, pages 293–305. Springer, 2014.

21 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192. IEEE, 2022.

22 Tuukka Korhonen and Daniel Lokshtanov. Induced-minor-free graphs: Separator theorem, subexponential algorithms, and improved hardness of recognition. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2024.

23 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. Subexponential parameterized algorithms on disk graphs (extended abstract). In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2031. SIAM, 2022.

24 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. A framework for approximation schemes on disk graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2228–2241. SIAM, 2023. doi:10.1137/1.9781611977554.CH84.

25 Jana Novotná, Karolina Okrasa, Michał Pilipczuk, Paweł Rzążewski, Erik Jan van Leeuwen, and Bartosz Walczak. Subexponential-time algorithms for finding large induced sparse subgraphs. *Algorithmica*, 83:2634–2650, 2021.

26 Karolina Okrasa and Paweł Rzążewski. Subexponential algorithms for variants of the homomorphism problem in string graphs. *Journal of Computer and System Sciences*, 109:126–144, 2020.

Arboricity-Dependent Algorithms for Edge Coloring

Sayan Bhattacharya ✉
University of Warwick, UK

Martín Costa ✉
University of Warwick, UK

Nadav Panski ✉
Tel Aviv University, Israel

Shay Solomon ✉
Tel Aviv University, Israel

Abstract

The problem of edge coloring has been extensively studied over the years. Recently, this problem has received significant attention in the *dynamic setting*, where we are given a dynamic graph evolving via a sequence of edge insertions and deletions and our objective is to maintain an edge coloring of the graph.

Currently, it is not known whether it is possible to maintain a $(\Delta + O(\Delta^{1-\mu}))$-edge coloring in $\tilde{O}(1)$ update time, for any constant $\mu > 0$, where Δ is the maximum degree of the graph.[1] In this paper, we show how to efficiently maintain a $(\Delta + O(\alpha))$-edge coloring in $\tilde{O}(1)$ amortized update time, where α is the arboricty of the graph. Thus, we answer this question in the affirmative for graphs of sufficiently small arboricity.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic Algorithms, Graph Algorithms, Edge Coloring, Arboricity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.12

Related Version *Full Version*: `https://arxiv.org/abs/2311.08367`

Funding Shay Solomon is funded by the European Union (ERC, DynOpt, 101043159). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Shay Solomon and Nadav Panski are supported by the Israel Science Foundation (ISF) grant No.1991/1. Shay Solomon is also supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF).

1 Introduction

Consider any graph $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges, and any integer $\lambda \geq 1$. A *(proper) λ-(edge) coloring* $\chi : E \to [\lambda]$ of G assigns a color $\chi(e) \in [\lambda]$ to each edge $e \in E$, in such a way that no two adjacent edges receive the same color. Our goal is to get a proper λ-coloring of G, for as small a value of λ as possible. It is easy to verify that any such coloring requires at least Δ colors, where Δ is the maximum degree of G. On the other hand, a textbook theorem by Vizing [13] guarantees the existence of a proper $(\Delta + 1)$-coloring in any input graph.

This work focuses on the edge coloring problem in the *dynamic setting*, where an extensive body of work has been devoted to this problem. Before describing our contributions, we first summarize the relevant state-of-the-art in the dynamic setting.

[1] We use $\tilde{O}(\cdot)$ to hide polylogarthmic factors.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dynamic Edge Coloring. In the dynamic setting, the input graph G undergoes a sequence of updates (edge insertions/deletions), and throughout this sequence the concerned algorithm has to maintain a proper coloring of G. We wish to design a dynamic algorithm whose *update time* (time taken to process an update) is as small as possible. The edge coloring problem has received significant attention within the dynamic algorithms community in recent years. It is known how to maintain a $(2\Delta - 1)$-coloring in $O(\log \Delta)$ update time [2, 3], and Duan et al. [11] showed how to maintain a $(1+\epsilon)\Delta$-coloring in $O(\log^8 n/\epsilon^4)$ update time when $\Delta = \Omega(\log^2 n/\epsilon^2)$. Subsequently, Christiansen [10] presented a dynamic algorithm for $(1+\epsilon)\Delta$-coloring with $O(\log^9 n \log^6 \Delta/\epsilon^6)$ update time, without any restriction on Δ. More recently, Bhattachrya et al. [5] showed how to maintain a $(1+\epsilon)\Delta$-coloring in $O(\log^4(1/\epsilon)/\epsilon^9)$ update time when $\Delta \geq (\log n/\epsilon)^{\Theta((1/\epsilon)\log(1/\epsilon))}$. At present, no dynamic edge coloring algorithm is known with a sublinear in Δ *additive approximation* and with $\tilde{O}(1)$ update time. We summarize the following basic question that arises.

> Is there a dynamic algorithm for maintaining a $(\Delta + O(\Delta^{1-\mu}))$-edge coloring with $\tilde{O}(1)$ update time, for any constant $\mu > 0$?

1.1 Our Contribution

We address the above question for the family of *bounded arboricity* graphs. Formally, a graph $G = (V, E)$ has *arboricity* (at most) α iff:

$$\left\lceil \frac{|E(G[S])|}{(|S|-1)} \right\rceil \leq \alpha \text{ for every subset } S \subseteq V \text{ of size } |S| \geq 2,$$

where $G[S]$ denotes the subgraph of G induced by S and $E(G[S])$ denotes the edge-set of $G[S]$. It is easily verified that the arboricity of any graph is upper bounded by its maximum degree. There are many instances of graphs, however, with very high maximum degree but low arboricity.[2] Intuitively, a graph with low arboricity is *sparse everywhere*. Every graph excluding a fixed minor has $O(1)$ arboricity, thus the family of constant arboricity graphs contains bounded treewidth and bounded genus graphs, and specifically, planar graphs. More generally, graphs of bounded (not necessarily constant) arboricity are of importance, as they arise in real-world networks and models, such as the world wide web graph, social networks and various random distribution models.

We now summarize our main result.

▶ **Theorem 1.** *There is a deterministic dynamic algorithm for maintaining a $(\Delta + (4+\epsilon)\alpha)$-edge coloring of an input dynamic graph with maximum degree Δ and arboricity α, with $O(\log^6 n/\epsilon^6)$ amortized update time and $O(\log^4 n/\epsilon^5)$ amortized recourse.*[3]

Thus, Theorem 1 addresses the above question in the affirmative, for all dynamic graphs with arboricity at most $O(\Delta^{1-\mu})$, for any constant $\mu > 0$.

An important feature of our dynamic algorithm is that it is *adaptive* to changes in the values of Δ and α over time: At each time-step t, we (explicitly) maintain a proper edge coloring of the input graph G using the colors $\{1, \dots, \Delta_t + (4+\epsilon)\alpha_t\}$, where Δ_t and α_t are respectively the maximum degree and arboricity of G at time t.

[2] Think of a star graph on n nodes. It has $\Delta = n - 1$ but $\alpha = 1$.

[3] A dynamic algorithm has an *amortized update time* (respectively, *amortized recourse*) of $O(\lambda)$, if, starting with an empty graph, the total runtime (resp., number of output changes) to handle any sequence of T updates is $O(T \cdot \lambda)$.

Before giving our full dynamic algorithm, we give a simpler "warmup" dynamic algorithm, where we assume access to values α and Δ such that $\alpha_t \leq \alpha$ and $\Delta_t \leq \Delta$ at each time-step t. In this setting, we can maintain a $(\Delta + (4+\epsilon)\alpha)$-edge coloring with $O(\log^2 n \log \Delta/\epsilon^2)$ amortized update time and $O(\log n/\epsilon)$ worst-case recourse. As an immediate corollary of our "warmup" dynamic algorithm, we also get the following structural result, which should be contrasted with the lower bound of [7] for extending partial colorings, which shows that there exist n-node graphs of maximum degree Δ and $(\Delta + c)$-edge colorings on those graphs (for any $c \in [1, \Delta/3]$), such that extending these colorings to color some uncolored edge requires changing the colors of $\Omega(\Delta \log(cn/\Delta)/c)$ many edges.

▶ **Corollary 2.** *Let $G = (V, E)$ be a graph with maximum degree Δ and arboricity α, and let χ be a $(\Delta + (2+\epsilon)\alpha)$-edge coloring of G. Then, given any uncolored edge $e \in E$, we can extend the coloring χ so that e is now colored by only changing the colors of $O(\log n/\epsilon)$ many edges.*

Independent Work. In independent and concurrent work, Christiansen, Rotenberg and Vlieghe also obtain a deterministic dynamic algorithm that maintains a $(\Delta + O(\alpha))$-edge coloring in $\tilde{O}(1)$ amortized update time [9].

1.2 Our Techniques

At a high level, our algorithm can be interpreted as a dynamization of a simple static algorithm that computes a $(\Delta + O(\alpha))$-edge coloring of a graph G, which can be implemented to run in near-linear time in the static sequential model of computation.[4] This algorithm is similar to the classic greedy algorithm for $(2\Delta - 1)$-edge coloring, which simply scans through all edges of the graph in an arbitrary order and, while scanning any edge e, assigns e an arbitrary color in $[2\Delta - 1]$ that has not been already assigned to one of its adjacent edges. Since e has at most $2\Delta - 2$ adjacent edges, such a color must always exist. This static algorithm does something quite similar – the difference is that it computes a "good" ordering of the edges in G instead of using an arbitrary ordering, which allows it to use fewer colors. More specifically, it repeatedly identifies a vertex of minimum degree in G, colors an edge incident on in, and removes that edge from the graph. For the sake of completeness, we include this algorithm and its analysis in Appendix A of the full version of our paper. We remark that a variant of this algorithm appears in [1], which considers the distributed model of computation.

To highlight the main conceptual insight underlying our approach, we describe the simpler case where Δ and α are *fixed* values (known to the algorithm in advance) that respectively give upper bounds on the maximum degree and arboricity of the input graph at all times. We sketch below how to maintain a $(\Delta + O(\alpha))$-coloring in $\tilde{O}(1)$ update time in this setting. Note that this directly implies a near-linear time static algorithm for $(\Delta + O(\alpha))$-coloring.[5] We later outline (Section 1.2.1) how we extend our dynamic algorithm to handle the scenario where Δ and α change over time.

Our starting point is a well-known "peeling process", which leads to a standard decomposition of an input graph $G = (V, E)$ with arboricity at most α [8]. The key observation is that *any* induced subgraph of G has average degree at most 2α.[6] Fix any constant $\gamma > 1$.

[4] Recently, [4] and [12] considered edge coloring on low arboricity graphs in the static setting, but for the problems of $\Delta + 1$ and Δ coloring respectively.

[5] Indeed, we can compute Δ and a good approximation of α in linear time, and then simply insert the edges in the input graph into the dynamic algorithm one after another.

[6] Indeed, for any subset $S \subseteq V$, the average degree of $G[S]$ is given by: $2 \cdot |E(G[S])|/|S| \leq 2\alpha$.

This motivates the following procedure, which runs for $L = \Theta_\gamma(\log n)$ rounds.

Initially, during round 1, we set $Z_1 := V$. Subsequently, during each round $i \in \{2, \ldots, L\}$, we find the set of nodes $S \subseteq Z_{i-1}$ that have degree $> 2\gamma\alpha$ in $G[Z_{i-1}]$, and set $Z_i := S$.

Consider any given round $i \in [L]$ during the above procedure. Since the subgraph $G[Z_{i-1}]$ has average degree at most 2α, it follows that at most a $1/\gamma$ fraction of the nodes in $G[Z_{i-1}]$ have degree more than $2\gamma\alpha$. In other words, we get $|Z_{i+1}| \leq |Z_i|/\gamma$, and hence after L iterations we would have $Z_L = \varnothing$. Bhattacharya et al. [6] showed how to maintain this decomposition dynamically with $\tilde{O}(1)$ amortized update time, provided that $\gamma > 2$.

Now, our dynamic $(\Delta + O(\alpha))$-coloring algorithm works as follows. Suppose that we are currently maintaining a valid coloring, along with the above decomposition. Upon receiving an update (edge insertion/deletion), we first run the dynamic algorithm of [6], which adjusts the decomposition $Z_1 \supseteq \cdots \supseteq Z_L$, in amortized $\tilde{O}(1)$ time. If the update consisted of an edge deletion, then we do not need to do anything else beyond this point, since the existing coloring continues to remain valid. We next consider the more interesting case, where the update consisted of the insertion of an edge (say) (u, v).

Let $i \in [L]$ be the largest index such that $(u, v) \in E(G[Z_i])$. Then there must exist some endpoint $x \in \{u, v\}$ that belongs to $Z_i \setminus Z_{i+1}$. W.l.o.g., let u be that endpoint. Since $u \in Z_i \setminus Z_{i+1}$, it follows that the node u has degree at most $2\gamma\alpha$ in $G[Z_i]$. Also, the node v trivially has degree at most Δ in G. Let $E_{(u,v)} \subseteq E$ denote the set of edges $e' \in E$ that belong to one of the following two categories: (I) e' is incident on u and lies in $G[Z_i]$, (II) e' is incident on v. We conclude that $|E_{(u,v)}| \leq \Delta + 2\gamma\alpha$. Thus, if we have a palette of at least $\Delta + 2\gamma\alpha + 1 = \Delta + \Theta(\alpha)$ colors, then there must exist a free color in that palette which is not assigned to any edge in $E_{(u,v)}$. Let c be that free color. Using standard binary search data structures, such a color c can be identified in $\tilde{O}(1)$ time [3]. We assign the color c to the edge (u, v). This can potentially create a conflict with some other adjacent edge $e'' \in E$ (which might already have been assigned the color c).

However, it is easy to see that such an edge e'' must be incident on u, i.e., $e'' = (u, y)$ for some $y \in V$, and there must exist some index $i_y < i$ such that $y \in Z_{i_y} \setminus Z_{i_y+1}$. We then uncolor the edge e'', set $i \leftarrow i_y$, and recolor e'' recursively using the same procedure described above. Since after each recursive call, the value of the index i decreases by at least one, this can go on at most L times. This leads to an overall update time of $L \cdot \tilde{O}(1) = \tilde{O}(1)$. See Section 3 for details.

1.2.1 Handling the scenario where Δ and α change over time

We now outline how we deal with changing values of Δ and α. Let α_t and Δ_t respectively denote the arboricity and maximum degree of the input graph G at the current time-step t. We need to overcome two technical challenges.

(i) The "warmup" algorithm described above works correctly only if it uses a parameter $\alpha \simeq \alpha_t$ to construct the decomposition of G. Informally, if α is too small w.r.t. α_t, then the number of iterations L required to construct the decomposition will become huge (possibly infinite, if we aim at achieving $Z_L = \varnothing$), and this in turn would blow up the update time of the algorithm. In contrast, if α is too large compared to α_t, then the algorithm would be using too many colors in its palette.

(ii) After the deletion of an edge e, the arboricity α and the maximum degree Δ of G might decrease. If either parameter drops by a significant amount (across some batch of updates), then we might have to recolor a significant number of edges to ensure that we are still only using $\Delta + O(\alpha)$ many colors, potentially leading to a prohibitively large update time.

To deal with challenge (i), we generalize the notion of graph decomposition to that of a *decomposition system*. At a high level, a decomposition system is just a collection of graph decompositions, where the relevant parameter across the decompositions is discretized into powers of $(1 + \epsilon)$. This ensures that no matter what the value of α is at the present moment, there is always some decomposition in our system that we can use to extend the coloring. Finally, to deal with challenge (ii), we ensure that the color of each edge satisfies certain *local constraints*, similar to the constraints used to give efficient dynamic algorithms in [3, 10]. After the deletion of an edge, we can just uncolor the edges that violate those local constraints, and then recolor them using the decomposition system. However, since the constraints on an edge e depend not just on the degrees of its endpoints but also on the decomposition system, we have to take extra care to ensure that these decompositions don't change too much between updates. See Section 4 for details.

1.3 Roadmap

The rest of the paper is organized as follows. Section 2 introduces the relevant preliminary concepts and notations. This is followed by Section 3, which contains our warmup dynamic algorithm for fixed α. In Section 4, we present our dynamic algorithm in its full generality. Appendix B in the full version of our paper gives the full details of the relevant data structures used by our algorithms.

2 Preliminaries

In this section, we define the notations used throughout our paper and describe the notion of *graph decompositions*, which are at the core of our algorithms. We then provide a simple extension of these graph decompositions, which we use as a central component in our final dynamic algorithm.

2.1 The Dynamic Setting

In the dynamic setting, we have a graph $G = (V, E)$ that undergoes updates via a sequence of intermixed edge insertions and deletions. Our task is to design an algorithm to explicitly maintain an edge coloring χ of G as the graph is updated. We assume that the graph G is initially empty, i.e. that the graph G is initialized with $E = \varnothing$. The *update time* of such an algorithm is the time it takes to handle an update, and its *recourse* is the number of edges that change colors while handling an update. More precisely, we say that an algorithm has a worst-case update time of λ if it takes at most λ time to handle an update, and an *amortized* update time of λ if it takes at most $T \cdot \lambda$ time to handle any arbitrary sequence of T updates (starting from the empty graph). Similarly, we say that an algorithm has a worst-case recourse of λ if it changes the colors of at most λ edges while handling an update, and an *amortized* recourse of λ if it changes the colors of at most $T \cdot \lambda$ edges while handling any arbitrary sequence of T updates (starting from the empty graph).

2.2 Notation

Let $G = (V, E)$ be an undirected, unweighted n-node graph. Given an edge set $S \subseteq E$, we denote by $G[S]$ the graph (V, S), and given a node set $A \subseteq V$, we denote by $G[A]$ the subgraph induced by A, namely $(A, \{(u, v) \in E \,|\, u, v \in A\})$. Given a node $u \in V$ and a subgraph H of G, we denote by $N_H(u)$ the set of edges in H that are incident on u, and by $\deg_H(u)$ the degree of u in H. For an edge (u, v), we define $N_H(u, v)$ to be $N_H(u) \cup N_H(v)$. When we are considering the entire graph G, we will often omit the subscripts in $N_G(\cdot)$ and $\deg_G(\cdot)$ and just write $N(\cdot)$ and $\deg(\cdot)$.

2.3 Graph Decompositions

A central ingredient in our dynamic algorithm is the notion of (β, d, L)-*decomposition*, defined by Bhattacharya et al. [6].

▶ **Definition 3.** *Given a graph $G = (V, E)$, $\beta \geq 1$, $d \geq 0$, and a positive integer L, a (β, d, L)-decomposition of G is a sequence $(Z_1, \dots, Z_L)$ of node sets, such that $Z_L \subseteq \cdots \subseteq Z_1 = V$ and*

$$Z_{i+1} \supseteq \{u \in Z_i \mid \deg_{G[Z_i]}(u) > \beta d\} \text{ and } Z_{i+1} \cap \{u \in Z_i \mid \deg_{G[Z_i]}(u) < d\} = \varnothing$$

hold for all $i \in [L - 1]$.

Given a (β, d, L)-decomposition $(Z_1, \dots, Z_L)$ of $G = (V, E)$, we abbreviate $G[Z_i]$ as G_i for all i, and for all $u \in V$, we abbreviate $\deg_{G_i}(u)$ as $\deg_i(u)$ and $N_{G_i}(u)$ as $N_i(u)$. We define $V_i := Z_i \setminus Z_{i+1}$ for all $i \in [L - 1]$, and $V_L := Z_L$. We say that V_i is the i^{th} *level* of the decomposition, and define the *level* $\ell(u)$ of any node $u \in V_i$ as $\ell(u) := i$. We define $\deg^+(u) := \deg_{\ell(u)}(u)$ and $N^+(u) := N_{\ell(u)}(u)$ for $u \in V$. Given an edge $e = (u, v)$, we define the level $\ell(e)$ of e as $\ell(e) := \min\{\ell(u), \ell(v)\}$. Note also that for all $u \in V \setminus V_L$, $\deg^+(u) \leq \beta d$. However, given some $u \in V_L$, $\deg^+(u)$ may be much larger than βd, which motivates the following useful fact concerning such decompositions.

▶ **Lemma 4** ([6]). *Let $G = (V, E)$ be an arbitrary graph with arboricity α, let β, ϵ, d be any parameters such that $\beta \geq 1$, $0 < \epsilon < 1$, $d \geq 2(1 + \epsilon)\alpha$, and let $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$. Then for any (β, d, L)-decomposition $(Z_1, ..., Z_L)$ of G, it holds that $Z_L = \varnothing$.*

Proof. Let $(Z_1, ..., Z_L)$ be a (β, d, L)-decomposition of G satisfying the conditions of the lemma. Let i be an arbitrary index in $[L - 1]$. Since the arboricity of G_i is at most α, the average degree in G_i is at most 2α. On the other hand, by definition, the degree of any node in Z_{i+1} in the graph G_i is at least $d \geq 2(1 + \epsilon)\alpha$. It follows that

$$2(1 + \epsilon)\alpha|Z_{i+1}| \leq \sum_{u \in Z_{i+1}} \deg_i(u) \leq \sum_{u \in Z_i} \deg_i(u) \leq 2\alpha|Z_i|,$$

and hence $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$. Inductively, we obtain $|Z_L| \leq (1+\epsilon)^{1-L}|Z_1| \leq 1/(1+\epsilon) < 1$, yielding $Z_L = \varnothing$. ◀

Orienting the Edges. For our purposes, it will be useful to think of a decomposition of G as inducing an *orientation* of the edges. In particular, given an edge $e = (u, v)$, *we orient the edge from the endpoint of lower level towards the endpoint of higher level.* If the two endpoints have the same level, we orient the edge arbitrarily. We write $u \prec v$ to denote that the edge e is oriented from u to v. Note that $\deg^+(u)$ is an upper bound on the out-degree of u with respect to this orientation of the edges.

Dynamic Decompositions. Bhattacharya et al. give a deterministic fully dynamic data structure that can be used to explicitly maintain a (β, d, L)-decomposition of a graph $G = (V, E)$ under edge updates with small amortized update time. This algorithm also has small amortized recourse, where the recourse of an update is defined as the number of edges that change level following the update. The following theorem, from Section 4.1 of [6], will be used as a black box in our dynamic algorithm.

▶ **Proposition 5** ([6]). *For any constant $\beta \geq 2 + 3\epsilon$, there is a deterministic fully-dynamic algorithm that maintains a (β, d, L)-decomposition of a graph $G = (V, E)$ with amortized update time and amortized recourse both bounded by $O(L/\epsilon)$.*

It is straightforward to modify this dynamic algorithm to explicitly maintain the orientation of the edges that we described above without changing its asymptotic behavior. Furthermore, we can assume that the orientation of an edge changes *only* when it changes level.

2.4 Graph Decomposition Systems

In order for our dynamic algorithm to be able to deal with dynamically changing arboricity α, we will need to give a slight generalization of Definition 3, which we refer to as a *decomposition system*. Intuitively, this will enable us to maintain multiple decompositions, one for each "guess" of the arboricity, allowing us to use whichever decomposition is most appropriate to modify the edge coloring while handling an update.

▶ **Definition 6.** *Given a graph $G = (V, E)$, $\beta \geq 1$, a sequence $(d_j)_{j \in [K]}$ such that $d_j \geq 0$, and a positive integer L, a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of G is a sequence $(Z_{i,j})_{i\in[L],j\in[k]}$ of node sets, where for each $j \in [K]$, $(Z_{i,j})_{i\in[L]}$ is a (β, d_j, L)-decomposition of G.*

Given a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of $G = (V, E)$, we denote the graph $G[Z_{i,j}]$ by $G_{i,j}$, $\deg_{G_{i,j}}(u)$ by $\deg_{i,j}(u)$, and $N_{G_{i,j}}(u)$ by $N_{i,j}(u)$ for $u \in V$. We say that $(Z_{i,j})_i$ is the j^{th} *layer* of the decomposition system. We denote by $\ell_j(u)$ the level of node u in the decomposition $(Z_{i,j})_i$ and define $\deg^+_j(u) := \deg_{\ell_j(u),j}(u)$ and $N^+_j(u) := N_{\ell_j(u),j}(u)$ for $u \in V$.

Given a node u, we define the *layer* of u as $\mathcal{L}(u) = \min\{j \in [K] \mid \ell_j(u) < L\}$. Given an edge $e = (u, v)$, we define the layer of e as $\mathcal{L}(e) = \min\{\mathcal{L}(u), \mathcal{L}(v)\}$. We denote the orientation of the edges induced by the decomposition $(Z_{i,j})_i$ by $\prec_j$.

We can use the data structure from Proposition 5 to dynamically maintain a decomposition system, giving us the following proposition. In this context, we define the recourse of an update to be the number of edges that change levels in *some* layer.

▶ **Proposition 7.** *For any constant $\beta \geq 2 + 3\epsilon$, there is a deterministic fully dynamic algorithm that maintains a $(\beta, (d_j)_{j\in[K]}, L)$-decomposition system of a graph $G = (V, E)$ with amortized update time and amortized recourse $O(KL/\epsilon)$.*

As before, we assume that the orientation of an edge e with respect to $\prec_j$ changes only when $\ell_j(e)$ changes.

3 A Warmup Dynamic Algorithm (for Fixed α)

We now turn our attention towards designing an algorithm that can *dynamically* maintain a $(\Delta + O(\alpha))$-edge coloring of the graph G as it changes over time. A starting point for creating such an algorithm is the static algorithm that we outline in Section 1.2. Unfortunately, the highly sequential nature of this algorithm makes it very challenging to dynamize directly, as it is not clear how to efficiently maintain the output in the dynamic setting. In order to overcome this obstacle, we use the notion of graph decompositions (see Section 2.3). Informally, these graph decompositions can be interpreted as an "approximate" version of the sequence in which the static algorithm colors the edges in the graph – where instead of peeling off a node with smallest degree one at a time, we peel off large batches of nodes with sufficiently small degrees simultaneously. This leads to a "more robust" structure that can be maintained dynamically in an efficient manner.

Let $G = (V, E)$ be a dynamic graph that undergoes updates via edge insertions and deletions. In this section, we work in a simpler setting where we assume that we are given an α and are guaranteed that the maximum arboricity of the graph G remains at most α throughout the entire sequence of updates. We then give a deterministic fully dynamic algorithm that maintains a $(\Delta + O(\alpha))$-edge coloring of G, where Δ is an upper bound on the maximum degree of G at any point throughout the entire sequence of updates.[7] Without dealing with implementation details, we show that it achieves $\tilde{O}(1)$ worst-case recourse per update. In Section 4, we extend our result to the setting where Δ and α are not bounded and show how to maintain a $(\Delta + O(\alpha))$-edge coloring of G where α and Δ are the *current* arboricity and maximum degree of G respectively and change over time.

3.1 Algorithm Description

For the rest of this section, fix some constants ϵ, β, and L such that: $0 < \epsilon < 1$, $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$. At a high level, our algorithm works by dynamically maintaining a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_{i=1}^{L}$ of the graph G by using Proposition 5. During an update, our algorithm first updates the decomposition $(Z_i)_i$, and then uses this decomposition to find a path of length at most L such that, by only changing the colors assigned to the edges in this path, it can update the coloring to be valid for the updated graph. Since $L = \tilde{O}(1)$, this immediately implies the worst-case recourse bound. Algorithm 1 gives the procedure that we call to initialize our data structure, creating a decomposition of the empty graph, and Algorithms 2 and 3 give the procedures called when handling insertions and deletions respectively.

Algorithm 1 INITIALIZE(G, α).

```
Input: An empty graph $G = (V, \varnothing)$ and a parameter $\alpha$
Create a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_{i \in [L]}$ of $G$
```

Algorithm 2 INSERT(e).

```
Input: An edge $e$ to be inserted into $G$
Insert the edge $e$ into $G$
$\chi(e) \leftarrow \perp$
Update the $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of $G$
ExtendColoring$(e, (Z_i)_i)$
```

[7] Note that the algorithm needs prior knowledge of α, but not Δ.

Algorithm 3 DELETE(e).

```
Input: An edge $e$ to be deleted from $G$
Delete the edge $e$ from $G$
$\chi(e) \leftarrow \perp$
Update the $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of $G$
```

Algorithm 4 EXTENDCOLORING($e, (Z_i)_i$).

```
Input: An uncolored edge $e$ and a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition $(Z_i)_i$ of $G$
$S \leftarrow \{e\}$
while $S \neq \emptyset$ do
    Let $f = (u, v)$ be any edge in $S$ where $u \prec v$
    $C_u^+ \leftarrow \chi(N^+(u))$
    $C_v \leftarrow \chi(N(v))$
    Set $c$ to any element in $[|C_u^+| + |C_v| + 1] \setminus (C_u^+ \cup C_v)$
    if $c \in \chi(N(u))$ then
        Let $f'$ be the edge in $N(u)$ with $\chi(f') = c$
        $\chi(f') \leftarrow \perp$ and $S \leftarrow S \cup \{f'\}$
    $\chi(f) \leftarrow c$ and $S \leftarrow S \setminus \{f\}$
```

The following theorem, which we prove next, summarizes the behavior of our warmup dynamic algorithm.

▶ **Theorem 8.** *The warmup dynamic algorithm is deterministic and, given a sequence of updates for a dynamic graph G and a value α such that the arboricity of G never exceeds α, maintains a $(\Delta + (4+\epsilon)\alpha)$-edge coloring, where Δ is the maximum degree of G throughout the entire sequence of updates. The algorithm has $O(\log n/\epsilon)$ worst-case recourse per update and $O(\log^2 n \log \Delta/\epsilon^2)$ amortized update time.*

3.2 Analysis of the Warmup Algorithm

We now show that the warmup algorithm maintains a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring and has a worst-case recourse of at most $L = O(\log n/\epsilon)$ per update.[8]

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph with maximum degree at most Δ and arboricity at most α. Let e be an edge in G, $(Z_i)_i$ a $(\beta, 2(1+\epsilon)\alpha, L)$-decomposition of G and χ a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of $G - e$. Then running* EXTENDCOLORING$(e, (Z_i)_i)$*:*

1. *changes the colors of at most L edges in G, and*
2. *turns χ into a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of G.*

Proof. We first prove (1). Let e_i denote the edge that is uncolored at the start of the i^{th} iteration of the while loop as we run the procedure. Let $\ell(e_i)$ denote the minimum of the level of both of its endpoints. Clearly $\ell(e_i) \leq L$ since this is the highest level and $\ell(e_i) \geq 1$ for all i since this is the lowest level. Suppose the while loop iterates at least i times for some integer $i \geq 2$. Let $e_{i-1} = (u, v)$ where $u \prec v$, and hence $\ell(u) \leq \ell(v)$ (see Section 2.3). Since $e_i \in N(u)$ during iteration $i-1$ but $\chi(e_i) \notin \chi(N^+(u))$, we have that $e_i \notin N^+(u)$, and hence the endpoint of e_i that is not u appears in a level strictly below the level of u, so

[8] Note that $2\beta(1+\epsilon)\alpha = (4 + O(\epsilon))\alpha$.

$\ell(e_i) < \ell(e_{i-1})$. It follows that $1 \leq \ell(e_i) \leq L + 1 - i$, so the while loop iterates at most L times. For (2), note that if we let $e_i = (u, v)$ where $u \prec v$, then $|C_u^+| = \deg^+(u) - 1$ and $|C_v| = \deg(v) - 1$, so

$$|C_u^+| + |C_v| + 1 \leq \deg^+(u) + \deg(v) - 1 \leq \Delta + 2\beta(1+\epsilon)\alpha,$$

and so the procedure never assigns any e_i a color larger than $\Delta + 2\beta(1+\epsilon)\alpha$. Since we know from (1) that the procedure terminates after at most L iterations, after which every edge in the graph is colored, and χ was a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of the graph $G - e_1$ at the start of the procedure, it follows by induction that after the procedure terminates χ assigns each edge in G a color from $[\Delta + 2\beta(1+\epsilon)\alpha]$. Furthermore, our algorithm can only terminate if this assignment forms a valid edge coloring. Hence, χ is a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of G. ◀

▶ **Lemma 10.** *The warmup algorithm maintains a $(\Delta + 2\beta(1+\epsilon)\alpha)$-edge coloring of the graph.*

Proof. We prove this by induction. Since G is initially empty, the empty map is trivially a coloring of G. Let $\lambda = \Delta + 2\beta(1+\epsilon)\alpha$. Suppose χ is a λ-edge coloring of G after the i^{th} update. If the $i + 1^{th}$ update is a deletion, χ is still a λ-edge coloring of the updated graph and we are done. If the $i + 1^{th}$ update is an insertion, then we run Algorithm 4 in order to update χ. By part (2) of Lemma 9, it follows that χ is a λ-edge coloring of the updated graph once the procedure terminates. ◀

▶ **Lemma 11.** *The warmup algorithm changes the colors of at most L edges while handling an update.*

Proof. While handling the deletion of an edge e, our algorithm uncolors the edge e and does not change the color of any other edge. While handling the insertion of an edge e, our algorithm only changes the colors of edges while handling the call to ExtendColoring$(e, (Z_i)_i)$. By part (1) of Lemma 9, this changes the colors of at most L edges. ◀

In the full version of our paper, we prove the following lemma.

▶ **Lemma 12.** *The warmup algorithm has an amortized update time of $O(\log^2 n \log \Delta/\epsilon^2)$.*

We also note that Corollary 2 follows immediately from Lemma 9. In particular, if we set $\beta = 1$, by Lemma 4, the proof Lemma 9 still holds. Hence, we can use ExtendColoring along with *any* $(1, 2(1+\epsilon)\alpha, L)$-decomposition of G in order to extend any $(\Delta+2(1+\epsilon)\alpha)$-edge coloring χ with an uncolored edge e so that the edge e is now colored by only changing the colors of $O(\log n/\epsilon)$ many edges.

4 The Dynamic Algorithm

We now describe our full dynamic algorithm and show that it maintains a $(\Delta + O(\alpha))$-edge coloring of the graph. We then use Proposition 7 to show that we can get $\tilde{O}(1)$ amortized recourse. In Appendix B of the full version of our paper, we describe the relevant data structures and use them to implement our algorithm to get $\tilde{O}(1)$ amortized update time.

4.1 Algorithm Description

In order to describe our algorithm, we fix some constant ϵ such that $0 < \epsilon < 1$ and set $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$. Let $\tilde{\alpha}_j := (1+\epsilon)^{j-1}$ and note that, for any n-node graph G with arboricity α, $\tilde{\alpha}_1 = 1 \leq \alpha \leq n < \tilde{\alpha}_L$.

Informal Description. Our algorithm works by maintaining the invariant that each edge $e = (u, v)$ receives a color in the set $[\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(e)})]$, where $u \prec_{\mathcal{L}(e)} v$. Since $\deg(v) \le \Delta$ and $\tilde{\alpha}_{\mathcal{L}(e)} = O(\alpha)$ (see Lemma 15), it follows that the algorithm uses at most $\Delta + O(\alpha)$ many colors. When an edge is inserted or deleted, this may cause some $\tilde{O}(1)$ many edges to violate the invariant. We begin by first identifying all such edges and uncoloring them. We then update the decomposition system maintained by our algorithm, which may again cause some $\tilde{O}(1)$ many edges (on average) to violate the invariant. We again identify and uncolor all such edges. We now want to color each of the uncolored edges, while ensuring that we satisfy this invariant at all times. We do this by using the decomposition system maintained by our algorithm: we take an uncolored edge $f = (u, v)$ such that $u \prec_{\mathcal{L}(f)} v$ and assign it a color c that is *not* assigned to any of the edges in $N^+_{\mathcal{L}(f)}(u)$ or $N(v)$. If there is an edge f' adjacent to f that is also colored with c, we uncolor this edge. We repeat this process iteratively until all edges are colored. We can show that: (1) there are at most $\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(f)})$ many edges in $N^+_{\mathcal{L}(f)}(u) \cup N(v)$, and hence we can find such a c in the palette $[\deg(v) + O(\tilde{\alpha}_{\mathcal{L}(f)})]$, and (2) if there is such an edge f' adjacent to f that is also colored with c, then either $\ell_{\mathcal{L}(f')}(f') < \ell_{\mathcal{L}(f)}(f)$ or $\mathcal{L}(f') < \mathcal{L}(f)$, allowing us to carry out a potential function argument that shows that the process terminates with all edges colored after $\tilde{O}(1)$ iterations on average, giving us an amortized recourse bound.

Formal Description. The following pseudo-code gives a precise formulation of our algorithm.

Algorithm 5 INITIALIZE(G).

```
Input: An empty graph $G = (V, \varnothing)$
Create a $(\beta, (2(1+\epsilon)\tilde{\alpha}_j)_{j\in[L]}, L)$-decomposition system $(Z_{i,j})_{i,j\in[L]}$ of $G$
```

Algorithm 6 INSERT(e).

```
Input: An edge $e$ to be inserted into $G$
Insert the edge $e$ into $G$
$S \leftarrow$ UPDATEDECOMPOSITIONS($e$)
$\chi(f) \leftarrow \perp$ for all $f \in S$
EXTENDCOLORING($S$)
```

Algorithm 7 DELETE(e).

```
Input: An edge $e$ to be deleted from $G$
Delete the edge $e$ from $G$
$S \leftarrow \varnothing$
for $v \in e$ do
    $S \leftarrow S \cup \{f = (u, v) \in N(v) \mid u \prec_{\mathcal{L}(f)} v$ and $\chi(f) > \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}\}$
$S \leftarrow S \cup$ UPDATEDECOMPOSITIONS($e$)
$\chi(f) \leftarrow \perp$ for all $f \in S$
EXTENDCOLORING($S$)
```

Algorithm 8 UPDATEDECOMPOSITIONS(e).

```
Input: The edge $e$ that has been inserted/deleted from $G$
Update the decomposition system $(Z_{i,j})_{i,j}$
Let $S' \subseteq E$ be the set of all edges whose level changes in some layer
return $S'$
```

Algorithm 9 ExtendColoring(S).

```
Input: A set $S$ of uncolored edges
while $S \neq \varnothing$ do
    Let $f = (u, v)$ be any edge in $S$ where $u \prec_{\mathcal{L}(f)} v$
    $C_u^+ \leftarrow \chi(N^+_{\mathcal{L}(f)}(u))$
    $C_v \leftarrow \chi(N(v))$
    Let $c$ be any element in $[|C_u^+| + |C_v| + 1] \setminus (C_u^+ \cup C_v)$
    if $c \in \chi(N(u))$ then
        Let $f'$ be the edge in $N(u)$ with $\chi(f') = c$
        $\chi(f') \leftarrow \perp$ and $S \leftarrow S \cup \{f'\}$
    $\chi(f) \leftarrow c$ and $S \leftarrow S \setminus \{f\}$
```

The following theorem, which we prove next, summarizes the behavior of our full dynamic algorithm.

▶ **Theorem 13.** *The dynamic algorithm is deterministic and, given a sequence of updates for a dynamic graph G, maintains a $(\Delta + (4 + \epsilon)\alpha)$-edge coloring, where Δ and α are the dynamically changing maximum degree and arboricity of G, respectively. The algorithm has $O(\log^4 n/\epsilon^5)$ amortized recourse per update and $O(\log^5 n \log \Delta/\epsilon^6)$ amortized update time.*[9]

We split the proof of Theorem 13 into two parts. In Section 4.2, we show that our dynamic algorithm maintains a $(\Delta + 2\beta(1 + \epsilon)^2\alpha)$-edge coloring and has an amortized recourse of $O(\log^4 n/\epsilon^5)$.[10] In Appendix B of the full version of our paper, we describe the data structures used by our algorithm, before showing how to use them in order to get $O(\log^5 n \log \Delta/\epsilon^6)$ amortized update time.

4.2 Analysis of the Dynamic Algorithm

For the rest of Section 4.2, fix a dynamic graph $G = (V, E)$, and a $(\beta, (2(1 + \epsilon)\tilde{\alpha}_j)_j, L)$-decomposition system $\mathcal{Z} = (Z_{i,j})_{i,j}$ of G. Recall that ϵ is a fixed constant with $0 < \epsilon < 1$, and that $\beta = 2 + 3\epsilon$, $L = 2 + \lceil \log_{1+\epsilon} n \rceil$.

We begin with the following simple observations.

▶ **Lemma 14.** *For all nodes $u \in V$, we have that $\mathcal{L}(u) \leq j^\star$, where $j^\star \in [L]$ is the unique value such that $\alpha \leq \tilde{\alpha}_{j^\star} < (1 + \epsilon)\alpha$.*

Proof. By Lemma 4, we know that $Z_{L,j^\star} = \varnothing$. Hence, $\mathcal{L}(u) \leq j^\star$ for every node $u \in V$. ◀

▶ **Corollary 15.** *For all edges $e \in E$, we have that $\tilde{\alpha}_{\mathcal{L}(e)} < (1 + \epsilon)\alpha$.*

We now define the notation of a *good* edge coloring. In such an edge coloring, the colors satisfy certain locality constraints, which makes it easier to maintain dynamically.

▶ **Definition 16.** *Given an edge coloring χ of the graph G, we say that χ is a* good *edge coloring of G with respect to the decomposition system $\mathcal{Z}$ if and only if for every edge $e = (u, v) \in E$ such that $\chi(e) \neq \perp$ and $u \prec_{\mathcal{L}(e)} v$, we have that $\chi(e) \leq \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(e)}$.*

[9] Whenever the term Δ appears in an amortized bound, this should be interpreted as being an upper bound on the maximum degree across the whole sequence of updates. In the introduction, we replaced the $\log \Delta$ term with $\log n$ for simplicity.

[10] Note that $2\beta(1 + \epsilon)^2\alpha = (4 + O(\epsilon))\alpha$.

The following lemma shows that our algorithm can be used to maintain a good edge coloring.

▶ **Lemma 17.** *Let χ be a good edge coloring of the graph G w.r.t. $\mathcal{Z}$ and let $S \subseteq E$ be the set of edges that are left uncolored by χ. Then running* ExtendColoring(S)*:*

1. *changes the colors of at most $L^2|S|$ edges in G, and*
2. *turns χ into a good edge coloring with no uncolored edges.*

Proof. We begin by proving (1). Given some edge f, define the potential of f by

$$\Psi(f) = L(\mathcal{L}(f) - 1) + \ell_{\mathcal{L}(f)}(f).$$

Given the set of edges S, define the potential of S as $\Psi(S) = \sum_{f \in S} \Psi(f)$. By Lemma 14, we have that, for any edge f, $1 \leq \Psi(f) = L(\mathcal{L}(f) - 1) + \ell_{\mathcal{L}(f)}(f) \leq L(L-1) + L = L^2$. Hence, $|S| \leq \Psi(S) \leq L^2|S|$. During each iteration of the while loop in Algorithm 9, exactly one edge receives a new color (and at most one edge becomes uncolored). We now show that during each iteration of the loop, $\Psi(S)$ drops by at least one, implying that we have at most $L^2|S|$ iterations in total, changing the colors of at most $L^2|S|$ many edges. Let f be the edge in S that we are coloring during some iteration of the loop and let c be the color that it receives. During the iteration, we remove f from S; furthermore, if there exists some edge f' colored with c that shares an endpoint with f, we uncolor f' and place it in S. If there is no such edge f', then $\Psi(S)$ drops by at least 1 since we remove f from S and $\Psi(f) \geq 1$. Suppose that there is such an edge f'. We now argue that $\Psi(f') < \Psi(f)$. We first note that one of the endpoints of f' is not contained in $Z_{i,j}$ where $i = \ell_{\mathcal{L}(f)}(u)$ and $j = \mathcal{L}(f)$. This implies that $\ell_{\mathcal{L}(f)}(f') < \ell_{\mathcal{L}(f)}(f)$, so $\mathcal{L}(f') \leq \mathcal{L}(f)$. Hence, if $\mathcal{L}(f) = \mathcal{L}(f')$, it follows that $\Psi(f') < \Psi(f)$. Otherwise, $\mathcal{L}(f') < \mathcal{L}(f)$, and we have that

$$\Psi(f) - \Psi(f') = L(\mathcal{L}(f) - \mathcal{L}(f')) + \ell_{\mathcal{L}(f)}(f) - \ell_{\mathcal{L}(f')}(f') \geq L + (1 - L) \geq 1.$$

In either case, $\Psi(S)$ drops by at least 1. We now prove (2). Let $f = (u, v)$ be the edge in S that we are coloring during some iteration of the while loop such that $u \prec_{\mathcal{L}(f)} v$. We need to show that the color c picked by the algorithm satisfies $c \leq \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$. It will then follow by induction that the coloring produced by calling ExtendColoring(S) is good given that we start with a good coloring. We first note that $|C_v| \leq \deg(v) - 1$. Now note that $|C_u^+| \leq \deg^+_{\mathcal{L}(f)}(u) - 1$. Since $\deg^+_{\mathcal{L}(f)}(u) \leq 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$, we get the desired bound on c. Finally, note that at the start of each iteration, the uncolored edges correspond to exactly the edges in S. Since the algorithm terminates if and only if $S = \varnothing$ and we know that the algorithm terminates after at most $L^2|S|$ many iterations, it follows that the resulting coloring has no uncolored edges. ◀

▶ **Lemma 18.** *The dynamic algorithm maintains a $(\Delta + 2\beta(1+\epsilon)^2\alpha)$-edge coloring of the graph.*

Proof. By showing that our algorithm maintains a good edge coloring, it follows by Corollary 15 that, for any edge $e \in E$, we have $\chi(e) \leq \Delta + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(e)} \leq \Delta + 2(1+\epsilon)^2\alpha$. We do this by showing that, after an update, the algorithm uncolors all of the edges $f = (u, v)$ in the graph that don't satisfy the condition $\chi(f) \leq \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$ for $u \prec_{\mathcal{L}(f)} v$ in the updated decomposition system, places them in a set S, and calls Algorithm 9 on the set S. By Lemma 17, it then follows that the algorithm maintains a good coloring of the entire graph.

We refer to an edge $e = (u, v)$ as *bad* if it does not satisfy the condition required by a good coloring, i.e. if $\chi(e) \neq \perp$ and $\chi(e) > \deg(v) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$ where $u \prec_{\mathcal{L}(f)} v$. Suppose we have a good edge coloring of the entire graph and insert an edge e into the graph. Since this cannot decrease the degrees of any nodes or change the levels of any edges (since we have not yet updated the decomposition system) this cannot cause any edges to become bad. On the other hand, if we delete an edge $e = (u, v)$, some of the edges incident to u and v might become bad since $\deg(u)$ and $\deg(v)$ decrease by 1. Any such edges that become bad must be contained within the set $\Gamma_u \cup \Gamma_v$ where

$$\Gamma_w = \{f = (w', w) \in N(w) \mid w' \prec_{\mathcal{L}(f)} w \text{ and } \chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}\}$$

where the degrees are w.r.t. the state of the graph G *after* the deletion of e. If we uncolor all of the edges in $\Gamma_u \cup \Gamma_v$, we restore χ to being a good edge coloring. After updating the decomposition system, the levels of some edges might change in some layers. Any edge that does not change levels in any layer will *not* become bad, since $\mathcal{L}(f)$ (and hence $\tilde{\alpha}_{\mathcal{L}(f)}$) and its orientation in $\prec_{\mathcal{L}(f)}$ do not change. However, an edge f that changes levels in some layer might become bad if $\mathcal{L}(f)$ decreases (causing the value of $\tilde{\alpha}_{\mathcal{L}(f)}$ to decrease) or if its orientation with respect to $\prec_{\mathcal{L}(f)}$ changes. Hence, we uncolor all such edges.[11] This guarantees that there are no bad edges when we call ExtendColoring. Since we give ExtendColoring all of the edges that are uncolored, it follows that we maintain a good edge coloring of the entire graph. ◀

▶ **Lemma 19.** *The dynamic algorithm has $O(\log^4 n/\epsilon^5)$ amortized recourse per update.*

Proof. Suppose that our algorithm handles a sequence of T updates (edge insertions or deletions) starting from an empty graph. Let $S^{(t)}$ denote the set of edges uncolored by our algorithm during the t^{th} update before calling ExtendColoring on the set $S^{(t)}$. By Lemma 17, we know that at most $L^2|S^{(t)}| = O(|S^{(t)}| \log^2 n/\epsilon^2)$ many edges will change color during this update. By showing that $(1/T) \cdot \sum_{t \in [T]} |S^{(t)}|$ is $O(\log^2 n/\epsilon^3)$, our claimed amortized recourse bound follows. Now fix some $t \in [T]$ and let $e = (u, v)$ be the edge being either inserted or deleted during this update. The edges uncolored by the algorithm while handling this update are either contained in the set $\Gamma_u \cup \Gamma_v$ (if the update is a deletion) or change levels in some layer after we update the decomposition system. There can only be at most $2L$ many edges of the former type. This is because, given some $j \in [L]$, there is at most one edge $f \in \Gamma_w$ with $\mathcal{L}(f) = j$ such that $\chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)}$. Otherwise, since all the edges incident on w have distinct colors, there exists such an edge f such that $\chi(f) > \deg(w) + 2\beta(1+\epsilon)\tilde{\alpha}_{\mathcal{L}(f)} + 1$, which contradicts the fact that χ was a good coloring of the graph before the deletion of e. It follows that $|\Gamma_w \cap \mathcal{L}^{-1}(j)| \leq 1$, so

$$|\Gamma_w| = \sum_{j \in [L]} |\Gamma_w \cap \mathcal{L}^{-1}(j)| \leq L$$

and hence $|\Gamma_u \cup \Gamma_v| \leq 2L$. To bound the number of edges that changed levels in at least one of the decompositions in the decomposition system, recall (see Proposition 7) that the amortized recourse of the algorithm that maintains the decomposition system is $O(L^2/\epsilon)$. It follows that the amortized number of such edges is $O(L^2/\epsilon)$. We have that

$$\frac{1}{T} \cdot \sum_{t \in [T]} |S^{(t)}| = O\left(\frac{L^2}{\epsilon}\right) + 2L = O\left(\frac{\log^2 n}{\epsilon^3}\right).$$ ◀

[11] Note that these are precisely the edges that contribute towards the recourse of the dynamic decomposition system.

References

1 Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed $(\Delta + o(\Delta))$-edge-coloring, and vertex-coloring of graphs with bounded diversity. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 175–184. ACM, 2017.

2 Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland*, volume 108 of *Procedia Computer Science*, pages 89–98. Elsevier, 2017.

3 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018.

4 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Density-sensitive algorithms for $(\Delta + 1)$-edge coloring. *CoRR*, abs/2307.02415, 2023. `arXiv:2307.02415`.

5 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Nibbling at long cycles: Dynamic (and static) edge coloring in optimal time. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3393–3440. SIAM, 2024.

6 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM, 2015.

7 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2633–2652. SIAM, 2018.

8 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

9 Aleksander B. J. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. *CoRR*, abs/2311.10616, 2023. `arXiv:2311.10616`.

10 Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1013–1026, 2023.

11 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1937–1945. SIAM, 2019.

12 Łukasz Kowalik. Edge-coloring sparse graphs with Δ colors in quasilinear time. *arXiv preprint*, 2024. `arXiv:2401.13839`.

13 V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.

On the Independence Number of 1-Planar Graphs

Therese Biedl
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Babak Miraftab
School of Computer Science, Carleton University, Ottawa, Canada

Abstract

An independent set in a graph is a set of vertices where no two vertices are adjacent to each other. A maximum independent set is the largest possible independent set that can be formed within a given graph G. The cardinality of this set is referred to as the independence number of G. This paper investigates the independence number of 1-planar graphs, a subclass of graphs defined by drawings in the Euclidean plane where each edge can have at most one crossing point. Borodin establishes a tight upper bound of six for the chromatic number of every 1-planar graph G, leading to a corresponding lower bound of $n/6$ for the independence number, where n is the number of vertices of G. In contrast, the upper bound for the independence number in 1-planar graphs is less studied. This paper addresses this gap by presenting upper bounds based on the minimum degree δ. A comprehensive table summarizes these upper bounds for various δ values, providing insights into achievable independence numbers under different conditions.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases 1-planar graph, independent set, minimum degree

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.13

Funding *Therese Biedl*: Supported by NSERC; FRN RGPIN-2020-03958.
Prosenjit Bose: Supported by NSERC.
Babak Miraftab: Supported by NSERC.

1 Introduction

An *independent set* in a graph contains vertices that are not adjacent to each other. A *maximum independent set* is an independent set of largest possible size for a given graph, and the number of vertices in this set is known as the *independence number* of G and denoted by $\alpha(G)$. The size $\alpha(G)$ serves as a crucial parameter in graph theory and holds significance in algorithmic contexts. For instance, Kirkpatrick [24] and Dobkin and Kirkpatrick [19] employed the repeated removal of independent sets from triangulations to devise data structures for efficient planar point location and distance computation between convex polytopes, respectively. Biedl and Wilkinson [6] explored the size of independent sets in bounded degree triangulations. In addition, Bose, Dujmović and Wood [11] obtained graphs of bounded degree with large independent sets.

The celebrated 4-color theorem [3, 29] immediately implies that every planar graph contains an independent set of size at least $n/4$, where n is the number of vertices in the graph. Interestingly, this bound represents the maximum attainable, as there exist planar graphs without larger independent sets; for instance, consider disjoint copies of complete graphs with 4 vertices. Some weaker lower bounds are also established [1, 17] that circumvent the complexity of the 4-color theorem (as suggested by Erdős [5]) via charging and discharging arguments. Also, Caro and Roditty in [13] gives the following upper bound.

▶ **Theorem 1** ([13]). *Let G be a planar graph with minimum degree δ. Then $\alpha(G) \leq \frac{2n-4}{\delta}$.*

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 13; pp. 13:1–13:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In addition, they construct an infinite family of planar graphs with $\alpha(G) = \frac{2n-4}{\delta}$, where δ takes values of 3, 4, and 5. From an algorithmic standpoint, determining the maximum independent set in planar graphs is NP-hard, even when restricted to planar graphs of maximum degree 3 [21, 28] or planar triangle-free graphs [26]. Consequently, efforts have shifted towards approximating large independent sets through methods like approximation algorithms [2, 4, 12, 15, 20, 26], parallel algorithms [16, 18, 22], or within certain minor-free planar graphs [20, 25, 27].

There are various generalizations of planar graphs, for example a 1-*planar* graph is a graph that can be drawn in the Euclidean plane with at most one crossing per edge. In this paper, we study the independence number of 1-planar graphs. Borodin [10] establishes that every 1-planar graph G has a 6-coloring, therefore $\alpha(G) \geq n/6$. This is tight, for example a graph consisting of disjoint copies of K_6 is 1-planar and has chromatic number 6. Unlike planar graphs, there were no prior results on the upper bound for the independence number of 1-planar graphs under degree conditions.

Our Results. This paper aims to explore the upper bounds on the independence number of 1-planar graphs. We provide such upper bounds, relative to the minimum degree δ. Furthermore, we construct 1-planar graphs that (for most values of δ) match the bound, i.e., they have this minimum degree and have an independent set of that size. Our results are summarized in Table 1.

Table 1 Bounds on the independence number of 1-planar graphs of minimum degree δ and optimal 1-planar graphs. The upper bound means that no graph can have a bigger independent set, while the lower bound means that some 1-planar graph has an independent set of this size. Note that the bounds match (up to small additive constants) except for $\delta = 7$.

	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	Optimal 1-planar
Upper bound	$\frac{6}{7}(n-2)$	$\frac{2}{3}(n-2)$	$\frac{4}{7}(n-2)$	$\frac{1}{2}(n-3)$	$\frac{8}{20}(n-2)$	$\frac{1}{3}(n-2)$
Lower bound	$\frac{6}{7}(n-2)$	$\frac{2}{3}(n-2)$	$\frac{4}{7}(n-2)$	$\frac{1}{2}(n-4)$	$\frac{8}{21}(n-13.5)$	$\frac{1}{3}(n-2)$

We also study optimal 1-planar graphs, which are 1-planar graphs with the maximum possible number of edges, and for these, we give an upper bound $\frac{1}{3}(n-2)$ on the independence number. In addition, we show that this upper bound is tight by providing a family of optimal 1-planar graphs that achieve this bound.

Our paper is organized as follows. After giving preliminaries, we first construct in Section 3 a number of 1-planar graphs as a warm-up to introduce this graph class. Specifically, we provide infinite families of 1-planar graphs with large independent sets for minimum degree $\delta = 3, 4, 5, 6, 7$. In Section 4, we then present upper bounds for the independence number of 1-planar graphs with minimum degrees $\delta = 3, 4, 5, 6, 7$. Here for $\delta = 3, 4, 5$ the upper bounds are proven with some techniques that were used to bound the size of matchings in such graphs [7]. For $\delta = 6, 7$ we prove the upper bounds by expanding and generalizing some known results. Section 5 focuses on the independence number of optimal 1-planar graphs, before we conclude in Section 6 with some further thoughts.

2 Preliminaries

Let $G = (V, E)$ be a graph on n vertices. We assume familiarity with basic terms in graph theory, such as connectivity. We refer the reader to Bondy and Murty [9] for graph theoretic notations. Throughout the paper our input is always a connected graph $G = (V, E)$ on n vertices, and $n \geq 3$. We also use the letter T to denote an *independent set*, i.e., a set of vertices without edges between them. The notation $\overline{T}$ refers to the set of vertices $V \setminus T$.

A drawing Γ of a graph G assigns vertices to points in $\mathbb{R}^2$ and edges to curves in $\mathbb{R}^2$ in such a way that edge-curves join the corresponding endpoints. In this paper we only consider *good* drawings, see [30], where the following holds:

1. no vertex-points coincide and no edge-curve intersects a vertex-point except at its two ends;
2. if two edge-curves intersect at a point p that is not a common endpoint, then they properly cross at p;
3. if three or more edge-curves intersect at a point p, then p is a common endpoint of the curves;
4. if the curves of two edges e, e' intersect twice at points $p \neq p'$, then e, e' are parallel edges and p, p' are their endpoints; and
5. if the curve of an edge e self-intersects at point p, then e is a loop and p is its endpoint.

A drawing is called *k-planar* if each edge is involved in at most k crossings; a 0-planar drawing is simply called *planar*. In this paper, all drawings are 1-planar. A graph is called *planar/1-planar* if it has a planar/1-planar drawing, respectively.

For a given drawing Γ, the *cells* are the connected regions of $\mathbb{R}^2 \setminus \Gamma$; if Γ is planar then these are also called *faces*. The unbounded cell is also called the *outer face* (even for drawings that are not planar).

3 1-planar graphs with large independent sets

In this section, we construct several families of 1-planar graphs, each corresponding to a specified minimum degree denoted by δ. These graphs are designed to possess a large independent set. We first provide a general overview of how to construct them, and in each subsection, we elaborate with full details.

3.1 1-planar graphs with large independent sets for $\delta = 3, 4, 5, 6$

All but one of our constructions use a scheme that we call a *standard construction* which we explain here in general terms (see Figure 1). Fix three integer parameters s, k, τ, where $s \geq 1$ is arbitrary (it serves to make the graph as big as we wish), while k and τ will depend on the minimum degree δ.

The standard construction (illustrated in Figure 1) starts with s *nested k-cycles*, i.e., cycles of length k that are drawn (in the 1-planar drawing that we construct) such that each next cycle is inside the previous one. We will show our drawings on the *standing flat cylinder*, i.e., a rectangle where the left and right side have been identified; the nested cycles then become horizontal lines.

The s nested cycles define $s{+}1$ faces; of these, $s{-}1$ faces (the *middle faces*) are bounded by two disjoint k-cycles while two faces (the *end faces*) are bounded by one k-cycle. Consider one middle face, say it is bounded by k-cycles P and P'. We place τ vertices $t_1, \ldots, t_\tau$ inside this middle face; these vertices (over all middle faces together, plus possibly a few more at the end-faces) become our independent set T. These vertices are drawn in white in Figure 1. We make each t_i adjacent to $\lceil \delta/2 \rceil$ vertices on one of P, P' and $\lfloor \delta/2 \rfloor$ vertices on the other. With this, the vertices in T have degree δ. The main bottleneck for τ and k is that we must be able to place these vertices so that the drawing is 1-planar and simple. Another bottleneck is that all vertices in $\overline{T} := V \setminus T$ must have degree at least δ. Both claims will be mostly proved by illustrations outlining the 1-planar embeddings.

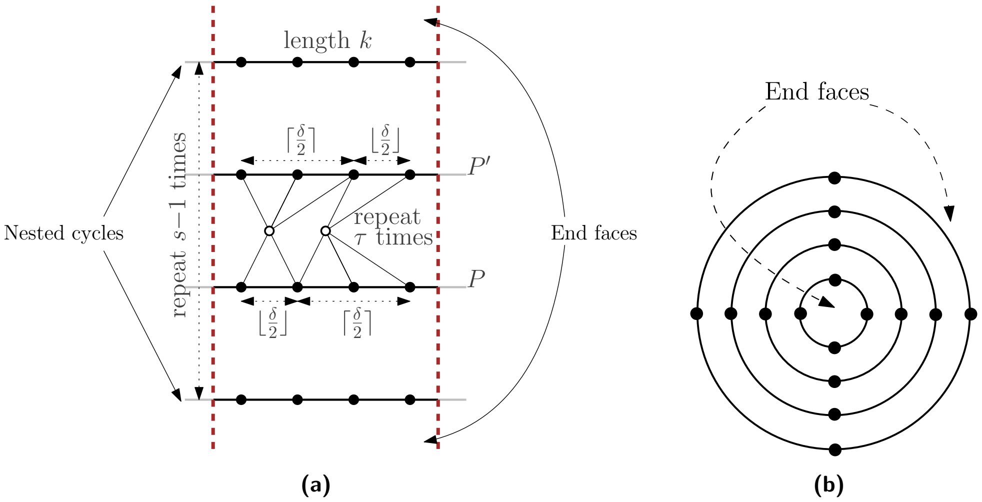

length k
Nested cycles
repeat $s-1$ times
$\lceil\frac{\delta}{2}\rceil$
$\lfloor\frac{\delta}{2}\rfloor$
P'
repeat τ times
P
$\lfloor\frac{\delta}{2}\rfloor$
$\lceil\frac{\delta}{2}\rceil$
End faces
End faces
(a)
(b)

Figure 1 The standard construction and the view when the graph is drawn in the plane.

The construction inside the end-faces depends very much on δ; sometimes we add nothing at all, sometimes we add edges, sometimes we add more vertices (in T or in $\overline{T}$ or both). The bottleneck is again that the vertices on the first/last nested cycle must have degree δ or more. In total the number of vertices is $n = s \cdot k + (s-1)\tau$ (plus whatever we added at the end-faces). The size of the independent set is $|T| = (s-1)\tau$ (plus whatever we added at the end-faces).

Now we give the specific constructions. (We should mention that for $\delta = 3, 4$ these are the same as the ones given in [7] to obtain 1-planar graphs for which the maximum matching is small, though described in a different way.)

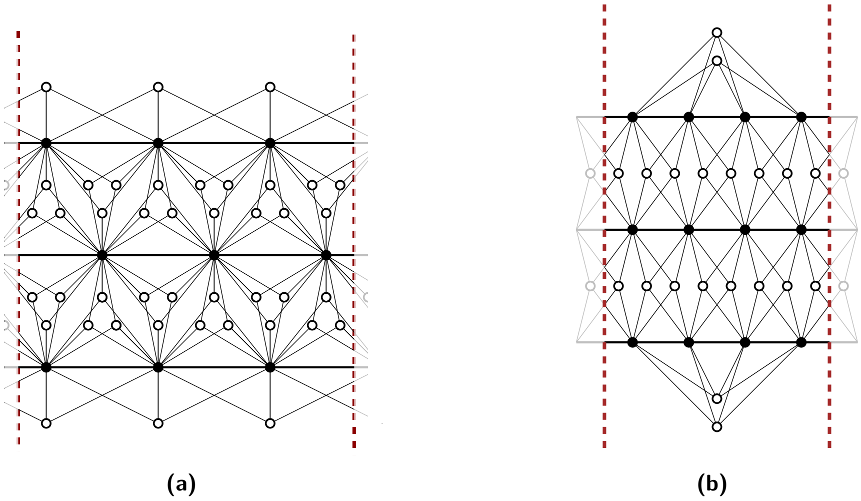


Figure 2 The graphs for $\delta = 3$ and $\delta = 4$ (for $s = 3$). Vertices in T are white, vertices in $\overline{T}$ are black.

▶ **Lemma 2.** *For any integer N and $\delta \in \{3, 4, 5, 6\}$, there exists a simple 1-planar graph with minimum degree δ and $n \geq N$ vertices with an independent set that has the size listed in Table 1 under "Lower bound".*

Proof. We follow the standard construction, choosing s big enough so that the resulting graph has at least N vertices. We choose k and τ as follows:

- For $\delta = 3$, we use $k = 3$ (so nested triangles) and $\tau = 18$. Into each end-face we add three more vertices of T that we make adjacent to all three vertices of the nested triangle that bounds the face. See Figure 2(a) to verify that this can be done such that the drawing is 1-planar and the minimum degree is 3. With this we have $n = 3s+18(s-1)+6 = 21s-12$ and $|T| = 18(s-1)+6 = 18s-12 = \frac{6}{7}(21s-14) = \frac{6}{7}(n-2)$.
- For $\delta = 4$, we use $k = 4$ (so nested quadrangles) and $\tau = 8$. Into each end-face we add two more vertices of T that we make adjacent to all four vertices of the nested quadrangle that bounds the face. See Figure 2(b) to verify that this can be done such that the drawing is 1-planar and the minimum degree is 4. With this we have $n = 4s+8(s-1)+4 = 12s-4$ and $|T| = 8(s-1)+4 = 8s-4 = \frac{2}{3}(12s-6) = \frac{2}{3}(n-2)$.
- For $\delta = 5$, we use $k = 12$ and $\tau = 16$. Into each end-face we add four more vertices of $\overline{T}$ connected as a path, and then 12 more vertices of T that we each make adjacent to two vertices of the path and three vertices of the 12-gon that bounds the face. See Figure 3, and verify that this can be done such that the drawing is 1-planar and the minimum degree is 5. With this we have $n = 12s + 16(s-1) + 32 = 28s + 16$ and $|T| = 16(s-1) + 24 = 16s + 8 = \frac{4}{7}(28s + 14) = \frac{4}{7}(n-2)$.

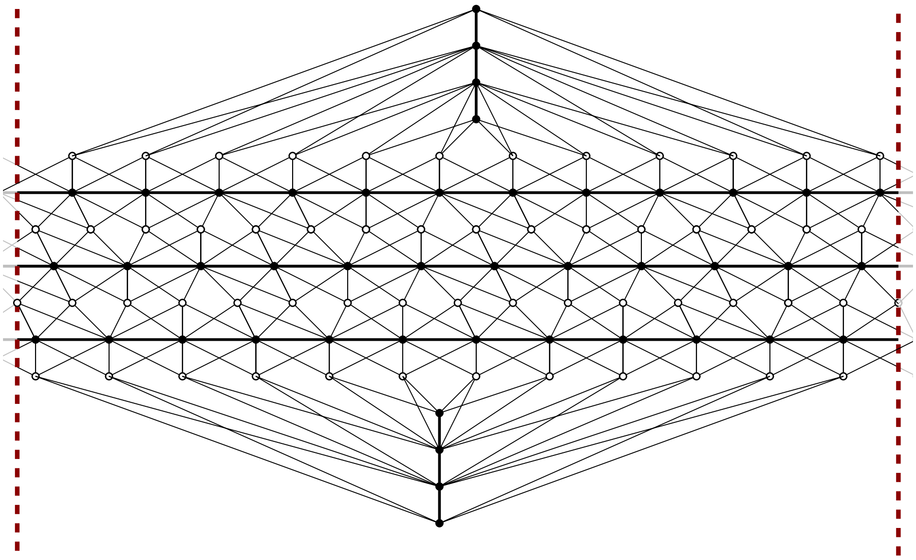


Figure 3 The graph for $\delta = 5$ (for $s = 3$).

- For $\delta = 6$, we use $k = 4$ (so nested quadrangles) and $\tau = 4$. Into each end-face we add a pair of crossing edges between the four vertices of the nested quadrangle that bounds the face. See Figure 4 to verify that this can be done such that the drawing is 1-planar and the minimum degree is 6. With this we have $n = 4s + 4(s-1) = 8s - 4$ and $|T| = 4(s-1) = 4s - 4 = \frac{1}{2}(8s-8) = \frac{1}{2}(n-4)$. ◀

3.2 1-planar graphs with large independent sets for $\delta = 7$

In this subsection, we show how to construct a 1-planar graph with minimum degree 7 and a large independent set. This does not use the construction from the previous section since it seems impossible to use equal-length nested cycles. Instead, we prove first a construction with desirable properties by induction, and then combine two such constructions into a graph with minimum degree 7.

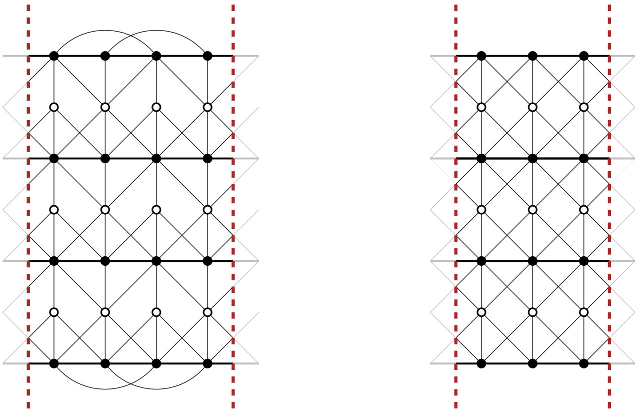


Figure 4 The graph for $\delta = 6$ (for $s = 4$). We also show a graph where all but 6 vertices have degree 6 that has an independent set of size $(n-3)/2$.

▷ Claim 3. For all $k \geq 0$, there exists a 1-planar graph G_k with $27 \cdot 2^k - 9$ vertices and an independent set T with $9 \cdot 2^k - 6$ vertices such that (in some 1-planar drawing)

- there are exactly $9 \cdot 2^k$ vertices on the outer-face, they form a cycle and each of them has degree at least 4,
- all other vertices have degree at least 7,
- no vertex of T is on the outer-face.

Proof. For the base case ($k = 0$), we need a graph with 18 vertices of which three form an independent set; see Figure 5(a) to verify all conditions.

Now assume that we have graph G_k with $9 \cdot 2^k$ vertices on the outer-face F_k. Insert $9 \cdot 2^k$ new vertices in F_k (let T_{k+1} be the set of added vertices) and make each of them adjacent to three vertices of F_k; Figure 5(b) shows that this can be done while retaining 1-planarity and keeping T_{k+1} on the outer-face. With this, all vertices in F_k receive three more neighbours and hence now have degree 7 or more. Insert $18 \cdot 2^k$ new vertices into the outer-face of the resulting graph, and connect them in a cycle that will form the outer-face F_{k+1} of the new graph G_{k+1}. Make each vertex of T_{k+1} adjacent to four vertices of F_{k+1}; the figure shows that can be done while remaining 1-planar. Also, with this all vertices on F_{k+1} receive two neighbours in T_{k+1}; this plus the cycle among them ensures that they have degree 4 while everyone else has degree at least 7. As desired T_{k+1} forms an independent set and has no edges to vertices of the independent set T_k of G_k since those are not on F_k by inductive hypothesis.

It remains to verify the claim on the size. Independent set $T_k \cup T_{k+1}$ has size $9 \cdot 2^k - 6 + 9 \cdot 2^k = 9 \cdot 2^{k+1} - 6$. The outer-face F_{k+1} of G_{k+1} has $18 \cdot 2^k = 9 \cdot 2^{k+1}$ vertices, and finally $|V(G_{k+1})| = |V(G_k)| + |T_{k+1}| + |F_{k+1}| = 27 \cdot 2^k - 9 + 9 \cdot 2^k + 18 \cdot 2^k = 27 \cdot 2^{k+1} - 9$. ◁

▶ **Lemma 4.** *For any integer N, there exists a simple 1-planar graph with minimum degree 7 and $n \geq N$ vertices with an independent set of size $\frac{8}{21}(n - 13.5)$.*

Proof. Let $k = \lceil \log_2((N+18)/63) \rceil$ and start with two copies of G_k, placed such that the two outer-faces F_k, F'_k of the two copies together bound one face. Into this face, insert $9 \cdot 2^k$ vertices that we call U_{k+1}, grouped into $3 \cdot 2^k$ paths of three vertices each. Each vertex of U_{k+1} is adjacent to three vertices each of F_k and F'_k; Figure 6 shows that this can be done while remaining 1-planar.

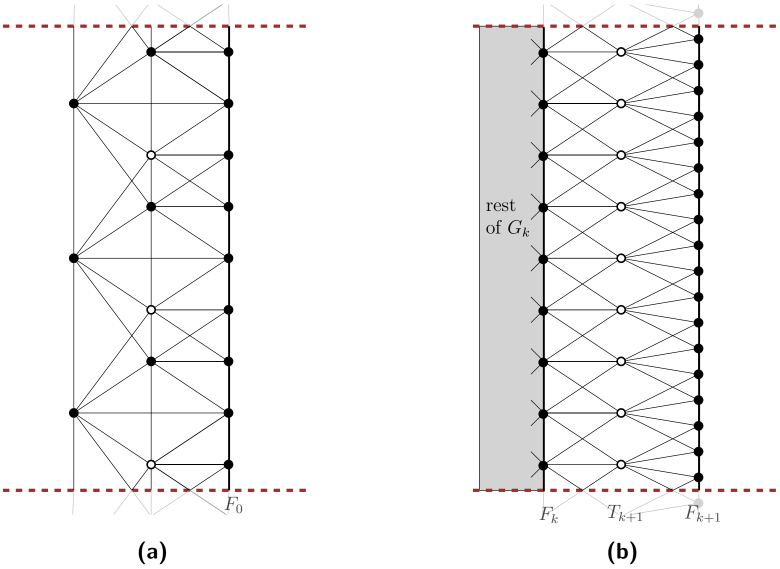


Figure 5 The base case and the induction step for building the graph G_k. For ease of reading we now show the construction on the rolling cylinder, rather than the standing one.

Since each vertex of U_{k+1} also has at least one neighbour in U_{k+1}, and each vertex of F_k and F'_k receives three more neighbours, the resulting graph G has minimum degree 7. Define T to consist of the two independent sets of the two copies of G_k as well as the $6 \cdot 2^k$ end-vertices of the paths in U_{k+1}; this is an independent set. See Figure 5.

It remains to analyze the size of G and T. Since G contains two copies of G_k, plus U_{k+1}, it has

$$n = 2 \cdot 27 \cdot 2^k - 2 \cdot 9 + 9 \cdot 2^k = 63 \cdot 2^k - 18 \geq N$$

vertices. Likewise T contains two copies of the independent set of G_k, plus the ends of the $3 \cdot 2^k$ paths, hence

$$|T| = 2 \cdot 9 \cdot 2^k - 2 \cdot 6 + 6 \cdot 2^k = 24 \cdot 2^k - 12.$$

Since $\frac{8}{21}(63 \cdot 2^k - 18 - 13.5) = 24 \cdot 2^k - \frac{144}{21} - \frac{108}{21} = 24 \cdot 2^k - 12$, the bound holds. ◀

With this we have proved all lower-bound entries in Table 1.

4 Upper bounds on the independence number

All our approaches to prove the upper bounds rely on bounding the maximum size of a bipartite 1-planar graph where one side of the bipartition is the bounded degree independent set. A previously known result here gives tight upper bounds for 1-planar graphs with minimum degree 3,4 and 5. For $\delta = 6$, by counting differently, we improve the bound. For $\delta = 7$ we improve the existing result by using a charging/discharging argument.

4.1 Upper bounds on the independence number for $\delta = 3, 4, 5$

To obtain our upper bounds in this section, we use the following lemma from [7] on independent sets in 1-planar graphs.

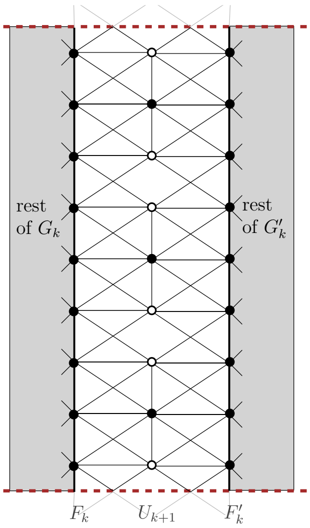


Figure 6 Combining two copies of G_k.

▶ **Lemma 5** ([7]). *Let G be a simple 1-planar graph. Let T be a non-empty independent set in G where $\deg(t) \geq 3$ for all $t \in T$. Let T_d be the vertices in T that have degree d. Then*

$$2|T_3| + \sum_{d\geq 4}(3d-6)|T_d| \leq 12|\overline{T}| - 24. \tag{1}$$

The notation of "minimum degree" is normally only defined for an entire graph, but we now use it also for a subset T of vertices, so T has minimum degree δ if all vertices in T have degree at least δ (but vertices in $\overline{T}$ may have smaller degrees).

Given the upper bound established in Lemma 5, we are able to use a counting argument to obtain the following upper bounds.

▶ **Corollary 6.** *Let G be a simple 1-planar graph and T be an independent set with minimum degree $\delta = 3$. Then $|T| \leq \frac{6}{7}(n-2)$.*

Proof. We have $2|T| = 2\sum_{d\geq 3}|T_d| \leq 2|T_3| + \sum_{d\geq 4}(3d-6)|T_d| \leq 12(n-|T|) - 24$ and therefore $14|T| \leq 12n - 24$. ◀

▶ **Corollary 7.** *Let G be a simple 1-planar graph and T be an independent set with minimum degree $\delta = 4$. Then $|T| \leq \frac{2}{3}(n-2)$.*

Proof. Since T_3 is empty, we have $6|T| = \sum_{d\geq 4} 6|T_d| \leq 2|T_3| + \sum_{d\geq 4}(3d-6)|T_d| \leq 12(n - |T|) - 24$ and therefore $18|T| \leq 12n - 24$. ◀

▶ **Corollary 8.** *Let G be a simple 1-planar graph and T be an independent set with minimum degree $\delta = 5$. Then $|T| \leq \frac{4}{7}(n-2)$.*

Proof. Since T_3 and T_4 are empty, we have $9|T| = \sum_{d\geq 5} 9|T_d| \leq 2|T_3| + \sum_{d\geq 4}(3d-6)|T_d| \leq 12(n-|T|) - 24$ and therefore $21|T| \leq 12n - 24$. ◀

4.2 Upper bounds on the independence number for $\delta = 6$

Note that if we apply the above Lemma for $\delta = 6$, we get a bound of $12|T| = \sum_{d\geq 6} 12|T_d| \leq 2|T_3| + \sum_{d\geq 4}(3d-6)|T_d| \leq 12(n-|T|) - 24$ and therefore $24|T| \leq 12n - 24$ which means $\frac{1}{2}(n-2)$. However, we are able to get a slightly better bound by using an alternative argument.

▶ **Lemma 9.** *Let G be a simple 1-planar graph. Then for any independent set T with minimum degree δ we have $|T| \leq \frac{3n-8-\chi}{\delta}$ where $\chi = 1$ if n is odd and $\chi = 0$ otherwise.*

Proof. Consider the 1-planar bipartite subgraph G^- of G that consists of the edges between T and $\overline{T}$. This graph has n vertices and has (by a result by Karpov [23]) at most $3n - 8 - \chi$ edges. Every vertex of T has no neighbour in T, so all its incident edges are in G^-. Therefore $\delta|T| \leq E(G^-) \leq 3n - 8 - \chi$ which implies the result. ◀

▶ **Corollary 10.** *Let G be a simple 1-planar graph and T be an independent set with minimum degree $\delta = 6$. Then $|T| \leq \frac{1}{2}(n-3)$.*

Proof. If n is odd then $|T| \leq \frac{1}{6}(3n-9) = \frac{1}{2}(n-3)$. If n is even then by integrality $|T| \leq \lfloor \frac{1}{6}(3n-8) \rfloor = \lfloor \frac{n}{2} - \frac{4}{3} \rfloor = \frac{n}{2} - 2 = \frac{1}{2}(n-4)$. ◀

4.3 Upper bounds on the independence number for $\delta = 7$

We notice that by using the counting argument as above, the upper bounds that can be obtained for $\delta = 7$ are

$$\tfrac{3n-8}{7}, \tfrac{4n-8}{9}$$

which are quite weak. We are able to obtain a better upper bound by revisiting the charging/discharging argument that was used in the proof of Lemma 5 (this was hinted at in [7], and many parts of the proof below are directly taken from there). We furthermore generalize the statement to graphs with parallel edges (in [7] simplicity of the graph was used only for $\delta = 3$). Specifically we assume that the graph has no loops and a *bigon-free* 1-planar drawing Γ, i.e., there is no cell whose boundary consists of two parallel uncrossed edges.

▶ **Lemma 11.** *Let G be a 1-planar graph with an independent set T that has minimum degree $\delta \geq 4$. Graph G may have parallel edges, but assume that it has no loops and a bigon-free 1-planar drawing Γ. Then*

$$|T| \leq \frac{4}{\delta + \lceil \frac{\delta}{3} \rceil}(n-2).$$

Proof. We use a charging scheme, where we assign some *charges* (units of weight) to edges in G (as well as to some edges that we add to G), redistribute these charges to the vertices in T, and then count the number of charges in two ways to obtain the bound.

As a first step, delete all edges within $\overline{T}$ so that G becomes bipartite. Also add any edge to Γ that connects T to $\overline{T}$ and that can be added without a crossing. We are allowed to add parallel edges, as long as they do not form a bigon. Both operations can only increase degrees of vertices in T, so it suffices to prove the bound in the resulting drawing Γ'.

As shown in [7], for any vertex $t \in T$ there cannot be three consecutive crossed edges in the circular ordering of edges at t. For if there were three such edges (say $(t, s_1), (t, s_2), (t, s_3)$) then the edge that crosses (t, s_2) has one endpoint in $\overline{T}$; we could have added an uncrossed edge from this endpoint to t, and since it would be before or after (t, s_2) in the circular ordering at t it would not have formed a bigon. This contradicts maximality.

We assign charges as follows: Let E_- be the uncrossed edges of Γ; each of those receives 2 charges. Let $E_\times$ be the crossed edges of Γ; each of those receives 1 charges. We know (see [7]) that $\frac{1}{2}|E_\times| + |E_-| \leq 2n - 4$ (this holds even with parallel edges if the drawing is bigon-free and has no loops). Hence

$$\#\text{charges} \quad = \quad 2|E_-| + 1|E_\times| \leq 4n - 8. \tag{2}$$

For $t \in T$, let $c(t)$ be the total charges of incident edges of t and write d for the degree of t. We know that there are at least $\lceil \frac{d}{3} \rceil$ uncrossed edges at t since there are no three consecutive crossed edges. Thus t obtains $2\lceil \frac{d}{3} \rceil$ charges from three uncrossed edges, and at least $d - \lceil \frac{d}{3} \rceil$ further charges from the remaining edges. Hence $c(t) \geq d + \lceil \frac{d}{3} \rceil \geq \delta + \lceil \frac{\delta}{3} \rceil$ and

$$\#\text{charges} \quad = \quad \sum_{t \in T} c(t) \geq |T|(\delta + \lceil \tfrac{\delta}{3} \rceil). \tag{3}$$

Combining this with (2) gives $|T|(\delta + \lceil \frac{\delta}{3} \rceil) \leq 4n - 8$ as desired. ◀

▶ **Corollary 12.** *Let G be a simple 1-planar graph and T be an independent set with minimum degree $\delta = 7$. Then $|T| \leq \frac{2}{5}(n-2)$.*

Proof. The proof follows from Lemma 11 by setting $\delta = 7$ ◀

With this we have proved all upper-bound entries in Table 1.

5 Optimal 1-planar graphs

Caro and Roditty in [13] showed that if G is a planar graph with order $n \geq 4$ and minimum degree δ, the equality $\alpha(G) = \frac{2n-4}{\delta}$ holds if and only if G can be formed from a planar graph H, all of whose faces are bounded by δ-cycles, by adding a vertex of degree δ inside each region. In particular, for planar graph the upper bound on the independence number is tight for *maximal planar graph*, i.e., planar graphs that have the maximum possible number $3n-6$ of edges.

In the same spirit, one should ask what the independence number can be for 1-planar graphs that have the maximum possible number of edges. It is known that every 1-planar graph has at most $4n-8$ edges, and a simple 1-planar graph G is called *optimal* if it has exactly $4n-8$ edges. An optimal 1-planar graph can equivalently be defined as the graphs obtained by taking a planar quadrangulated graph Q (i.e., all faces are bounded by 4-cycles) and inserting a pair of crossing edges into each face. Numerous results are known for such graphs, see [8]. In particular, a simple optimal 1-planar graph has exactly $n-2$ pairs of crossing edges, all vertex-degrees are even, and the minimum degree is 6.

▶ **Lemma 13.** *Let G be a simple optimal 1-planar graph. Then for any independent set T we have $|T| \leq \frac{1}{3}(n-2)$.*

Proof. Fix an arbitrary vertex $t \in T$, say it has degree $d \geq 6$. In the 1-planar drawing of G, the cyclic order of edges around t alternates between uncrossed and crossed edges. Therefore half of the incident edges of t are crossed, and we assign all these crossings to t. This does not double-count crossings, because (in an optimal 1-planar graph) the four endpoints of a crossing induce K_4 and so at most one of them can belong to T. We assigned at least three crossings to every vertex in T, and there are exactly has $n-2$ crossings, so $|T| \leq \frac{1}{3}(n-2)$. ◀

▶ **Lemma 14.** *For any integer N, there exists a simple optimal 1-planar graph with $n \geq N$ vertices and an independent set of size $\frac{1}{3}(n-2)$.*

Proof. Let H be a $2s$-*prism*, i.e., it consists of two cycles of length $2s$ with corresponding vertices of the cycles connected by an edge. Here $s \geq \max\{4, \frac{N-2}{6}\}$. Graph H has $4s$ vertices and is bipartite; we let T be one of its colors classes and note that $|T| = 2s$. See also Figure 7.

Now obtain graph G by adding the dual graph H^* to H and connecting every dual vertex v_F of H^* to all vertices of the face F of H that v_F represents. It is well-known that this gives an optimal 1-planar graph, and since H has $2s+2$ faces, we have $n = |V(G)| = 6s+2 \geq N$. No two vertices of T were connected, so T is an independent set of G of size $2s = \frac{1}{3}(n-2)$. ◀

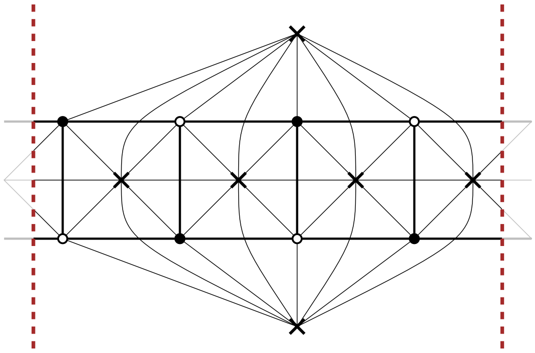


Figure 7 Graph H (bold) for $s = 2$ and the resulting optimal 1-planar graph that has an independent set (white) of size $\frac{n-2}{3}$.

Combining the two results we obtain:

▶ **Theorem 15.** *The independence number of optimal* 1*-planar graphs is exactly* $\frac{n-2}{3}$ *for the only feasible minimum degree* $\delta = 6$.

6 Further thoughts

In this paper, we studied upper bounds on the independence number of 1-planar graphs of minimum degree δ (for $\delta = 3, 4, 5, 6, 7$). This considered all interesting cases, because for $\delta = 2$ the complete bipartite graph $K_{2,n-2}$ is 1-planar (in fact, planar), so the independence number can be arbitrarily close to n, and there are no simple 1-planar graphs with minimum degree $\delta = 8$. We also provided 1-planar graphs for these minimum degrees that have large independent sets.

For $\delta = 3, 4, 5$, our lower and upper bound match exactly (see also Table 1). For $\delta = 6$, our bounds are within a very small constant of each other. We do leave a larger gap between upper and lower bounds for $\delta = 7$.

One reason for this gap is that the arguments used for upper bounds (Lemmas 5, 9, and 11) are ignoring some information: they do not use that the entire *graph* has minimum degree δ, but they only use that the independent set T has minimum degree δ. While this (surprisingly) does not seem to make a difference for $\delta = 3, 4, 5$, it makes a tiny (additive) difference for $\delta = 6$ and a noticeable (multiplicative) difference for $\delta = 7$. For $\delta = 6$, we can construct a graph with an independent set T of the (maximum possible size) $(n-3)/2$ if we allow just six vertices in $\overline{T}$ to have smaller degree; see also Figure 4. It is also not hard to construct a 1-planar graph with an independent set T of size $\frac{2}{5}n - O(1)$ that has minimum degree 7 (but some vertices of $\overline{T}$ have smaller degree); roughly speaking take two graphs G_k (from Lemma 4) and identify the vertices of the two copies of F_k (we leave the calculations to the reader). So Lemma 11 as written is tight, but can we prove a smaller upper bound in the scenario where vertices of $\overline{T}$ also must have minimum degree δ?

Last but not least, it would be interesting to explore algorithmic questions around finding independent sets of a certain size. For example, it is easy to find an independent set of size $\frac{n}{8}$ in any 1-planar graph (because they are 7-degenerate and so can be 8-coloured in linear time). With more effort, we can even 7-color the graph in linear time, so find an independent set of size at least $\frac{n}{7}$ [14]. But can we find, say, an independent set of size $\frac{n}{3} - O(1)$ in an optimal 1-planar graph efficiently?

References

1 Michael O. Albertson. A lower bound for the independence number of a planar graph. *Journal of Combinatorial Theory, Series B*, 20(1):84–93, 1976.

2 Vladimir E. Alekseev, Vadim V. Lozin, Dmitriy S. Malyshev, and Martin Milanic. The maximum independent set problem in planar graphs. In *Proc. MFCS*, volume 5162 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2008.

3 Kenneth Appel and Wolfgang Haken. *Every Planar Map is Four Colorable.* American Mathematical Society, Providence, RI, 1989.

4 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

5 Claude Berge. *Graphes et hypergraphes.* Dunod, Paris, 1970.

6 T. Biedl and D. Wilkinson. Bounded-degree independent sets in planar graphs. *Theory Comput. Syst.*, 38(3):253–278, 2005.

7 T. Biedl and J. Wittnebel. Matchings in 1-planar graphs with large minimum degree. *J. Graph Theory*, 99(2):217–320, 2022. doi:10.1002/jgt.22736.

8 R. Bodendiek, H. Schumacher, and K. Wagner. Über 1-optimale graphen. *Mathematische Nachrichten*, 117(1):323–339, 1984. doi:10.1002/mana.3211170125.

9 J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory.* Graduate Texts in Mathematics. Springer, 2008.

10 O. V. Borodin. Solution of the Ringel problem on vertex-face coloring of planar graphs and coloring of 1-planar graphs. *Metody Diskret. Analiz.*, 41:12–26, 108, 1984.

11 Prosenjit Bose, Vida Dujmović, and David R. Wood. Induced subgraphs of bounded degree and bounded treewidth. In *Graph-theoretic concepts in computer science*, volume 3787 of *Lecture Notes in Comput. Sci.*, pages 175–186. Springer, Berlin, 2005. doi:10.1007/11604686_16.

12 James E. Burns. The maximum independent set problem for cubic planar graphs. *Networks*, 19(3):373–378, 1989.

13 Y. Caro and Y. Roditty. On the vertex-independence number and star decomposition of graphs. *Ars Combin.*, 20:167–180, 1985.

14 Zhi-Zhong Chen and Mitsuharu Kouno. A linear-time algorithm for 7-coloring 1-plane graphs. *Algorithmica*, 43(3):147–177, 2005. doi:10.1007/S00453-004-1134-X.

15 Norishige Chiba, Takao Nishizeki, and Nobuji Saito. An algorithm for finding a large independent set in planar graphs. *Networks*, 13(2):247–252, 1983.

16 Marek Chrobak and Joseph Naor. An efficient parallel algorithm for computing a large independent set in planar graph. *Algorithmica*, 6(6):801–815, 1991.

17 Daniel W. Cranston and Landon Rabern. Planar graphs have independence ratio at least 3/13. *Electron. J. Comb.*, 23(3):3, 2016.

18 Norm Dadoun and David G. Kirkpatrick. Parallel algorithms for fractional and maximal independent sets in planar graphs. *Discret. Appl. Math.*, 27(1-2):69–83, 1990.

19 David P. Dobkin and David G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(3):381–392, 1985.

20 Zdenek Dvorák and Matthias Mnich. Large independent sets in triangle-free planar graphs. *SIAM J. Discret. Math.*, 31(2):1355–1373, 2017.

21 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

22 Xin He. A nearly optimal parallel algorithm for constructing maximal independent set in planar graphs. *Theor. Comput. Sci.*, 61:33–47, 1988.

23 D.V. Karpov. An upper bound on the number of edges in an almost planar bipartite graph. *Journal of Mathematical Sciences*, 196:737–746, 2014. doi:10.1007/s10958-014-1690-9.

24 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

25 Vadim V. Lozin and Martin Milanic. On the maximum independent set problem in subclasses of planar graphs. *J. Graph Algorithms Appl.*, 14(2):269–286, 2010.

26 C. E. Veni Madhavan. Approximation algorithm for maximum independent set in planar traingle-free [sic!] graphs. In *Proc. FSTTCS*, volume 181 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 1984.

27 Avner Magen and Mohammad Moharrami. Robust algorithms for on minor-free graphs based on the Sherali-Adams hierarchy. In *Proc. APPROX*, volume 5687 of *Lecture Notes in Computer Science*, pages 258–271. Springer, 2009.

28 Bojan Mohar. Face covers and the genus problem for apex graphs. *J. Comb. Theory, Ser. B*, 82(1):102–117, 2001.

29 Neil Robertson, Daniel P. Sanders, Paul D. Seymour, and Robin Thomas. The four-colour theorem. *J. Comb. Theory, Ser. B*, 70(1):2–44, 1997.

30 Marcus Schaefer. The graph crossing number and its variants: a survey. *Electron. J. Combin.*, DS21:90, 2013.

Size-Constrained Weighted Ancestors with Applications

Philip Bille
Technical University of Denmark, Lyngby, Denmark

Yakov Nekrich
Michigan Technological University, Houghton, MI, US

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Abstract

The *weighted ancestor* problem on a rooted node-weighted tree T is a generalization of the classic *predecessor* problem: construct a data structure for a set of integers that supports fast predecessor queries. Both problems are known to require $\Omega(\log\log n)$ time for queries provided $\mathcal{O}(n \operatorname{poly}\log n)$ space is available, where n is the input size. The weighted ancestor problem has attracted a lot of attention by the combinatorial pattern matching community due to its direct application to suffix trees. In this formulation of the problem, the nodes are weighted by string depth. This research has culminated in a data structure for weighted ancestors in suffix trees with $\mathcal{O}(1)$ query time and an $\mathcal{O}(n)$-time construction algorithm [Belazzougui et al., CPM 2021].

In this paper, we consider a different version of the weighted ancestor problem, where the nodes are weighted by any function weight that maps each node of T to a positive integer, such that $\mathsf{weight}(u) \leq \mathsf{size}(u)$ for any node u and $\mathsf{weight}(u_1) \leq \mathsf{weight}(u_2)$ if node u_1 is a descendant of node u_2, where $\mathsf{size}(u)$ is the number of nodes in the subtree rooted at u. In the *size-constrained weighted ancestor* (SWA) problem, for any node u of T and any integer k, we are asked to return the lowest ancestor w of u with weight at least k. We show that for *any rooted tree* with n nodes, we can locate node w in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing. In particular, this implies a data structure for the SWA problem in suffix trees with $\mathcal{O}(1)$ query time and $\mathcal{O}(n)$-time preprocessing, when the nodes are weighted by weight. We also show several string-processing applications of this result.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases weighted ancestors, string indexing, data structures

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.14

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2311.15777`

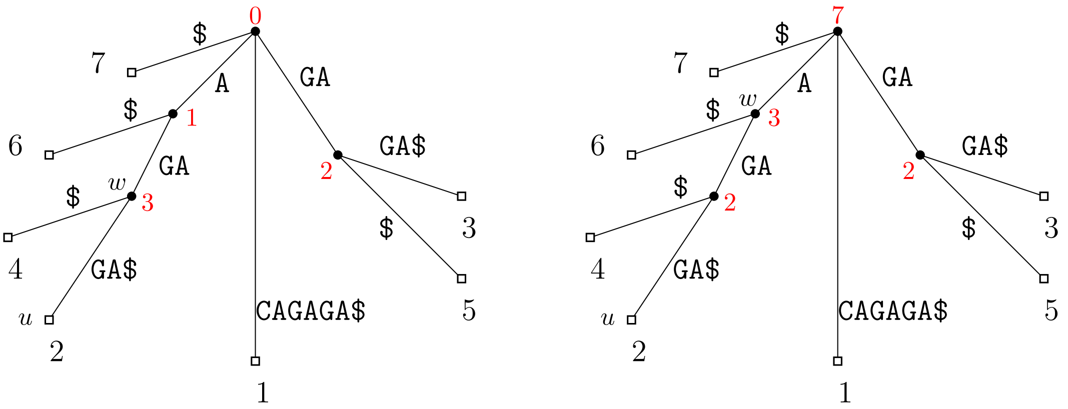
Funding *Philip Bille*: Supported by the Independent Research Fund Denmark (DFF-9131-00069B).
Yakov Nekrich: Supported by the National Science Foundation under NSF grant 2203278.
Solon P. Pissis: Supported by the PANGAIA (No 872539) and ALPACA (No 956229) projects.

1 Introduction

In the classic *predecessor* problem [27, 16, 29, 24, 23], we are given a set S of keys from a universe U with a total order. The goal is to preprocess set S into a compact data structure supporting the following on-line queries: for any element $q \in U$, return the maximum $p \in S$ such that $p \leq q$; p is called the *predecessor* of q.

The *weighted ancestor* problem, introduced by Farach and Muthukrishnan in [15], is a natural generalization of the predecessor problem on rooted node-weighted trees. In particular, given a rooted tree T, whose nodes are weighted by positive integers and such that these weights decrease when ascending from any node to the root, the goal is to preprocess

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 14; pp. 14:1–14:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) The internal nodes are weighted by *string depth* (in red). Asking a weighted ancestor query for $i = 2$ (node u) and $k = 2$ will take us to node w. Indeed, (w, k) is the locus of substring AG in the suffix tree of X.

(b) The internal nodes are weighted by *frequency* (in red). Asking a weighted ancestor query for $i = 2$, $j = 7$ (node u) and $k = 3$ will take us to node w. Indeed, A is the longest prefix of AGAGA$ that occurs at least 3 times in X.

Figure 1 Weighted ancestor queries on the suffix tree of string $X =$ CAGAGA$. The leaf nodes in both trees are labeled by the starting position of the suffix of X they represent.

tree T into a compact data structure supporting the following on-line queries: for any given node u and any integer $k > 0$, return the farthest ancestor of u whose weight is at least k. Both the predecessor and the weighted ancestor problems require $\Omega(\log \log n)$ time for queries provided $\mathcal{O}(n \operatorname{poly} \log n)$ space is available, where n is the input size of the problem [17].

The weighted ancestor problem has attracted a lot of attention in the combinatorial pattern matching community [15, 4, 22, 21, 17, 8, 6] due to its direct application to suffix trees [28]. The *suffix tree* of a string X is the compacted trie of the set of suffixes of X; see Figure 1a. In this formulation of the problem, a node u is weighted by *string depth*: the length of the string spelled from the root of the suffix tree to u; and a weighted ancestor query for two integers i and $k > 0$ returns the locus of substring $X[i \mathinner{.\,.} i + k - 1]$ in the suffix tree of X. We refer the reader to [17] for several applications. This research has culminated in a data structure for weighted ancestors in suffix trees, given by Belazzougui, Kosolobov, Puglisi, and Raman [8], supporting $\mathcal{O}(1)$-time queries after an $\mathcal{O}(n)$-time preprocessing.

However there are other tree weighting schemes that are of interest to string processing. For example, each suffix tree node can be weighted by the number of its leaf descendants; see Figure 1b. Thus the weight of a node u is equal to the frequency of the substring represented by the root-to-u path. If we use this weighting function, then the following basic string problem can be translated into a weighted ancestor query: *Given a substring $I = X[i \mathinner{.\,.} j]$ of string X and an integer $k > 0$, find the longest prefix of I that occurs at least k times in X.*

Unfortunately, the existing data structures for the weighted ancestor problem *on suffix trees* [17, 8] depend strongly on the fact that the suffix tree nodes are weighted by string depth. They thus *cannot be applied* to solve the aforementioned basic string problem.

Motivated by this fact, we introduce a different version of the weighted ancestor problem on *general rooted trees*. Let T be a rooted tree on a set V of n nodes. By $\mathsf{size}(u)$, we denote the number of nodes in the subtree rooted at a node $u \in V$. Let $\mathsf{weight} : V \to \mathbb{N}$ denote any function that maps each node of T to a positive integer, such that $\mathsf{weight}(u) \leq \mathsf{size}(u)$ for any node $u \in V$ and $\mathsf{weight}(u_1) \leq \mathsf{weight}(u_2)$ if node $u_1 \in V$ is a descendant of node $u_2 \in V$. The latter is also known as the *max-heap property*: the weight of each node is less than or equal to the weight of its parent, with the maximum-weight element at the root. We will say that a function $\mathsf{weight} : V \to \mathbb{N}$ satisfying both properties is a *size-constrained max-heap* weight function. For any node $u \in V$ and any integer $k > 0$, a *size-constrained weighted ancestor query*, denoted by $\mathsf{SWA}(u, k) = w$, asks for the lowest ancestor $w \in V$ of u with weight at least k. The *size-constrained weighted ancestor* (SWA) problem, formalized next, is to preprocess T into a compact data structure supporting fast SWA queries:

SIZE-CONSTRAINED WEIGHTED ANCESTOR (SWA)

Preprocess: A rooted tree T on a set V of n nodes weighted by a size-constrained max-heap function $\mathsf{weight} : V \to \mathbb{N}$.

Query: Given a node $u \in V$ and an integer $k > 0$, return the lowest ancestor w of u with $\mathsf{weight}(w) \geq k$.

We assume throughout the standard word RAM model of computation with word size $\Theta(\log n)$; basic arithmetic and bit-wise operations on $\mathcal{O}(\log n)$-bit integers take $\mathcal{O}(1)$ time. Note that, since function weight must satisfy the max-heap property, one can employ the existing data structures for the weighted ancestor problem *on general rooted trees* [15, 4], to answer SWA queries in $\mathcal{O}(\log \log n)$ time after $\mathcal{O}(n)$-time preprocessing (see also [25]). Our main result in this paper can be formalized as follows (see Section 3 and Section 4).

▶ **Theorem 1.** *For any rooted tree with n nodes weighted by a size-constrained max-heap function* weight*, there exists an $\mathcal{O}(n)$-space data structure answering* SWA *queries in $\mathcal{O}(1)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

As a preliminary step, we design an $\mathcal{O}(n \log n)$-space solution using an involved combination of *rank-select* data structures [7], *fusion trees* [16], and *heavy-path decompositions* [26]. We then design a novel application of *ART decomposition* [2] to arrive to Theorem 1.

Applications. Notably, Theorem 1 presents a data structure for the SWA problem in suffix trees with $\mathcal{O}(1)$ query time and $\mathcal{O}(n)$-time preprocessing, when the nodes are weighted by a size-constrained max-heap weight function weight. We show several string-processing applications of this result since $\mathsf{weight}(u)$ can be defined as the number of leaf nodes in the subtree rooted at u. Let us first provide some intuition on the applicability of Theorem 1.

Consider a relatively long query submitted to a search-engine text database. If the database returns no (or not sufficiently many) results, one usually tries to *repeatedly* truncate some prefix and/or some suffix of the original query until they obtain sufficiently many results. Our Theorem 1 can be applied to solve this problem directly in optimal time.

In particular, Theorem 1 yields *optimal data structures*, with respect to preprocessing and query times, for the following basic string-processing problems (see Section 5):

1. Preprocess a string X into a linear-space data structure supporting the following on-line queries: for any i, j, f return the longest prefix of $X[i\,..\,j]$ occurring at least f times in X.
2. Preprocess a dictionary $\mathcal{D}$ of documents into a linear-space data structure supporting the following on-line queries: for any string P and any integer f, return a longest substring of P occurring in at least f documents of $\mathcal{D}$.
3. Preprocess a string X into a linear-space data structure supporting the following on-line queries: for any string P and any integer f, return a longest substring of P occurring at least f times in X.

Theorem 1 also directly improves on the data structure presented by Pissis et al. [25] for computing the *frequency-constrained substring complexity* of a given string (see Section 5).

2 Preliminaries

For any bit string B of length m and any $\alpha \in \{0,1\}$, the classic rank and select queries are defined as follows:

- rank_α: for any given $i \in [1,m]$, it returns the number of ones (or zeros) in $B[1\,..\,i]$; more formally, $\mathsf{rank}_\alpha(B,i) = |\{j \in [1,i] : B[j] = \alpha\}|$.
- select_α: for any given rank i, it returns the leftmost position where the bit vector contains a one (or zero) with rank i; more formally, $\mathsf{select}_\alpha(B,i) = \min\{j \in [1,m] : \mathsf{rank}_\alpha(B,j) = i\}$.

The following result is known.

▶ **Lemma 2** (Rank and Select [7]). *Let B be a bit string of length $m \leq n$ stored in $\mathcal{O}(1 + m/\log n)$ words. We can preprocess B in $\mathcal{O}(1 + m/\log n)$ time into a data structure of $m + o(m)$ bits supporting* rank *and* select *queries in $\mathcal{O}(1)$ time.*

Bit strings can also be used as a representation of monotonic integer sequences supporting predecessor queries; see [5], for example. Assume we have a set S of m keys from a universe U with a total order. In the *predecessor* problem, we are given a query element $q \in U$, and we are to find the maximum $p \in S$ such that $p \leq q$; we denote this query by $\mathsf{predecessor}(q) = p$. The following result is known for a special case of the predecessor problem.

▶ **Lemma 3** (Fusion Tree [16]). *We can preprocess a set of $m = \log^{\mathcal{O}(1)} n$ integers in $\mathcal{O}(m)$ time and space to support* $\mathsf{predecessor}$ *queries in $\mathcal{O}(1)$ time.*

3 Constant-time Queries using $\mathcal{O}(n \log n)$ Space

We first show how to solve the SWA problem in $\mathcal{O}(1)$ time using $\mathcal{O}(n \log n)$ space. This solution forms the basis for our linear-time and linear-space solution in Section 4.

3.1 Heavy-path Decomposition

Let T be a rooted tree with n nodes. We compute the *heavy-path decomposition* of T in $\mathcal{O}(n)$ time [26]. Recall that, for any node u in T, we define $\mathsf{size}(u)$ to be number of nodes in the subtree of T rooted at u. We call an edge (u,v) of T *heavy* if $\mathsf{size}(v)$ is maximal among every edge originating from u (breaking ties arbitrarily). All other edges are called *light*. We call a node that is reached from its parent through a heavy edge *heavy*; otherwise, the node is called *light*. The heavy path of T is the path that starts at the root of T and at each node on the path descends to the heavy child as defined above. The *heavy-path decomposition* of T is then defined recursively: it is a union of the heavy path of T and the heavy-path decompositions of the off-path subtrees of the heavy path. A well-known property of this decomposition is that every root-to-node path in T passes through at most $\log n$ light edges. In particular, the following lemma is implied.

▶ **Lemma 4** (Heavy-path Decomposition [26]). *Let T be a rooted tree with n nodes. Any root-to-leaf path in T consists of at most $\log n + \mathcal{O}(1)$ heavy paths.*

3.2 Data Structure

We construct a heavy-path decomposition of T. Consider a heavy path $H = v_1 \dots v_\ell$. We construct a bit string $B(H)$ that represents the differences between node weights using unary coding. Suppose that nodes $v_1 \dots v_\ell$ of H are listed in decreasing order of their depth and let $\delta(v_i) = \mathsf{weight}(v_i) - \mathsf{weight}(v_{i-1})$, for all $i > 1$. We define $B(H)$ as follows:

$$B(H) = \mathsf{enc}(\mathsf{weight}(v_1)) \cdot \mathsf{enc}(\delta(v_2)) \cdot \dots \cdot \mathsf{enc}(\delta(v_i)) \cdot \dots \cdot \mathsf{enc}(\delta(v_\ell)),$$

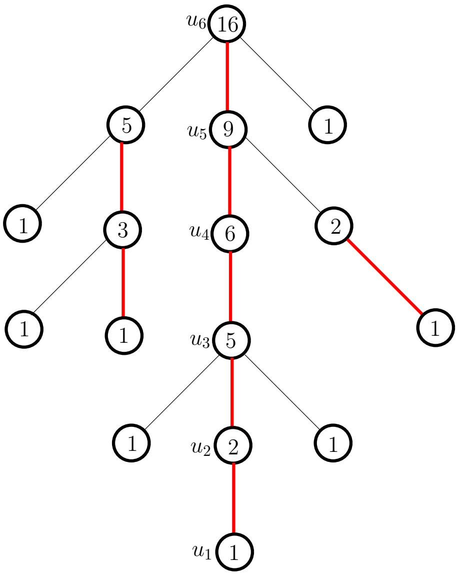
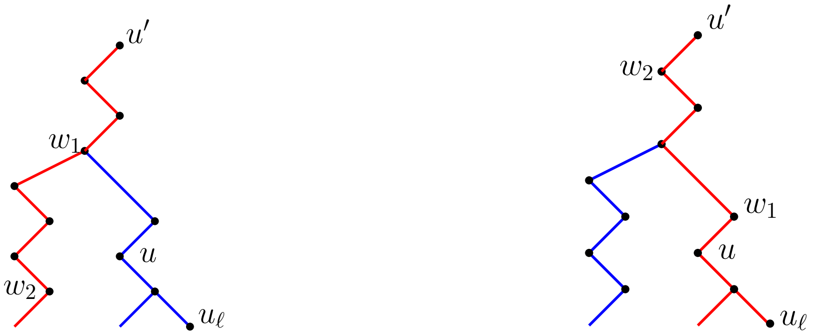


Figure 2 A rooted tree T with $n = 16$ nodes. Each node u of T is weighted by $\mathsf{weight}(u) = \mathsf{size}(u)$. For example, $\mathsf{weight}(u_5) = \mathsf{size}(u_5) = 9$, because there are 9 nodes in the subtree rooted at u_5, and $\mathsf{SWA}(u_2, 7) = u_5$ because the lowest ancestor of u_2 with weight at least 7 is node u_5. A heavy-path decomposition of T is also depicted: the heavy edges are the red edges. For example, the heavy path of the whole T is $u_1u_2 \ldots u_6$.

where $\mathsf{enc}(i)$ denotes the unary code of i; i.e., $\mathsf{enc}(i)$ consists of i 1's followed by a single 0. The important property of our encoding is that the total number of 0-bits in $B(H)$ is ℓ and the total number of 1-bits is $\mathsf{weight}(v_\ell)$.

▶ **Example 5.** Let $H = u_1u_2 \ldots u_6$ be the heavy path of T from Figure 2. We have $\ell = 6$ and $\mathsf{weight}(u_1) = 1$, $\mathsf{weight}(u_2) = 2$, $\mathsf{weight}(u_3) = 5$, $\mathsf{weight}(u_4) = 6$, $\mathsf{weight}(u_5) = 9$, $\mathsf{weight}(u_6) = 16$. We have $B(H) = \mathtt{1010111010111011111110}$. For instance, the second 1 denotes $\delta(u_2) = \mathsf{weight}(u_2) - \mathsf{weight}(u_1) = 1$. The leftmost occurrence of 111 denotes $\delta(u_3) = \mathsf{weight}(u_3) - \mathsf{weight}(u_2) = 3$ 1's.

For any heavy path H, we can construct $B(H)$ in $\mathcal{O}(\ell)$ time using standard word RAM bit manipulations to construct the unary codes and concatenate the underlying bit strings. By Lemma 4, every leaf node of T has $\mathcal{O}(\log n)$ ancestors v_t, such that v_t is the topmost node of some heavy path H. Since any node in T is counted in the weight of $\mathcal{O}(\log n)$ topmost nodes, the total weight of all topmost nodes, summed over all heavy paths H, is $\mathcal{O}(n \log n)$. Thus, the total length of all bit strings $B(H)$ is $\mathcal{O}(n \log n)$ and we can construct them all in $\mathcal{O}(n)$ time since the total length of the heavy paths is $\mathcal{O}(n)$. We store each such bit string according to Lemma 2 to support $\mathcal{O}(1)$-time rank and select queries using $\mathcal{O}(n)$ preprocessing time and words of space. Furthermore, for each leaf node v in T we store the weights of the top nodes of each heavy path on the path from the root to v. By Lemma 4, there are $\mathcal{O}(\log n)$ such top nodes for each leaf. For every leaf node we store the weights of its top node ancestors in a *fusion tree* data structure according to Lemma 3. The total space used by all such fusion trees is $\mathcal{O}(n \log n)$ words and the preprocessing time is $\mathcal{O}(n \log n)$. Finally, we construct a *lowest common ancestor* (LCA) data structure over T. Such a data structure answers LCA queries in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space preprocessing [9].



(a) Case 1: Only w_1 is an ancestor of u. The heavy path H_w is shown in red. The $(f+1)$th node w_2 on H_w is below w_1. The node w_1 is the $(f+g)$th node on H_w for some $g > 1$, and so w_1 is the answer.

(b) Case 2: Both w_1 and w_2 are ancestors of u. The heavy path H_w is shown in red. The $(f+1)$th node w_2 on H_w is above w_1, and so w_2 is the answer.

Figure 3 The two cases of the querying algorithm.

3.3 Queries

Suppose we are given a node u and an integer k as an $\mathsf{SWA}(u,k)$ query. We are looking for the lowest ancestor w of u with weight at least k. If the weight of u is at least k, we return u. Otherwise we proceed as follows. First, we locate the heavy path H_w that contains node w: we find an arbitrary leaf descendant u_ℓ of u; then, using the fusion tree of u_ℓ, we find the lowest ancestor u' of u_ℓ with weight at least k, such that u' is a top node. H_w is the heavy path, such that u' is its top node. When we find H_w, we answer a query $f = \mathsf{rank}_0(B(H_w), j)$ for $j = \mathsf{select}_1(B(H_w), k)$ using Lemma 2 in $\mathcal{O}(1)$ time. Let w_1 denote the lowest ancestor of u on the heavy path H_w (see Figure 3). If u is on H_w (Figure 3b), then w_1 is simply the parent of u. Otherwise (Figure 3a), w_1 can be found as the lowest common ancestor of the lowest node on H_w and node u. In the latter case, w_1 can be found using an LCA query that takes $\mathcal{O}(1)$ time. Let w_2 denote the $(f+1)$th node on H_w. The node w is the highest node among w_1 and w_2. The query time is $\mathcal{O}(1)$ by Lemma 3 for finding H_w and by Lemma 2 for finding f. Example 6 shows how we use $B(H_w)$ to find f and thus the $(f+1)$th node.

▶ **Example 6.** Let $B(H) =$ 1010111010111011111110 from Example 5, u_2 from Figure 2, and $k = 7$. Then $j = \mathsf{select}_1(B(H), 7) = 11$ and $f = \mathsf{rank}_0(B(H), 11) = 4$. The output node is u_5, the $(f+1)$th node on H. Indeed, $\mathsf{weight}(u_5) = 9 \geq k = 7$ and $\mathsf{weight}(u_4) = 6 < k = 7$.

In summary, we have shown the following result, which we will improve in the next section.

▶ **Lemma 7.** *For any rooted tree with n nodes weighted by a size-constrained max-heap function* weight*, there exists an $\mathcal{O}(n \log n)$-space data structure answering* SWA *queries in $\mathcal{O}(1)$ time. The preprocessing algorithm runs in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log n)$ space.*

4 Constant-time Queries using $\mathcal{O}(n)$ Space

We now improve the above solution to the SWA problem (Lemma 7) to linear-time and linear-space preprocessing. We will reuse the previous section's linear-time heavy-path decomposition and the corresponding bit string encoding. The key challenge is identifying the top nodes of heavy paths in $\mathcal{O}(1)$ time using linear space.

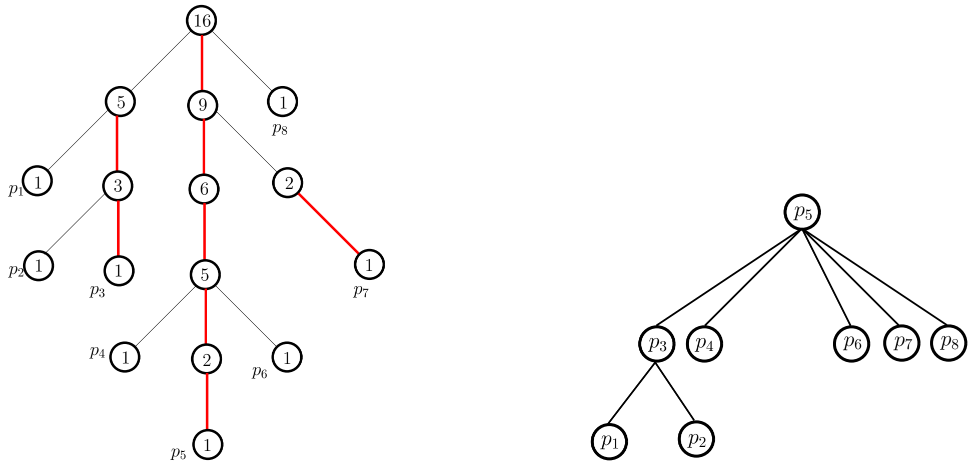
4.1 ART Decomposition

The ART decomposition, proposed by Alstrup, Husfeldt, and Rauhe [2], partitions a rooted tree into a *top tree* and several *bottom trees* with respect to an input parameter χ. Each node v of minimal depth, with no more than χ leaf nodes below it, is the root of a bottom tree consisting of v and all its descendants. The top tree consists of all nodes that are not in any bottom tree. The ART decomposition satisfies the following important property:

▶ **Lemma 8** (ART Decomposition [2]). *Let T be a rooted tree with ℓ leaf nodes. Further let χ be a positive integer. The ART decomposition of T with parameter χ produces a top tree with at most $\mathcal{O}(\ell/\chi)$ leaves. Such a decomposition of T can be computed in linear time.*

4.2 Data Structure

Recall that T consists of n nodes. As discussed in Section 3.2, we compute the heavy-path decomposition of T, construct bit strings for each heavy path, and preprocess the bit strings to support rank and select queries in $\mathcal{O}(1)$ time. This takes $\mathcal{O}(n)$ preprocessing time and space, allowing us to answer queries on a heavy path in $\mathcal{O}(1)$ time. Thus what remains is a linear-space and $\mathcal{O}(1)$-time solution to locate the top nodes of heavy paths.



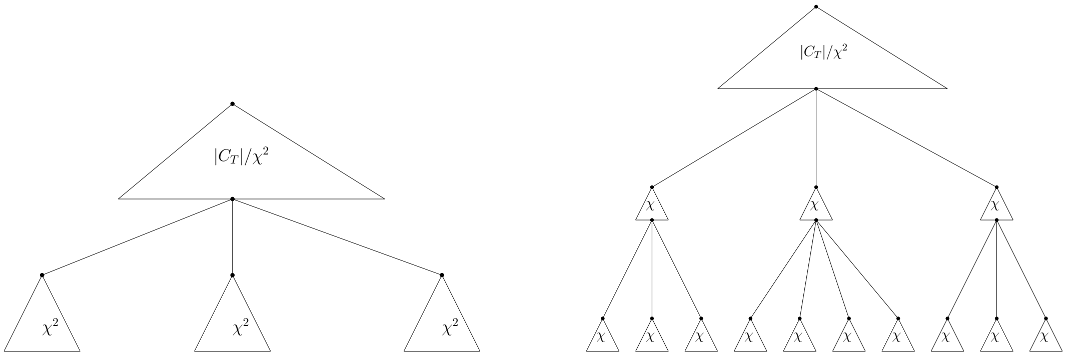
(a) The tree T from Figure 2. We write the heavy path id p_i at the end of the ith heavy path.

(b) The contracted tree C_T.

Figure 4 The contraction process of the tree T from Figure 2.

First, we construct the *contracted tree* C_T of T obtained by contracting all edges of heavy paths in T. In particular, this leaves all the light edges from T in C_T and removes all the heavy edges from T (see Figure 4). We then apply the ART decomposition on C_T (see Figure 5a) with parameter χ^2, where $\chi = \epsilon\frac{\log n}{\log\log n}$ and ϵ is a positive constant. We apply the ART decomposition again with parameter χ (see Figure 5b) on each resulting bottom tree. The resulting partition of C_T contains three levels of trees that we call the *top tree*, the *middle trees*, and the *bottom trees*. Since the heavy-path decomposition of T can be computed in $\mathcal{O}(n)$ time, contracting T takes $\mathcal{O}(n)$ time by processing the heavy-path decomposition of T. By Lemma 8, the ART decompositions of T cost $\mathcal{O}(n)$ total time.

Let us first consider the top tree. As in Section 3.2, we store a fusion tree for each leaf node in the top tree. By Lemma 8, the top tree has $\mathcal{O}(\frac{|C_T|}{\chi^2})$ leaves and hence, by Lemmas 3 and 4, this uses $\mathcal{O}(\frac{|C_T|}{\chi^2}\cdot\log n) = \mathcal{O}(\frac{n(\log\log n)^2}{\log n}) = o(n)$ space and preprocessing time.



(a) First application of ART decomposition on C_T.

(b) Application of ART decomposition on the bottom trees of the tree in Figure 5a.

Figure 5 Application of ART decompositions on C_T.

For the middle or bottom trees, we tabulate the answers to all possible queries in a global table. The index in the table is given by a tree encoding and the node u along with integer k for the SWA query. The corresponding value in the table is the output node of the $\mathsf{SWA}(u, k)$ query. We encode the input to a query as follows. We represent each middle and bottom tree compactly as a bit string encoding the tree structure and the weights of all nodes. Since each internal node in C_T is branching, the number of nodes in a middle or bottom tree is bounded by $\mathcal{O}(\chi)$. Thus, we can encode the tree structure using $\mathcal{O}(\chi)$ bits. The weight of a node in a middle or bottom tree is bounded by $\mathcal{O}(\chi^2)$ or $\mathcal{O}(\chi)$, respectively, and can thus be encoded in $\mathcal{O}(\log \chi)$ bits. Hence, we can encode the tree structure and all weights using $\mathcal{O}(\chi \log \chi)$ bits. We encode the query node u using $\mathcal{O}(\log \chi)$ bits. Since the maximum weight is $\mathcal{O}(\chi^2)$ we can also encode the query integer k using $\mathcal{O}(\log \chi)$ bits. Hence, the full encoding uses $\mathcal{O}(\chi \log \chi) + \mathcal{O}(\log \chi) + \mathcal{O}(\log \chi) = \mathcal{O}(\chi \log \chi)$ bits. To encode the output node stored in the global table we use $\mathcal{O}(\log \chi)$ bits. Thus, the table uses $2^{\mathcal{O}(\chi \log \chi)} \log \chi = 2^{\mathcal{O}(\epsilon \log n)} = o(n)$ bits for a sufficiently small constant $\epsilon > 0$. The table can be constructed in $o(n)$ time.

4.3 Queries

Suppose we are given a node u and an integer k as an $\mathsf{SWA}(u, k)$ query. Let u_t denote the top node on the heavy path of u in T and let u_H denote the corresponding node in the contracted tree C_T. We find the lowest ancestor w_H of u_H with weight at least k in C_T. If u_H is in the top tree we find w_H as described in Section 3.2. If u_H is in a middle or bottom tree, we use the global table to find w_H. If the result is not in the middle or bottom tree (the weight of the top node in such a tree is smaller than k), we move up a level and query the middle or top tree, respectively. Each of these at most three queries takes $\mathcal{O}(1)$ time. Thus w_H is found in $\mathcal{O}(1)$ time. Suppose that w_H corresponds to a node w' in the initial tree and let H' denote the heavy path such that w' is its top node. As explained in Section 3.2, we can find the lowest ancestor of u with weight at least k on H' in $\mathcal{O}(1)$ time using rank and select queries on $B(H')$. In total the $\mathsf{SWA}(u, k)$ query takes $\mathcal{O}(1)$ time.

In summary, we have obtained the following result.

▶ **Theorem 1.** *For any rooted tree with n nodes weighted by a size-constrained max-heap function* weight*, there exists an $\mathcal{O}(n)$-space data structure answering* SWA *queries in $\mathcal{O}(1)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

5 String-processing Applications

In this section, we show several applications of Theorem 1 on suffix trees. Recall that the number of leaf nodes in the subtree rooted at node u in a suffix tree is the number of occurrences (i.e., the frequency) of the substring represented by the root-to-u path.

5.1 Internal Longest Frequent Prefix

Internal pattern matching is an active topic [20, 3, 12, 11, 13, 1, 5] in the combinatorial pattern matching community. We introduce the following basic string problem. The *internal longest frequent prefix* problem asks to preprocess a string X of length n over an integer alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$ into a compact data structure supporting the following on-line queries:

- $\mathsf{ILFP}_X(i, j, f)$: return the longest prefix of $X[i\mathinner{.\,.}j]$ occurring at least f times in X.

Our solution to this problem will form the basic tool for solving the problems in Sections 5.2 and 5.3. We first construct the suffix tree T of X in $\mathcal{O}(n)$ time [14], and preprocess it in $\mathcal{O}(n)$ time for classic weighted ancestor queries [8] as well as for SWA queries using Theorem 1. For SWA queries, as $\mathsf{weight}(u)$, we use the number of leaf nodes in the subtree rooted at node u in T. Such an assignment satisfies the requested properties of $\mathsf{weight}(\cdot)$ and can be done in linear time using a standard DFS traversal on T. Any $\mathsf{ILFP}_X(i, j, f)$ query can be answered by first finding the locus $(u, j-i+1)$ of $X[i\mathinner{.\,.}j]$ in T in $\mathcal{O}(1)$ time using a classic weighted ancestor query on T, and, then, answering $\mathsf{SWA}(u, f)$ in T in $\mathcal{O}(1)$ time using Theorem 1. We obtain the following result.

▶ **Theorem 9.** *For any string X of length n over alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$, there exists an $\mathcal{O}(n)$-space data structure that answers ILFP_X queries in $\mathcal{O}(1)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

5.2 Longest Frequent Substring

The *longest frequent substring* problem is the following: preprocess a dictionary $\mathcal{D}$ of d strings (documents) of total length n over an integer alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$ into a compact data structure supporting the following on-line queries:

- $\mathsf{LFS}_{\mathcal{D}}(P, f)$: return a longest substring of P that occurs in at least f documents of $\mathcal{D}$.

This longest substring of P represents a most *relevant* part of the query with respect to $\mathcal{D}$. The length of $\mathsf{LFS}_{\mathcal{D}}(P, f)$ can also be used as a *measure of similarity* between P and the strings in $\mathcal{D}$, for some f chosen appropriately based on the underlying application.

We start by constructing the generalized suffix tree T of $\mathcal{D}$ in $\mathcal{O}(n)$ time [14] and preprocess it in $\mathcal{O}(n)$ time for SWA queries using Theorem 1. For SWA queries, $\mathsf{weight}(u)$ is equal to the number of dictionary strings having at least one leaf node in the subtree rooted at node u in T. This assignment satisfies the requested properties of $\mathsf{weight}(\cdot)$ and can be done in linear time [19]. Let us denote by (v_i, ℓ_i) the locus in T of the longest prefix of $P[i\mathinner{.\,.}|P|]$ that occurs in any string in $\mathcal{D}$. In fact, we can compute (v_i, ℓ_i), for all $i \in [1, |P|]$, in $\mathcal{O}(|P|)$ time using the *matching statistics* algorithm of P over T [10, 18]. For each locus (v_i, ℓ_i), we trigger a $\mathsf{SWA}(v_i, f)$ query using Theorem 1 (this is essentially an instance of the *internal longest frequent prefix* problem). In total this takes $\mathcal{O}(|P|)$ time. We obtain the following result.

▶ **Theorem 10.** *For any dictionary $\mathcal{D}$ of total length n over alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$, there exists an $\mathcal{O}(n)$-space data structure that answers $\mathsf{LFS}_{\mathcal{D}}(P, f)$ queries in $\mathcal{O}(|P|)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

An analogous result can be achieved for the following version of the longest frequent substring problem: preprocess a string X of length n over an integer alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$ into a compact data structure supporting the following on-line queries:

- $\mathsf{LFS}_X(P, f)$: return a longest substring of P that occurs at least f times in X.

In particular, instead of a generalized suffix tree, we now construct the suffix tree T of X and follow the same querying algorithm as above. For SWA queries, $\mathsf{weight}(u)$ is equal to the number of leaf nodes in the subtree rooted at node u in T. Such an assignment satisfies the requested properties of $\mathsf{weight}(\cdot)$ and can be done in linear time using a standard DFS traversal on T. We obtain the following result.

▶ **Theorem 11.** *For any string X of length n over alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$, there exists an $\mathcal{O}(n)$-space data structure that answers $\mathsf{LFS}_X(P, f)$ queries in $\mathcal{O}(|P|)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

5.3 Frequency-constrained Substring Complexity

For a string X, a dictionary $\mathcal{D}$ of d strings (documents) and a partition of $[d]$ in τ intervals $\mathcal{I} = I_1, \ldots, I_\tau$, the function $f_{X,\mathcal{D},\mathcal{I}}(i, j)$ maps i, j to the number of distinct substrings of length i of X occurring in at least α_j and at most β_j documents in $\mathcal{D}$, where $I_j = [\alpha_j, \beta_j]$. Function f is known as the *frequency-constrained substring complexity* of X [25].

▶ **Example 12.** Let $\mathcal{D} = \{\texttt{a}, \texttt{ananan}, \texttt{baba}, \texttt{ban}, \texttt{banna}, \texttt{nana}\}$. For $X = \texttt{banana}$ and $I_1 = [1, 2], I_2 = [3, 4], I_3 = [5, 6]$, we have $f_{X,\mathcal{D},\mathcal{I}}(2, 2) = 3$: `ba` occurs in $3 \in I_2$ documents; `an` occurs in $4 \in I_2$ documents; and `na` occurs in $3 \in I_2$ documents.

The function $f_{X,\mathcal{D},\mathcal{I}}$ is very informative about X; it provides fine-grained information about the contents (the substrings) of X. It can thus facilitate the tuning of string-processing algorithms by setting bounds on the length or on frequency of substrings; see [25].

Let S be a 2D array such that $S[i, j] = f_{X,\mathcal{D},\mathcal{I}}(i, j)$. Pissis et al. [25] showed that after an $\mathcal{O}(n)$-time preprocessing of a dictionary $\mathcal{D}$ of d strings of total length n over an integer alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$, for any X and any partition $\mathcal{I}$ of $[d]$ in τ intervals given on-line, S can be computed in near-optimal $\mathcal{O}(|X|\tau \log\log d)$ time.

The solution in [25] can be summarized as follows. In the preprocessing step, we construct the generalized suffix tree T of $\mathcal{D}$. In querying, the first step is to construct the suffix tree of X and compute the document frequency of its nodes in $\mathcal{O}(|X|)$ time. In the second step, we enhance the suffix tree of X with $\mathcal{O}(|X|\tau)$ nodes with document frequencies by answering SWA queries on T in $\mathcal{O}(\log\log d)$ time per query [4]. The whole step thus takes $\mathcal{O}(|X|\tau \log\log d)$ time. In the third step, we infer a collection of length intervals, one per node of the enhanced suffix tree and sort them in $\mathcal{O}(|X|\tau)$ time using radix sort. In the last step, we sweep through the intervals from left to right to compute array S in $\mathcal{O}(|X|\tau)$ total time. This concludes the summary of the solution in [25]. We amend the solution as follows.

We plug in Theorem 1 for preprocessing T and for the second step (SWA queries). For SWA queries, as $\mathsf{weight}(u)$, we use the number of dictionary strings having at least one leaf node in the subtree rooted at node u in T. Such an assignment satisfies the requested properties of $\mathsf{weight}(\cdot)$ and can be done in linear time [19]. We obtain the following result.

▶ **Theorem 13.** *For any dictionary $\mathcal{D}$ of d strings of total length n over alphabet $\Sigma = [1, n^{\mathcal{O}(1)}]$, there exists an $\mathcal{O}(n)$-space data structure that answers $S = f_{X,\mathcal{D},\mathcal{I}}$ queries in $\mathcal{O}(|X|\tau)$ time. The preprocessing algorithm runs in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Since S is of size $|X| \cdot \tau$ (it consists of $|X| \cdot \tau$ integers), the complexity bounds are optimal with respect to the preprocessing and query times.

References

1 Paniz Abedin, Arnab Ganguly, Solon P. Pissis, and Sharma V. Thankachan. Efficient data structures for range shortest unique substring queries. *Algorithms*, 13(11):276, 2020. doi:10.3390/A13110276.

2 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 534–544. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743504.

3 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/S00453-020-00744-0.

4 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

5 Golnaz Badkobeh, Panagiotis Charalampopoulos, Dmitry Kosolobov, and Solon P. Pissis. Internal shortest absent word queries in constant time and linear space. *Theor. Comput. Sci.*, 922:271–282, 2022. doi:10.1016/J.TCS.2022.04.029.

6 Golnaz Badkobeh, Panagiotis Charalampopoulos, and Solon P. Pissis. Internal shortest absent word queries. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland*, volume 191 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CPM.2021.6.

7 Tim Baumann and Torben Hagerup. Rank-select indices without tears. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2019. doi:10.1007/978-3-030-24766-9_7.

8 Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. Weighted ancestors in suffix trees revisited. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland*, volume 191 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.8.

9 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

10 William I. Chang and Eugene L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994. doi:10.1007/BF01185431.

11 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Counting distinct patterns in internal dictionary matching. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.8.

12 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal dictionary matching. *Algorithmica*, 83(7):2142–2169, 2021. doi:10.1007/S00453-021-00821-Y.

13 Maxime Crochemore, Costas S. Iliopoulos, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Internal quasiperiod queries. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval – 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2020. doi:10.1007/978-3-030-59212-7_5.

14 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.

15 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In Daniel S. Hirschberg and Eugene W. Myers, editors, *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Laguna Beach, California, USA, June 10-12, 1996, Proceedings*, volume 1075 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 1996. doi:10.1007/3-540-61258-0_11.

16 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

17 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

19 Lucas Chi Kwong Hui. Color set size problem with application to string matching. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 1992. doi:10.1007/3-540-56024-6_19.

20 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

21 Tsvi Kopelowitz, Gregory Kucherov, Yakov Nekrich, and Tatiana Starikovskaya. Cross-document pattern matching. *J. Discrete Algorithms*, 24:40–47, 2014. doi:10.1016/J.JDA.2013.05.002.

22 Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 565–574. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283444.

23 Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Comput. Surv.*, 53(5):105:1–105:35, 2021. doi:10.1145/3409371.

24 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 232–240. ACM, 2006. doi:10.1145/1132516.1132551.

25 Solon P. Pissis, Michael Shekelyan, Chang Liu, and Grigorios Loukides. Frequency-constrained substring complexity. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26-28, 2023, Proceedings*, volume 14240 of *Lecture Notes in Computer Science*, pages 345–352. Springer, 2023. doi:10.1007/978-3-031-43980-3_28.

26 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

27 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

28 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

29 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Inf. Process. Lett.*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

Range Reporting for Time Series via Rectangle Stabbing

Lotte Blank
University of Bonn, Germany

Anne Driemel
University of Bonn, Germany

Abstract

We study the Fréchet queries problem. It is a data structure problem for range reporting, where we are given a set S of n polygonal curves and a distance threshold ρ. The data structure should support queries with a polygonal curve q for the elements of S, for which the continuous Fréchet distance to q is at most ρ. Afshani and Driemel in 2018 studied this problem for two-dimensional polygonal curves of constant complexity and gave upper and lower bounds on the space-query time tradeoff. We study the case that the ambient space of the curves is one-dimensional and show an intimate connection to the well-studied rectangle stabbing problem. Here, we are given a set of hyperrectangles as input and a query with a point q should return all input rectangles that contain this point. Using known data structures for rectangle stabbing or orthogonal range searching this directly leads to a data structure with size in $\mathcal{O}(n \log^{t-1} n)$ and query time in $\mathcal{O}(\log^{t-1} n + k)$, where k denotes the output size and t can be chosen as the maximum number of vertices of either (a) the stored curves or (b) the query curves. Note that we omit factors depending on the complexity of the curves that do not depend on n. The resulting bounds improve upon the bounds by Afshani and Driemel in both the storage and query time. In addition, we show that known lower bounds for rectangle stabbing and orthogonal range reporting with dimension parameter $d = \lfloor t/2 \rfloor$ can be applied to our problem via reduction.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Fréchet distance, Rectangle Stabbing, Orthogonal Range Searching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.15

Related Version *Full Version*: https://arxiv.org/abs/2401.03762

Funding This work was funded by 390685813 (Germany's Excellence Strategy – EXC-2047/1: Hausdorff Center for Mathematics); 416767905; and the Deutsche Forschungsgemeinschaft (DFG, German ResearchFoundation) – 459420781 (FOR AlgoForGe).

1 Introduction

The *Fréchet distance* is a popular measure of similarity of two curves q and s with broad application in many areas, including geographical information science [15, 17, 18], computational biology [14, 19], image processing [3, 16], and quantum chemistry [20]. We focus on a data structuring problem for range reporting which we refer to as the *Fréchet queries problem*. Here, in the preprocessing phase, we are given a set S of n polygonal curves and the distance threshold ρ. The task is to store this set in a data structure that can answer the following type of queries efficiently: For a polygonal curve q, output all curves in S that have distance at most ρ to q. Afshani and Driemel [2] studied this problem in 2018 for two-dimensional curves providing non-trivial upper bounds for the exact case. Recently, Cheng and Huang [8] have generalized their approach for higher dimensions. Other works on variants of this problem have focused on the approximate setting [5, 9, 10, 11, 13].

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 15; pp. 15:1–15:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We focus on the exact setting and – following previous work [5, 10] – we restrict the ambient space of the curves to be 1-dimensional, that is, they are time series. Time series appear in massive amounts in many different applications where they are used to track, e.g., the change over time in stock market value, vitality parameters of patients, atmospheric conditions, such as temperature, the Earth's population, and the hourly requests of a webpage.

In the following, we refer to the number of vertices of a polygonal curve as the complexity of the curve and usually assume that this complexity is constant. We will see that this complexity roughly corresponds to the dimension of the problem when viewed as a rectangle stabbing or orthogonal range reporting problem.

Previous work: Exact setting. Afshani and Driemel [2] proposed a data structure based on multi-level partition trees for two-dimensional curves using semi-algebraic range searching. An essential ingredient to their work is the use of a finite number of predicates that retain sufficient information on the curves to solve the Fréchet queries problem within the partition tree framework. The resulting data structure for polygonal curves in the plane has size in $\mathcal{O}\left(n(\log\log n)^{\mathcal{O}({t_s}^2)}\right)$ and uses query time in $\mathcal{O}\left(\sqrt{n}\cdot\log^{\mathcal{O}({t_s}^2)} n + k\right)$, where t_s denotes the complexity of the input curves and k the output size. The same technique can be applied to solve the problem for 1-dimensional curves using orthogonal range searching. In this case, their bounds reduce to size in $\mathcal{O}\left(n\left(\log n/\log\log n\right)^{\mathcal{O}({t_s}^2)}\right)$ and query time in $\mathcal{O}\left(\log n\left(\log n/\log\log n\right)^{\mathcal{O}({t_s}^2)} + k\right)$. For all time series s of complexity t_s, an $\mathcal{O}({t_s}^2)$-dimensional point $p(s)$ is stored. The dimension of $p(s)$ is quadratic in t_s, because for every pair of vertices of s values depending on both vertices are stored. These are used to evaluate the predicates mentioned above. We substantially simplify these predicates and show that this leads to improved bounds in the 1-dimensional case.

Cheng and Huang [8] used the same predicates as Afshani and Driemel to build a data structure for d-dimensional polygonal curves. They constructed a set of polynomials such that their signs encode the truth values of those predicates. This leads to a data structure of size in $\mathcal{O}(t_q t_s n)^{\mathcal{O}(d^4 {t_q}^2 \log(dt_q))}$ and query time in $\mathcal{O}((dt_q)^{O(1)}\log(t_q t_s n) + k)$, where t_q denotes the complexity of the query time series.

Afshani and Driemel [2] also proved lower bounds in the pointer machine model. Using a volume argument, they show a lower bound stating that every data structure with query time in $Q(n) + \mathcal{O}(k)$, where k is the output size, has to use roughly space in $\Omega((n/Q(n))^2)$ in the worst case even if the query curves are just line segments or points for the discrete Fréchet distance.

Previous work: Approximate setting. Bringmann, Driemel, Nusser, and Psarros [5] studied the setting of 1-dimensional curves. Their work focuses on the c-approximate version of the near-neighbor (c-ANN) problem under the Fréchet distance. In this problem, only one of the curves in the query range needs to be reported and only if the query range is not empty. The approximation is defined with respect to the query radius. Using a bucketing method, they construct a set of curves S' depending on the input curves S, which are stored in a dictionary. They show that, given a query curve q, there must exist a curve in S' very close to q if there exists some curve in S within distance ρ of q. In this way, they constructed a $(1+\varepsilon)$-ANN data structure of size in $n\cdot\mathcal{O}(t_s/(t_q\varepsilon))^{t_q}$ and query time in $\mathcal{O}(1)^{t_q}$. The query time reduces to $\mathcal{O}(t_q)$ with the same space bound for the $(2+\varepsilon)$-ANN data structure. Furthermore, a $(2+\varepsilon)$-ANN data structure with linear size $\mathcal{O}(t_s n)$ and query time in $\mathcal{O}(1/\varepsilon)^{t_q+2}$ is obtained. Their lower bounds show tightness of these bounds in several parameters, assuming the complexity of the curves depends on n (i.e., it is not a constant). To this end, they consider the total time necessary to build the data structure and to answer n queries. They show that, assuming the Orthogonal Vectors Hypothesis, a running time of $n \cdot (t_s/t_q)^{\Theta(t_q)}$ is necessary for any data structure that achieves an approximation factor $\alpha \in [1,2)$.

The conditional lower bounds of Bringmann et al. [5] also apply to the exact setting, however, they assume the parameters t_s and t_q to be non-constant. In light of this, we focus on the setting where t_s and t_q are constants independent of n.

Our results. Section 2 contains a formal definition of the data structure problem studied in this paper: the Fréchet queries problem. Section 3 contains the definition and known results for rectangle stabbing, as well as its dual problem, orthogonal range reporting. Our analysis shows an intimate connection to these classical problems studied in computational geometry as we use them for deriving both upper and lower bounds for the Fréchet queries problem. We start in Section 4 with a reduction showing that both rectangle stabbing and orthogonal range reporting in d dimensions can be solved using a data structure for the Fréchet queries problem using curves of complexity $t = 2d$.

In Section 5, we review the known predicates of Afshani and Driemel [2] which are used to test the Fréchet distance within the partition tree framework. Section 6 contains our main lemmas for simplifying these predicates and introduces the new concept of forward and backward numbers. Here, we take advantage of the fact that the direction of each edge of a time series can only be orientated forward or backward with respect to the x-axis.

The resulting data structures are presented in Section 7. We present two variants. Let t_s be the complexity of the input and t_q of the query and assume t_s and t_q are constant. The first data structure has size in $\mathcal{O}(n \log^{t_q-2} n)$ and uses query time in $\mathcal{O}(\log^{t_q-1} n + k)$ and is independent of t_s, except for a constant factor of the form $\left(\frac{t_s}{t_q}\right)^{t_q}$. The second data structure has size in $\mathcal{O}(n(\log n/\log\log n)^{t_s-1})$ and query time in $\mathcal{O}(\log n(\log n/\log\log n)^{t_s-3} + k)$ and is independent of t_q, except for a constant factor of the form $\left(\frac{t_q}{t_s}\right)^{t_s}$. In both variants, k denotes the size of the output (without duplicates).

Together with known lower bounds for rectangle stabbing and orthogonal range reporting, our analysis in Section 4 implies that every data structure that solves the Fréchet queries problem and uses nh space has to use query time in $\Omega(\log n(\log n/\log h)^{\lfloor t/2\rfloor-2} + k)$, where $t = \min\{t_q, t_s\}$. If the data structure uses query time in $\mathcal{O}(\log^c n + k)$, where c is a constant, it must use space in $\Omega(n(\log n/\log\log n)^{\lfloor t/2\rfloor-1})$.

2 Problem Definition

For any two points $p, q \in \mathbb{R}^d$, $\overline{pq}$ is the directed line segment from p to q. The linear interpolation of each pair of consecutive vertices of a sequence of vertices $s_1, \ldots, s_{t_s} \in \mathbb{R}^d$ is called a polygonal curve and its *complexity* is the number of its vertices. This curve is also denoted as $\langle s_1, \ldots, s_{t_s}\rangle$. We can represent polygonal curves as functions $s : [1, t_s] \to \mathbb{R}^d$, where $s(i+\alpha) = (1-\alpha)s_i + \alpha s_{i+1}$ for $i \in \{1, \ldots, t_s\}$ and $\alpha \in [0,1]$. The *(continuous) Fréchet distance* between polygonal curves $q : [1, t_q] \to \mathbb{R}^d$ and $s : [1, t_s] \to \mathbb{R}^d$ is defined as

$$d_{\mathrm{F}}(q,s) = \inf_{h_q \in \mathcal{F}_q, h_s \in \mathcal{F}_s} \max_{p \in [0,1]} \|q(h_q(p)) - s(h_s(p))\|_2,$$

where $\mathcal{F}_q$ is the set of all continuous, non-decreasing functions $h_q : [0,1] \to [1, t_q]$ with $h_q(0) = 1$ and $h_q(1) = t_q$, respectively $\mathcal{F}_s$ for s.

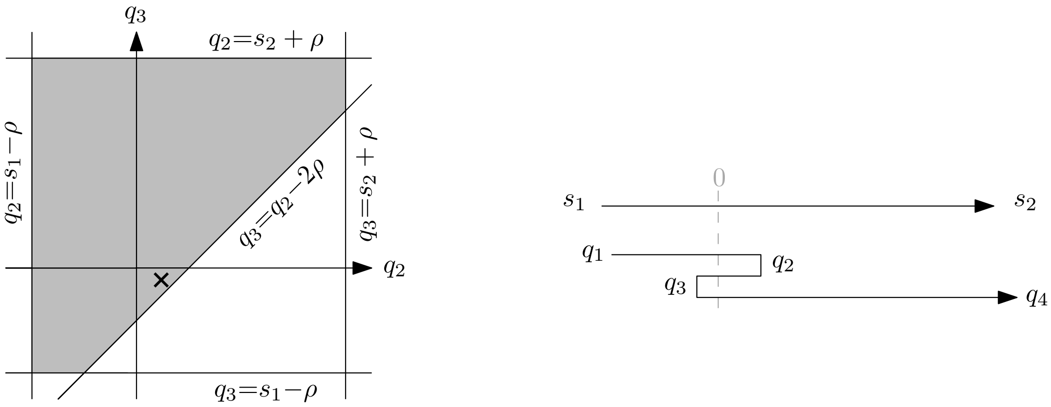


Figure 1 The second and third vertices of the time series q such that $d_F(q, s) \leq \rho$ of Example 2. Additionally, it must hold that $q_1 \in [s_1 - \rho, s_1 + \rho]$ and $q_4 \in [s_2 - \rho, s_2 + \rho]$. On the right is an example for such a time series q with respect to s and the corresponding point (p_2, p_3) is marked. In this paper, the vertices of the time series are drawn as vertical segments for clarity.

▶ **Problem 1** (Fréchet queries). *Given a set S of n time series all of complexity at most t_s, the complexity t_q of the query time series and a distance parameter $\rho \in \mathbb{R}_{\geq 0}$. Find a data structure that stores this set S and can answer the following type of queries. For any query time series q of complexity t_q, return all elements of S that have continuous Fréchet distance at most ρ to q.*

▶ **Example 2.** We give a simple example demonstrating why our results are surprising. At first sight, it seems intriguing to believe that the set of queries corresponding to an input curve can be viewed as a finite union of axis-aligned hyperrectangles in the dimension of the (fixed) query curve complexity t_q. However, a simple example shows that this is not always the case. Let $s = \langle s_1, s_2 \rangle$ be a time series of complexity 2, where $s_1 \leq s_2$. In Section 6, we show the following statement. For every time series $q = \langle q_1, q_2, q_3, q_4 \rangle$ of complexity 4, it holds that $d_F(q, s) \leq \rho$ if and only if

- $q_1 \in [s_1 - \rho, s_1 + \rho]$,
- $q_2,\ q_3 \in [s_1 - \rho, s_2 + \rho]$,
- $q_4 \in [s_2 - \rho, s_2 + \rho]$, and
- $q_3 \geq q_2 - 2\rho$.

The (non-orthogonal) condition $q_3 \geq q_2 - 2\rho$ stems from the monotonicity requirement in the definition of the Fréchet distance. The query space can be re-parameterized by introducing new variables to overcome this and to obtain a finite union of axis-aligned hyperrectangles, as this is implicitly done by Afshani and Driemel [2]. For this specific example, we can introduce an additional variable h with $h = q_2 - q_3$ and $h \in [s_1 - s_2 - 2\rho, 2\rho]$. Achieving this with only a few additional variables (without blowing up the dimension quadratically as in the work of Afshani and Driemel) is the main challenge of our work. The key ingredient to our analysis is a simplification of the predicates – which goes along with a reduction of their overall number.

3 Data Structure Techniques

In this paper, we will show an intimate connection of the Fréchet queries problem in one dimension to rectangle stabbing and orthogonal range searching. We first describe these data structure problems independently and state the known results we will use in our analysis.

Rectangle Stabbing. For rectangle stabbing the task is as follows. Preprocess a set S of n axis-aligned d-dimensional rectangles in $\mathbb{R}^d$ into a data structure so that all rectangles in S containing a query point q can be reported efficiently, ensuring that each such rectangle is reported exactly once.

Chazelle [6] developed a data structure for this problem with constant dimension d that has size in $\mathcal{O}\left(n \log^{d-2} n\right)$ and query time in $\mathcal{O}(\log^{d-1} n + k)$, where k is the size of the output. Afshani, Arge and Larsen [1] proved the following lower bound for the rectangle stabbing problem. Any data structure that operates on a pointer machine and uses nh space must use query time in $\Omega\left(\log n (\log n / \log h)^{d-2} + k\right)$, where k is the output size. To prove a lower bound for the Fréchet queries problem, we need a bounded version of rectangle stabbing. Here, all rectangles in S are contained in $[0,1]^d$. The constructive proof for the lower bound uses only instances, where the input rectangles are all contained in a d-dimensional cube with side length $m < n$. By scaling this instance, we obtain that the lower bound holds also for bounded rectangle stabbing.

Orthogonal Range Searching. Orthogonal range searching is defined as follows. Preprocess a set S of n points in $\mathbb{R}^d$ into a data structure so that for a d-dimensional axis-aligned query rectangle R all points contained in S can be reported efficiently, ensuring that each such point is reported exactly once.

Afshani, Arge and Larsen [1] constructed a data structure for constant dimension $d > 3$ using space in $\mathcal{O}\left(n(\log n / \log\log n)^{d-1}\right)$ and query time in $\mathcal{O}\left(\log n (\log n / \log\log n)^{d-3} + k\right)$, where k is the size of the output. Later, we reduce the orthogonal range searching problem to the Fréchet queries problem and then use the following lower bound by Chazelle [7]. Consider a data structure of orthogonal range searching on n points in $\mathbb{R}^d$ that operates on a pointer machine, and let c be an arbitrary constant. If the data structure provides a query time in $\mathcal{O}((\log n)^c + k)$, where k is the output size, then its size must be in $\Omega(n(\log n / \log\log n)^{d-1})$.

4 Lower Bounds

We transform the bounded rectangle stabbing problem to the Fréchet queries problem such that we can use a known lower bound for the bounded rectangle stabbing problem to obtain a lower bound for the Fréchet queries problem. An illustration of the reduction can be found in Example 3.

Given a set S of n axis-aligned rectangles contained in $[0,1]^d$ as an instance of the d-dimensional bounded rectangle stabbing problem. We define a set S' containing n time series of complexity $2d$. For a rectangle $R = [l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_d, r_d]$ in S, we store the time series $s(R) = \langle s_1, \ldots, s_{2d} \rangle$, where

$$s_{2i-1} = (r_i + 1) + 6i \text{ and } s_{2i} = (l_i - 1) + 6i.$$

The set S' is stored in a data structure for the Fréchet queries problem. We define a query time series $q = \langle q_1, \ldots, q_{2d} \rangle$ for a query point $p = (p_1, \ldots, p_d) \in [0,1]^d$, where

$$q_{2i-1} = (p_i + 2) + 6i \text{ and } q_{2i} = (p_i - 2) + 6i.$$

To find all time series $s(R)$ in S' within Fréchet distance at most 1 to q, we use the stored data structure. All rectangles R, where $d_{\mathrm{F}}(q, s(R)) \leq 1$, will be returned. Theorem 4 implies that this reduction is correct by showing that $p \in R \Leftrightarrow d_{\mathrm{F}}(q, s(R)) \leq 1$.

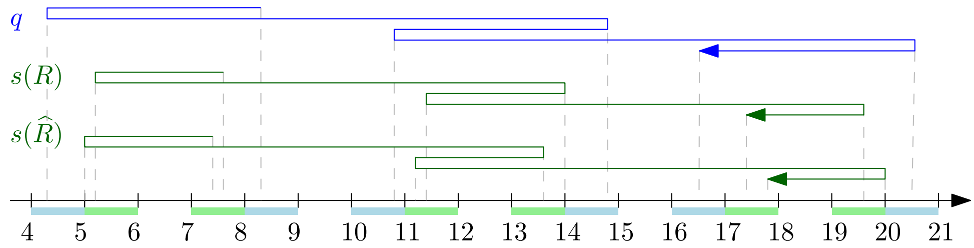


Figure 2 The time series q, $s(R)$ and $s(\widehat{R})$ as in Example 3.

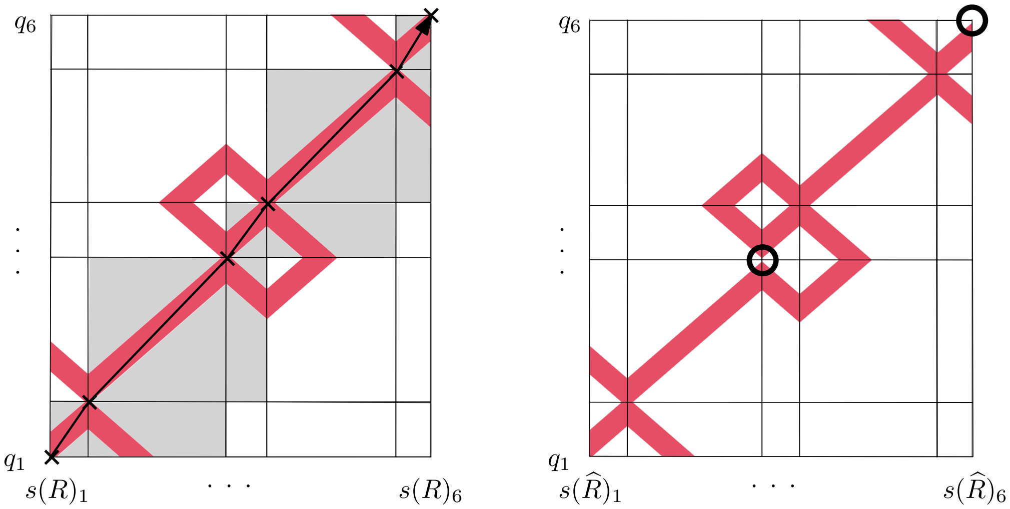


Figure 3 The free space diagrams $F_1(q, s(R))$ and $F_1(q, s(\widehat{R}))$ defined in Example 3. A sequence of cells $\mathcal{C}$ that is feasible in $F_1(q, s(R))$ is drawn in grey.

▶ **Example 3.** The input set S of a rectangle stabbing instance contains the rectangles $R = [0.2, 0.6] \times [0.4, 1] \times [0.4, 0.6]$ and $\widehat{R} = [0, 0.4] \times [0.2, 0.6] \times [0.8, 1]$ and the query point is $p = (0.3, 0.8, 0.5)$. It is evident that $p \in R$ and $p \notin \widehat{R}$. Through our reduction, the two stored time series are $s(R) = \langle 7.6, 5.2, 14, 11.4, 19.6, 17.4\rangle$ and $s(\widehat{R}) = \langle 7.4, 5, 13.6, 11.2, 20, 17.8\rangle$ and the query time series for p is $q = \langle 8.3, 4.3, 14.8, 10.8, 20.5, 16.5\rangle$, as illustrated in Figure 2. The left side of Figure 3 depicts the free space diagram of the time series q and $s(R)$ with respect to $\rho = 1$ (i.e., $F_1(q, s(R))$). Notably, the points (i, i) lie in the free space for all i, resulting in $d_{\mathrm{F}}(q, s(R)) \leq 1$. Conversely, the right side of Figure 3 corresponds to the free space diagram of the time series q and $s(\widehat{R})$ with respect to $\rho = 1$ (i.e., $F_1(q, s(\widehat{R}))$). It does not contain a feasible path and $|q_3 - s(\widehat{R})_3| > 1$ and $|q_6 - s(\widehat{R})_6| > 1$.

▶ **Theorem 4.** *The d-dimensional bounded rectangle stabbing problem can be solved with a data structure for the Fréchet queries problem, where the stored time series as well as the query time series have complexity $2d$. The instance for the Fréchet queries problem can be computed in linear time.*

Proof. We use the reduction as described above. Hence, it remains to prove that $p \in R \Leftrightarrow d_{\mathrm{F}}(q, s(R)) \leq 1$. For all i, it follows by $l_i, r_i, p_i \in [0, 1]$ that

$$|q_{2i-1} - s_{2i-1}| \leq 1 \;\Leftrightarrow\; |((p_i + 2) + 6i) - ((r_i + 1) + 6i)| \leq 1 \;\Leftrightarrow\; p_i \leq r_i \text{ and}$$
$$|q_{2i} - s_{2i}| \leq 1 \;\Leftrightarrow\; |((p_i - 2) + 6i) - ((l_i - 1) + 6i)| \leq 1 \;\Leftrightarrow\; p_i \geq l_i.$$

If $p \in R$, then it holds that $|q_{2i-1} - s_{2i-1}| \leq 1$ and $|q_{2i} - s_{2i}| \leq 1$ for all i, since $l_i \leq p_i \leq r_i$. Therefore, $\mathcal{C} = ((1,1),(1,2),(2,2),(2,3),(3,3),(3,4),\ldots,(2d-1,2d),(2d,2d))$ is a feasible sequence of cells in $F_1(q, s(R))$ because all cells are convex and boundary points of a cell belong to all neighboring cells. So, $d_\mathrm{F}(q, s(R)) \leq 1$.

If $d_\mathrm{F}(q,s) \leq 1$, then by the definition of the Fréchet distance, for all points q_{2i-1} and q_{2i}, there exist points x_{2i-1} and x_{2i} such that $|q_{2i-1} - x_{2i-1}| \leq 1$, $|q_{2i} - x_{2i}| \leq 1$, and x_{2i-1} lies not after x_{2i} on the time series $s(R)$. By construction, it holds $s_{2k} - 1 < s_{2k-1} - 1 < q_{2i} < q_{2i-1} < s_{2l} + 1 < s_{2l-1} + 1$ for all $k < i < l$. It holds that $|s_k - q_l| > 1$ for $k \neq l$ by definition. Therefore, x_{2i-1}, x_{2i} must lie on one of the following edges $\overline{s_{2i-2}s_{2i-1}}$, $\overline{s_{2i-1}s_{2i}}$, or $\overline{s_{2i}s_{2i+1}}$. By construction, it holds that $q_{2i-1} \in [6i+2, 6i+3]$ and $|x_{2i-1} - q_{2i-1}| \leq 1$. Hence, $x_{2i-1} \in [6i+1, 6i+4]$. Further, $q_{2i} \in [6i-2, 6i-1]$ and $|x_{2i} - q_{2i}| \leq 1$. Hence, $x_{2i} \in [6i-3, 6i]$. Assume that $x_{2i-1} \in \overline{s_{2i}s_{2i+1}}$. Then since x_{2i} lies after x_{2i-1} on $s(R)$ it follows that $x_{2i} \in \overline{s_{2i}s_{2i+1}}$ and in particular $6i + 1 \leq x_{2i-1} \leq x_{2i}$. This leads to a contradiction to $x_{2i} \leq 6i$. In the same way, it follows that $x_{2i} \notin \overline{s_{2i-2}s_{2i-1}}$. So, x_{2i-1} lies on $\overline{s_{2i-2}s_{2i-1}}$ or $\overline{s_{2i-1}s_{2i}}$ and x_{2i} lies on $\overline{s_{2i-1}s_{2i}}$ or $\overline{s_{2i}s_{2i+1}}$. It holds that $x_{2i-1} \leq s_{2i-1}$ and $x_{2i} \geq s_{2i}$, because $s_{2i-2} \leq s_{2i} \leq s_{2i-1} \leq s_{2i+1}$. It follows by $x_{2i-1} \leq s_{2i-1} \leq q_{2i-1}$ and $|x_{2i-1} - q_{2i-1}| \leq 1$ that $|q_{2i-1} - s_{2i-1}| = q_{2i-1} - s_{2i-1} \leq q_{2i-1} - x_{2i-1} = |q_{2i-1} - x_{2i-1}| \leq 1$. By the same argument, it follows that $|s_{2i} - q_{2i}| \leq 1$ because $q_{2i} \leq s_{2i} \leq x_{2i}$. Therefore, $l_i \leq p_i \leq r_i$ for all i, i.e., $p \in R$, which concludes the proof. ◀

The result in Theorem 4 together with the lower bound for bounded rectangle stabbing queries by Afshani, Arge and Larsern [1] yields the following lower bound for the Fréchet queries problem.

▶ **Corollary 5.** *Every data structure that solves the Fréchet queries problem that operates on a pointer machine, and uses nh space must use query time in $\Omega(\log n (\log n / \log h)^{\lfloor t/2 \rfloor - 2} + k)$, where k is the size of the output (without duplicates) and $t = \min\{t_q, t_s\}$.*

Given an instance of d-dimensional orthogonal range searching, we can construct the stored (resp. query) time series in the way as the query (resp. stored) time series were constructed in Theorem 4 after scaling the instance such that all points are in $[0,1]^d$. Using this construction, it holds by the same arguments as in the proof of Theorem 4 that $p \in R$ if and only if $d_F(q, s(p)) \leq 1$. Therefore, we get the following corollary.

▶ **Corollary 6.** *The d-dimensional orthogonal range searching can be solved with a data structure for the Fréchet queries problem, where the stored time series as well as the query time series have complexity $2d$. The instance for the Fréchet queries problem can be computed in linear time.*

Chazelles [7] lower bound for orthogonal range searching provides to the following:

▶ **Corollary 7.** *Every data structure that solves the Fréchet queries problem and uses query time in $\mathcal{O}(\log^c n + k)$, where c is a constant, must use size in $\Omega(n(\log n / \log\log n)^{\lfloor t/2 \rfloor - 1})$, where k is the size of the output (without duplicates) and $t = \min\{t_q, t_s\}$.*

5 Predicates for Evaluating the Fréchet distance

In this section, we review the predicates used by Afshani and Driemel [2] and how they enable the evaluation of the Fréchet distance in a data structure context.

For this, we first recall the definition of the free space diagram from Alt and Godau [4]. For polygonal curves $q : [1, t_q] \to \mathbb{R}^d$ and $s : [1, t_s] \to \mathbb{R}^d$ the *free space diagram* $F_\rho(q,s)$ is a subset of $[1, t_q] \times [1, t_s]$, such that for all points $(x,y) \in F_\rho(q,s)$ the distance between $q(x)$ and $s(y)$ is at most ρ. Refer to Figure 4 for an example. Formally,

$$F_\rho(q,s) := \{(x,y) \in [1,t_q] \times [1,t_s] \mid \|q(x) - s(y)\|_2 \leq \rho\}.$$

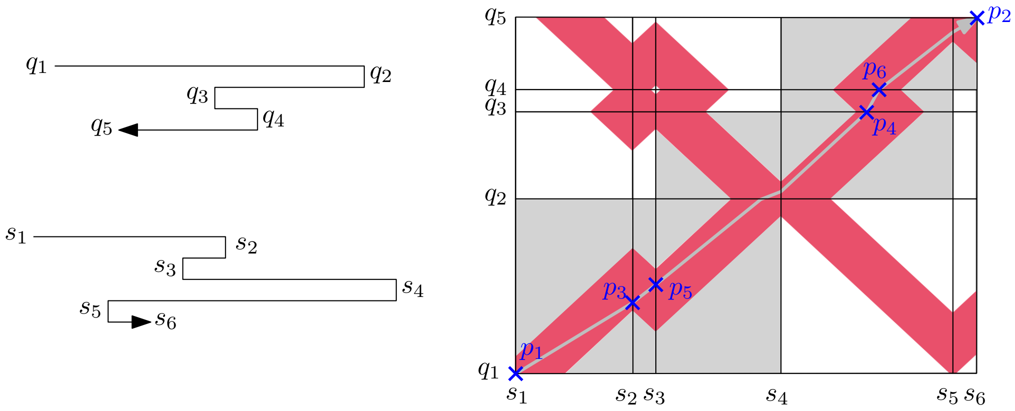


Figure 4 The free space diagram $F_\rho(q,s)$ of two time series with a feasible path trough a feasible sequence of cells $\mathcal{C} = ((1,1),(1,2),(1,3),(2,3),(2,4),(3,4),(4,4),(4,5))$, which is drawn in grey. Predicates $(P_1),(P_2),(P_3(1,2)),(P_4(3,4)),(P_5(1,2,3))$ and $(P_6(3,4,4))$ are true, because the points p_i are contained in the free space.

▶ **Lemma 8** (Alt and Godau [4]). *The Fréchet distance between polygonal curves s and q is at most ρ if and only if there exists a path in $F_\rho(q,s)$ from $(1,1)$ to (t_q,t_s) which is monotone in both coordinates. For such a path, we say it is* feasible.

We can decompose the rectangle $[1,t_q]\times[1,t_s]$ into $(t_q-1)\cdot(t_s-1)$ cells such that the cell $C_{ij}=[i,i+1]\times[j,j+1]$ corresponds to the part in the free space diagram defined by the edges $\overline{q_iq_{i+1}}$ and $\overline{s_js_{j+1}}$. By definition of the free space diagram, it follows that $C_{ij}\cap F_\rho(q,s)$ lies between two parallel lines. Therefore, we focus on the boundary of the cells C_{ij}.

Our query algorithm will iterate over all possibilities of sequences of cells that a feasible path could traverse in the free space diagram. Therefore, we call a sequence of cells $\mathcal{C}=((i_1,j_1),\ldots,(i_t,j_t))$ *valid*, if $i_1=1$, $j_1=1$, $i_t=t_q-1$, $j_t=t_s-1$, and for all $m<t$ either $i_{m+1}=i_m$ and $j_{m+1}=j_m+1$, or $i_{m+1}=i_m+1$ and $j_{m+1}=j_m$. The tuple (i,j) represents the cell C_{ij}. Further, a valid sequence of cells is called *feasible* in $F_\rho(q,s)$, if there exists a feasible path in $F_\rho(q,s)$ that traverses exactly the cells in $\mathcal{C}$. Refer to Figure 4 for an example.

The following predicates due to Afshani and Driemel [2] can be used to decide whether a valid sequence of cells is feasible in $F_\rho(q,s)$. Figure 4 visualizes the predicates.

(P_1) *(Endpoint (start))* This predicate is true iff $|s_1-q_1|\le\rho$.
(P_2) *(Endpoint (end))* This predicate is true iff $|s_{t_s}-q_{t_q}|\le\rho$.
$(P_3(i,j))$ *(Vertex of s – edge of q)* This predicate is true iff $\exists\, p_3\in\overline{q_iq_{i+1}}$ s.t. $|p_3-s_j|\le\rho$.
$(P_4(i,j))$ *(Vertex of q – edge of s)* This predicate is true iff $\exists\, p_4\in\overline{s_js_{j+1}}$ s.t. $|p_4-q_i|\le\rho$.
$(P_5(i,j,k))$ *(Monotone in q)* This predicate is true iff $\exists\, p_3,p_5\in\overline{q_iq_{i+1}}$ s.t. p_3 lies not after p_5 on the time series q and $|p_3-s_j|\le\rho$ and $|p_5-s_k|\le\rho$.
$(P_6(i,l,j))$ *(Monotone in s)* This predicate is true iff $\exists\, p_4,p_6\in\overline{s_js_{j+1}}$ s.t. p_4 lies not after p_6 on the time series s and $|p_4-q_i|\le\rho$ and $|p_6-q_l|\le\rho$.

The following lemma verifies that the predicates can be used to test if the Fréchet distance between two curves is at most a given value.

▶ **Lemma 9** (Afshani and Driemel [2]). *Let $\mathcal{C}=((i_1,j_1),(i_2,j_2),\ldots,(i_t,j_t))$ be a valid sequence of cells. Then $\mathcal{C}$ is feasible in $F_\rho(q,s)$ if and only if the following predicates defined by q, s and ρ are true:*

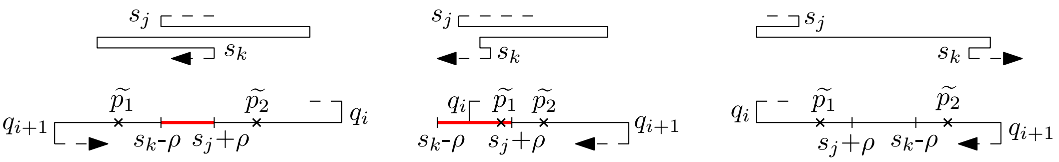


Figure 5 The left and the middle show Case (i) of Lemma 11. Here, $[s_j-\rho, s_j+\rho]\cap[s_k-\rho, s_k+\rho]$ is marked in red. The right visualizes Case (ii) of Lemma 11 and Case (iii) is Case (ii) mirrored.

(i) (P_1) *and* (P_2),
(ii) $(P_3(i,j))$ *if* $(i,j-1),(i,j)\in\mathfrak{C}$,
(iii) $(P_4(i,j))$ *if* $(i-1,j),(i,j)\in\mathfrak{C}$,
(iv) $(P_5(i,j,k))$ *if* $(i,j-1),(i,k)\in\mathfrak{C}$, *and*
(v) $(P_6(i,l,j))$ *if* $(i-1,j),(l,j)\in\mathfrak{C}$.
We say that those predicates are induced by $\mathfrak{C}$.

Afshani and Driemel [2] showed that, for a given query, a fixed assignment of truth values to the set of all predicates defines a semi-algebraic set. This set contains all curves for which the predicates yield the given truth assignment. A query to the data structure then corresponds to a finite union of semi-algebraic range queries for which the truth assignments yield a valid sequence of cells.

In our paper, we modify this approach. Instead of fixing the truth assignment to all predicates, we only fix a combinatorial path in the free space diagram (that is a valid sequence of cells) and we consider the predicates that are induced by it. This results in potential duplicates in the query output as an input curve may have different combinatorial paths in the free space diagram with the query. However, the overall number of elements in the output only changes by a constant factor as long as the complexity of the input and query is constant.

6 Simplification of the Predicates

Given a sequence of cells $\mathfrak{C}$, we want to find intervals $I_1,\ldots,I_{t_q}$ defined by a stored time series s such that $\mathfrak{C}$ is feasible in $F_\rho(q,s)$ if and only if $q_i\in I_i$ for all i, where $q=\langle q_1,\ldots,q_{t_q}\rangle$ is a time series with some additional properties. The intervals will be defined using the predicates. Lemma 9 shows which predicates need to be true such that $\mathfrak{C}$ is feasible in $F_\rho(q,s)$. For the endpoint and vertex-edge predicates $((P_1),(P_2),(P_3)$ and $(P_4))$, the needed intervals follow easily:

▶ **Observation 10.** *Let* $q=\langle q_1,\ldots,q_{t_q}\rangle$ *and* $s=\langle s_1,\ldots,s_{t_s}\rangle$ *be two time series. Then the following holds for the predicates in the free space diagram* $F_\rho(q,s)$*:*
(i) (P_1) *is true* $\Leftrightarrow q_1\in[s_1-\rho, s_1+\rho]$,
(ii) (P_2) *is true* $\Leftrightarrow q_{t_q}\in[s_{t_s}-\rho, s_{t_s}+\rho]$,
(iii) $(P_3(i,j))$ *is true* $\Leftrightarrow$ *if* $q_i\le q_{i+1}:\ q_i\le s_j+\rho$ *and* $q_{i+1}\ge s_j-\rho$ *and*
if $q_i\ge q_{i+1}:\ q_i\ge s_j-\rho$ *and* $q_{i+1}\le s_j+\rho$,
(iv) $(P_4(i,j))$ *is true* $\Leftrightarrow q_i\in[\min\{s_j-\rho, s_{j+1}-\rho\},\max\{s_j+\rho, s_{j+1}+\rho\}]$.

The next lemma defines the intervals needed such that the monotone in q predicate $(P_5(i,j,k))$ is true and is visualized in Figure 5.

▶ **Lemma 11.** *Let* $q=\langle q_1,\ldots,q_{t_q}\rangle$ *and* $s=\langle s_1,\ldots,s_{t_s}\rangle$ *be two time series, then the monotone in* q *predicate* $(P_5(i,j,k))$ *is true if and only if the vertex of* s *- edge of* q *predicates* $(P_3(i,j))$ *and* $(P_3(i,k))$ *are true and one of the following holds*

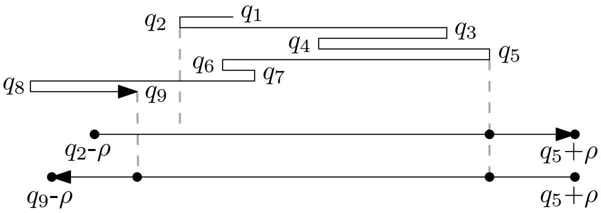


Figure 6 Illustration of the values $f_2(s) = 5$ and $b_5(s) = 9$ for a time series s.

(i) *$|s_j - s_k| \leq 2\rho$, or*
(ii) *$|s_j - s_k| > 2\rho$ and $s_j \leq s_k$ and $q_i \leq s_j + \rho$ and $q_{i+1} \geq s_k - \rho$, or*
(iii) *$|s_j - s_k| > 2\rho$ and $s_j > s_k$ and $q_i \geq s_j - \rho$ and $q_{i+1} \leq s_k + \rho$.*

Proof. Assume $(P_5(i,j,k))$ to be true. Then there exist points $p_1, p_2 \in \overline{q_i q_{i+1}}$ such that p_1 lies not after p_2 on the time series q and $p_1 \in [s_j - \rho, s_j + \rho]$, $p_2 \in [s_k - \rho, s_k + \rho]$. Hence, $(P_3(i,j))$ and $(P_3(i,k))$ are true. In addition, if $|s_j - s_k| > 2\rho$ and $s_j \leq s_k$, it holds that $q_i \leq p_1 \leq s_j + \rho < s_k - \rho \leq p_2 \leq q_{i+1}$. Further, if $|s_j - s_k| > 2\rho$ and $s_j > s_k$, it holds that $q_{i+1} \leq p_2 \leq s_k + \rho < s_j - \rho \leq p_1 \leq q_i$.

It remains to prove the other direction. Let $(P_3(i,j))$ and $(P_3(i,k))$ be true. Then, there exist $\widetilde{p_1} \in \overline{q_i q_{i+1}}$ and $\widetilde{p_2} \in \overline{q_i q_{i+1}}$ such that $|\widetilde{p_1} - s_j| \leq \rho$ and $|\widetilde{p_2} - s_k| \leq \rho$.
Case (i): Let $|s_j - s_k| \leq 2\rho$. We can set $p_1 = p_2 = \widetilde{p_1}$ if $\widetilde{p_1} \in [s_j - \rho, s_j + \rho] \cap [s_k - \rho, s_k + \rho]$. The same holds if $\widetilde{p_2} \in [s_j - \rho, s_j + \rho] \cap [s_k - \rho, s_k + \rho]$. Otherwise, $\emptyset \neq [s_j - \rho, s_j + \rho] \cap [s_k - \rho, s_k + \rho] \subseteq \overline{\widetilde{p_1}\widetilde{p_2}} \subseteq \overline{q_i q_{i+1}}$ and we can set $p_1 = p_2$ as any point in this intersection. In each of those cases, $(P_5(i,j,k))$ is true.
Case (ii): Let $|s_j - s_k| > 2\rho$, $s_j \leq s_k$, $q_i \leq s_j + \rho$ and $q_{i+1} \geq s_k - \rho$. Then it holds that $\widetilde{p_1} < \widetilde{p_2}$ and $q_i < q_{i+1}$ because $s_j + \rho < s_k - \rho$. Therefore, $\widetilde{p_1}$ lies before $\widetilde{p_2}$ on $\overline{q_i q_{i+1}}$ and we can simply set $p_1 = \widetilde{p_1}$ and $p_2 = \widetilde{p_2}$. Those points have the required properties in the definition of $(P_5(i,j,k))$. Symmetrically, in Case (iii) it holds that $(P_5(i,j,k))$ is true. ◀

To determine the truth value of the monotone in s predicates (P_6), we define the *forward* and *backward numbers* $f_i(q)$ and $b_i(q)$. Refer to Figure 6 as an example.

▶ **Definition 12** (forward and backward numbers). *For a time series $q = \langle q_1, \dots, q_{t_q} \rangle$ and $i \in \{1, \dots, t_q\}$, we denote by the* forward number *$f_i(q)$ the highest number such that $\langle q_i - \rho, q_{f_i(q)} + \rho \rangle$ is oriented forward and its Fréchet distance to the time series $\langle q_i, \dots, q_{f_i(q)} \rangle$ is at most ρ, i.e.,*

$$f_i(q) := \max\{k \in \{i, \dots, t_q\} \mid d_F(\langle q_i, \dots, q_k \rangle, \langle q_i - \rho, q_k + \rho \rangle) \leq \rho \text{ and } q_i - \rho \leq q_k + \rho\}$$

and by the backward number *$b_i(q)$ the highest number such that $\langle q_i + \rho, q_{b_i(q)} - \rho \rangle$ is oriented backward and its Fréchet distance to the time series $\langle q_i, \dots, q_{b_i(q)} \rangle$ is at most ρ, i.e.,*

$$b_i(q) := \max\{k \in \{i, \dots, t_q\} \mid d_F(\langle q_i, \dots, q_k \rangle, \langle q_i + \rho, q_k - \rho \rangle) \leq \rho \text{ and } q_i + \rho \geq q_k - \rho\}.$$

▶ **Observation 13.** *For all $i \leq x \leq f_i(q)$, it holds that $d_F(\langle q_i, \dots, q_x \rangle, \langle q_i - \rho, q_x + \rho \rangle) \leq \rho$ and $q_i - \rho \leq q_x + \rho$. Respectively, for $b_i(q)$.*

Proof. By the definition of the Fréchet distance and $f_i(q)$, there exist points $q_i - \rho \leq p_i \leq p_{i+1} \leq \dots \leq p_x \leq q_x + \rho$ such that $|p_j - q_j| \leq \rho$ for all j. Further, since the free space in every cell is convex, the statement follows. ◀

The next lemma shows how the forward and backward numbers can be used to determine values of the monotone in s predicates (P_6). To decide whether a valid sequence of cells is feasible or not in $F_\rho(q,s)$, we need predicate $(P_6(i,l,j))$ to be true only if we also need all predicates $(P_6(x,y,j))$ to be true with $i \le x < y \le l$ by Lemma 9.

▶ **Lemma 14.** *Let $q = \langle q_1, \dots, q_{t_q}\rangle$ and $s = \langle s_1, \dots, s_{t_s}\rangle$ be two time series, $i, l \in \{1, \dots, t_q\}$ with $i < l$ and $j \in \{1, \dots, t_s - 1\}$. If $s_j \le s_{j+1}$, then*

$$(P_6(x,y,j)) \text{ is true } \forall i \le x < y \le l \;\Leftrightarrow\; f_i(q) \ge l \text{ and } (P_4(x,j)) \text{ is true for all } i \le x \le l$$

and if $s_j \ge s_{j+1}$, then

$$(P_6(x,y,j)) \text{ is true } \forall i \le x < y \le l \;\Leftrightarrow\; b_i(q) \ge l \text{ and } (P_4(x,j)) \text{ is true for all } i \le x \le l.$$

Proof. We discuss the case that $s_j \le s_{j+1}$. The other case can be proven in the same way.

Let $(P_6(x,y,j))$ be true for all $i \le x < y \le l$. Note that by definition of (P_6), it holds that the predicates $(P_4(x,j))$ are true for all $i \le x \le l$. It remains to prove $f_i(q) \ge l$. Let $i \le x < y \le l$. Since $(P_6(i,x,j))$ is true there exist $p_4, p_6 \in \overline{s_j s_{j+1}}$ such that $p_4 \le p_6$ and $p_i \le p_4 + \rho$ and $q_x \ge p_6 - \rho$. Therefore, it follows that $q_i - q_x \le p_4 + \rho - (p_6 - \rho) \le 2\rho$. In the same way, we get that $q_x - q_l \le 2\rho$. In particular, this implies that $\langle q_i - \rho, q_l + \rho\rangle$ is a forward edge. Assume for the sake of a contradiction that $d_F(\langle q_i, \dots, q_l\rangle, \langle q_i - \rho, q_l + \rho\rangle) > \rho$. Then, there must exist two vertices q_x, q_y such that there are no two points $p_1, p_2 \in [q_i - \rho, q_l + \rho]$ such that $p_1 \le p_2$, $|p_1 - q_x| \le \rho$ and $|p_2 - q_y| \le \rho$. Further, since $(P_6(x,y,j))$ is true we know that there exist $p_4 \le p_6$ on $\overline{s_j s_{j+1}}$ such that $|p_4 - q_x| \le \rho$ and $|p_6 - q_y| \le \rho$. Now set $\widetilde{p_1} = \max\{\min\{q_l + \rho, p_4\}, q_i - \rho\}$ and $\widetilde{p_2} = \max\{\min\{q_l + \rho, p_6\}, q_i - \rho\}$. It holds that $\widetilde{p_1}, \widetilde{p_2} \in [q_i - \rho, q_l + \rho]$ and $\widetilde{p_1} \le \widetilde{p_2}$. Further, since $(q_i - \rho) - q_x \le \rho$ and $q_x - (q_l + \rho) \le \rho$ it holds that $|\widetilde{p_1} - q_x| \le \rho$. Similarly, $|\widetilde{p_2} - q_y| \le \rho$. This contradicts the assumption. Hence, $d_F(\langle q_i, \dots, q_l\rangle, \langle q_i - \rho, q_l + \rho\rangle) \le \rho$ and $\langle q_i - \rho, q_l + \rho\rangle$ is a forward edge. So, $f_i(q) \ge l$.

To prove the other direction, assume $f_i(q) \ge l$ and $(P_4(x,j))$ is true for all $i \le x \le l$. Therefore, by Observation 13, it holds that $d_F(\langle q_i, \dots, q_l\rangle, \langle q_i - \rho, q_l + \rho\rangle) \le \rho$ and $q_i - \rho \le q_l + \rho$. Let $i \le x < y \le l$. Then, there exists points $p_x < p_y$ on the edge $\overline{q_i - \rho, q_l + \rho}$ such that $|p_x - q_x| \le \rho$, $|p_y - q_y| \le \rho$. Further, there exists points $p_1, p_2 \in [s_j, s_{j+1}]$ such that $|p_1 - q_x| \le \rho$, $|p_2 - q_y| \le \rho$ by the properties of predicate (P_4). We define $p_4 = \min\{\max\{s_j, p_x\}, s_{j+1}\}$ and $p_6 = \min\{\max\{s_j, p_y\}, s_{j+1}\}$. It follows that $p_4 \le p_6$ and $p_4, p_6 \in \overline{s_j s_{j+1}}$. Furthermore, if $p_4 = s_j$, then $p_x \le s_j \le p_1$, resulting in $|s_j - q_x| \le \rho$. If $p_4 = s_{j+1}$ then $p_1 \le s_{j+1} \le p_x$ and $|s_{j+1} - q_x| \le \rho$. Therefore, $|p_4 - q_x| \le \rho$. Similarly, it follows that $|p_6 - q_y| \le \rho$. The points p_4, p_6 fulfill the conditions of the definition of the monotone in s predicate $(P_6(x,y,j))$, i.e., $(P_6(x,y,j)$ is true. ◀

Observation 10 and Lemma 11 and 14 show how we can determine whether a valid sequence of cells is feasible in $F_\rho(q,s)$ using intervals defined by s and ρ for the vertices of q and the forward and backward numbers $f_i(q)$ and $b_i(q)$.

7 Data Structure

In this section, we present two data structures solving the Fréchet queries problem. We start with some assumptions, that can be made for the time series. Let $s = \langle s_1, \dots, s_t\rangle$ be a time series. Then, we assume that either $s_{2j-1} \le s_{2j} \ge s_{2j+1}$ for all $j = 2, \dots, \lfloor t/2 \rfloor$ (*M-shaped*), or $s_{2j-1} \ge s_{2j} \le s_{2j+1}$ for all $j = 2, \dots, \lfloor t/2 \rfloor$ (*W-shaped*), because if $s_{2j-1} \le s_{2j} \le s_{2j+1}$ or $s_{2j-1} \ge s_{2j} \ge s_{2j+1}$, s has the same shape as $\langle s_1, \dots, s_{2j-1}, s_{2j+1}, \dots, s_t\rangle$. Moreover, we can assume that the complexity of all time series in S is exactly t_s by simply adding dummy vertices in the end otherwise, since the value of two consecutive vertices can also be equal. In Figure 6, the time series q is W-shaped.

The query algorithm iterates over all valid sequences of cells $\mathcal{C}$. By Lemma 9, $\mathcal{C}$ is feasible in the free space diagram $F_\rho(q,s)$ if and only if the predicates induced by $\mathcal{C}$ are true. The truth assignment of all needed predicates $(P_1),(P_2),(P_3),(P_4)$ and (P_5) can be determined using intervals defined by s and ρ. Furthermore, $\mathcal{C}$ can only be feasible in $F_\rho(q,s)$ if for all $(i-1,j),(l,j)\in\mathcal{C}$ with $i\leq l$, the monotone in s predicate $(P_6(i,l,j))$ is true. By Lemma 9, we can use the forward number $f_i(q)$ in the case that $s_j\leq s_{j+1}$ (i.e., j is odd if s is M-shaped) to determine whether $(P_6(i,l,j))$ is true. We define the *forward number* $f_i(\mathcal{C})$ as the highest such number l that is needed for $\mathcal{C}$ to be feasible in $F_\rho(q,s)$. Respectively, if $s_j\geq s_{j+1}$ (i.e., j is even if s is M-shaped) for $b_i(q)$ and we define the *backward number* $b_i(\mathcal{C})$. Formally, we get

$$f_i(\mathcal{C})=\begin{cases} l\geq i, & \text{if } \exists\ (i-1,j),(l,j)\in\mathcal{C} \text{ s.t. } j \text{ is odd and } (l+1,j)\notin\mathcal{C},\\ i, & \text{otherwise}\end{cases}$$

and

$$b_i(\mathcal{C})=\begin{cases} l\geq i, & \text{if } \exists\ (i-1,j),(l,j)\in\mathcal{C} \text{ s.t. } j \text{ is even and } (l+1,j)\notin\mathcal{C},\\ i, & \text{otherwise.}\end{cases}$$

As $\mathcal{C}$ is valid there exists a unique j such that $(i-1,j),(i,j),\dots,(l,j)\in\mathcal{C}$. Hence, the numbers $f_i(\mathcal{C})$ and $b_i(\mathcal{C})$ are well-defined. Note that we do not need $f_1(\mathcal{C})$, $b_1(\mathcal{C})$, $f_{t_q}(\mathcal{C})$ and $b_{t_q}(\mathcal{C})$ because we never consider $(P_6(1,l,j))$ and $(P_6(t_q,l,j)$.

The Data structure. Let S_M be the set of stored time series that are M-shaped and S_W the set of those that are W-shaped. We will describe how S_M is stored. The time series in S_W are stored in the same way after they were mirrored at the origin. Consequently, for those the query algorithm mirrors the query time series q at the origin and is then the same as for the time series in S_M.

For all valid sequences of cells $\mathcal{C}$, we build two associated rectangle stabbing data structures storing the time series in S_M as t_q-dimensional axis-aligned rectangles. One for the case that the query time series q is M-shaped and the other one for the case that q is W-shaped. Knowing the shape of q, Observation 10 and Lemma 11 define for every $s\in S_M$ an interval for every vertex q_i of the query time series in which it must lie such that $\mathcal{C}$ can be feasible in $F_\rho(q,s)$. For a time series s, we store the Cartesian product of those t_q intervals in the associated rectangle stabbing data structure. Note that even if the complexity of the stored time series is greater than t_q, we store only a t_q-dimensional rectangle for it.

The Query Algorithm. Let q be a query time series of complexity t_q. The query algorithm starts with computing the numbers $f_1(q),\dots,f_{t_q}(q),b_1(q),\dots,b_{t_q}(q)$. For all valid sequences of cells $\mathcal{C}$, we check whether $f_i(\mathcal{C})\leq f_i(q)$ and $b_i(\mathcal{C})\leq b_i(q)$ for all i. If so, we do a query search in the rectangle stabbing data structure depending on $\mathcal{C}$ and the shape of q with the point $(q_1,q_2,\dots,q_{t_q})$ and output all time series associated with a rectangle containing this point.

▶ **Theorem 15.** *The Fréchet queries problem for constant $t_q\geq 2$ and t_s can be solved with a data structure of size $S_R(n,t_q)$ using $Q_R(n,t_q)+\mathcal{O}(k)$ query time, where k is the size of the output (without duplicates) and $S_R(n,t_q)$ denotes the size and $Q_R(n,t_q)$ the query time of a rectangle stabbing data structure that stores n rectangles of dimension t_q. In particular, there exists a data structure of size in $\mathcal{O}(n\log^{t_q-2}n)$ and query time in $\mathcal{O}(\log^{t_q-1}n+k)$ using the rectangle stabbing data structure by Afshani, Arge and Larsen [1].*

Proof. For S_M, two rectangle stabbing data structures are stored for every valid sequence of cells $\mathcal{C}$. In each, there are stored at most n axis-aligned rectangles of dimension t_q. In a valid sequence of cells, every step is either $(i,j),(i,j+1)$ (right) or $(i,j),(i+1,j)$ (upwards) and the first cell is $(1,1)$ and the last is (t_q-1,t_s-1). Therefore, a valid sequence of cells consists of t_q+t_s-4 steps and t_q-2 upwards steps. Hence, the number of valid sequence of cells is $\binom{t_q+t_s-4}{t_q-2}$. Since t_q and t_s are considered constant, this is a constant, which completes the proof of the claimed size of the data structure.[1]

Computing the numbers $f_1(q),\ldots,f_{t_q}(q),b_1(q),\ldots,b_{t_q}(q)$ can be done in $\mathcal{O}({t_q}^3\log t_q)$ time by simply computing all distances $d_F(\langle q_i,\ldots,q_k\rangle,\langle q_i-\rho,q_k+\rho\rangle)$ and $d_F(\langle q_i,\ldots,q_k\rangle,$ $\langle q_i+\rho,q_k-\rho\rangle)$. Each computation takes time in $\mathcal{O}(t_q\log t_q)$ by Alt and Godau [4]. The query time follows by the fact that for all valid sequences of cells $\mathcal{C}$ we perform at most one query search in an associated rectangle stabbing data structure.

By Observation 10, Lemma 9, 11, and 14, a sequence of cells $\mathcal{C}$ is feasible in $F_\rho(q,s)$ for an M-shaped time series $s\in S$ if and only if all vertices of q lie in the intervals defined by the induced predicates of $\mathcal{C}$ depending on s and Observation 10 and Lemma 11, and $f_i(q)\geq f_i(\mathcal{C})$ and $b_i(q)\geq b_i(\mathcal{C})$ for all $i\in\{2,\ldots,t_q-1\}$. Therefore, the correctness follows by the fact that we iterated over all valid sequences of cells and by Lemma 8. ◀

The output in Theorem 15 may contain a constant fraction of duplicates. As such it cannot be easily used for range counting. To remove duplicates, one can use standard techniques, such as hashing.

Using an orthogonal range searching data structure it is possible to store the time series as t_s-dimensional points and the query time series defines then t_s-dimensional axis-aligned rectangles.

▶ **Corollary 16.** *The Fréchet queries problem for constant t_q and $t_s>2$ can be solved with a data structure of size $S(n,t_s)$ using $Q(n,t_s)+\mathcal{O}(k)$ query time, where k is the size of the output (without duplicates) and $S(n,t_s)$ denotes the size and $Q(n,t_s)$ the query time of an orthogonal range searching data structure that stores n points in dimension t_s. In particular, there exists a data structure of size in $\mathcal{O}\left(n(\log n/\log\log n)^{t_s-1}\right)$ and query time in $\mathcal{O}(\log n(\log n/\log\log n)^{t_s-3}+k)$.*

Proof. We use a similar idea as in the proof of Theorem 15 with the difference that the time series in S are stored as t_s-dimensional points and the query time series defines t_s-dimensional axis-aligned rectangles. We build two data structures one for M-shaped query time series and one for W-shaped time series. We describe only the one for the M-shaped case here. The other one is build symmetrically. Note that in the following we exchange the role of q and s and consider $F_\rho(s,q)$ instead of $F_\rho(q,s)$. For every valid sequence of cells $\mathcal{C}$ in $F_\rho(s,q)$, we build an orthogonal range searching data structures storing the time series s where $f_j(s)\geq f_j(\mathcal{C})$ and $b_j(s)\geq b_j(\mathcal{C})$ for all j. The query algorithm computes for all valid sequences of cells $\mathcal{C}$ a rectangle $R(\mathcal{C})$ such that $\mathcal{C}$ is feasible in $F_\rho(s,q)$ if and only if $(s_1,\ldots,s_{t_s})\in R(\mathcal{C})$ and s is stored in the data structure defined by $\mathcal{C}$. The rectangle can be computed with Observation 10 and Lemma 11. The correctness follows by Observation 10 and Lemma 9, 11 and 14. The bounds for the size and query time follow in the same way as in Theorem 15 and by using the orthogonal range searching data structure by Afshani, Arge and Larsen [1]. ◀

[1] In more detail, if $t_s\geq t_q$ it holds that $\binom{t_q+t_s-4}{t_q-2}\leq\binom{2t_s}{t_q}\leq\left(\frac{2et_s}{t_q}\right)^{t_q}$ by Stirling's approximation of the factorial function.

8 Conclusions

We believe that with some modifications it is possible to solve the Fréchet queries problem also for the case where the complexity of the query time series is not given at preprocessing time within the same bounds. Further, we believe that using the orthogonal intersection searching data structure by Edelsbrunner and Maurer [12], it is possible to build a data structure of size in $\mathcal{O}(n \log^{t_q} n)$ and query time in $\mathcal{O}(\log^{t_q - 1} n)$ for the Fréchet queries problem where the distance threshold is not given at preprocessing time.

References

1 P. Afshani, L. Arge, and K.G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of the 2012 Symposuim on Computational Geometry*, pages 323–338, 2012. doi:10.1145/2261250.2261299.

2 P. Afshani and A. Driemel. On the complexity of range searching among curves. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 898–917, 2018. doi:10.1137/1.9781611975031.58.

3 H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 235–248. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-03456-5_16.

4 H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(01& 02):75–91, 1995. doi:10.1142/S0218195995000064.

5 K. Bringmann, A. Driemel, A. Nusser, and I. Psarros. Tight bounds for approximate near neighbor searching for time series under the Fréchet distance. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 517–550, 2022. doi:10.1137/1.9781611977073.25.

6 B. Chazelle. Filtering search: a new approach to query-answering. *SIAM Journal on Computing*, 15(03):703–724, 1986. doi:10.1137/0215051.

7 B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(02):200–212, 1990. doi:10.1145/77600.77614.

8 Siu-Wing Cheng and Haoqiang Huang. Solving Fréchet distance problems by algebraic geometric methods. *ArXiv*, abs/2308.14569, 2023. doi:10.48550/arXiv.2308.14569.

9 Mark de Berg, Atlas F. Cook, and Joachim Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013. doi:10.1016/j.comgeo.2012.11.006.

10 A. Driemel and I. Psarros. ANN for time series under the Fréchet distance. In *Algorithms and Data Structures*, pages 315–328, 2021. doi:10.1007/978-3-030-83508-8_23.

11 A. Driemel and F. Silvestri. Locality-sensitive hashing of curves. In *33rd International Symposium on Computational Geometry*, volume 77, pages 37:1–37:16, 2017. doi:10.4230/LIPIcs.SoCG.2017.37.

12 H. Edelsbrunner and H.A. Maurer. On the intersection of orthogonal objects. *Information Processing Letters*, 13(04):177–181, 1981. doi:10.1016/0020-0190(81)90053-3.

13 A. Filtser, O. Filtser, and M.J. Katz. Approximate nearest neighbor for curves: simple, efficient, and deterministic. *Algorithmica*, 2022. doi:10.1007/s00453-022-01080-1.

14 M. Jiang and B. Zhu Y. Xu. Protein structure-structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 06(01):51–64, 2008. doi:10.1142/s0219720008003278.

15 W. Meulemans. *Similarity measures and algorithms for cartographic schematization.* PhD thesis, Technische Universiteit Eindhoven, 2014. doi:10.6100/IR777493.

16 E. Sriraghavendra, K. Karthik, and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 461–465, 2007. doi:10.1109/ICDAR.2007.4378752.

17 K. Toohey and M. Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, 2015. doi:10.1145/2782759.2782767.

18 C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: localizing global curve-matching algorithms. In *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pages 379–388, 2006. doi:10.1109/SSDBM.2006.11.

19 T. Wylie and B. Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1372–1383, 2013. doi:10.1109/TCBB.2013.17.

20 Y. Zhu, J. Peng, H. Liu, and Z. Lan. Chapter 26 – Analysis of nonadiabatic molecular dynamics trajectories. In *Quantum Chemistry in the Age of Machine Learning*, pages 619–651. Elsevier, 2023. doi:10.1016/B978-0-323-90049-2.00013-5.

On the Online Weighted Non-Crossing Matching Problem

Joan Boyar
Department of Mathematics and Computer Science, University of Southern Denmark,
Odense, Denmark

Shahin Kamali
Department of Electrical Engineering and Computer Science, York University, Toronto, Canada

Kim S. Larsen
Department of Mathematics and Computer Science, University of Southern Denmark,
Odense, Denmark

Ali Mohammad Lavasani[1]
Department of CSSE, Concordia University, Montreal, Canada

Yaqiao Li
Department of CSSE, Concordia University, Montreal, Canada

Denis Pankratov
Department of CSSE, Concordia University, Montreal, Canada

Abstract

We introduce and study the weighted version of an online matching problem in the Euclidean plane with non-crossing constraints: $2n$ points with non-negative weights arrive online, and an algorithm can match an arriving point to one of the unmatched previously arrived points. In the vanilla model, the decision on how to match (if at all) a newly arriving point is irrevocable. The goal is to maximize the total weight of matched points under the constraint that straight-line segments corresponding to the edges of the matching do not intersect. The unweighted version of the problem was introduced in the offline setting by Atallah in 1985, and this problem became a subject of study in the online setting with and without advice in several recent papers.

We observe that deterministic online algorithms cannot guarantee a non-trivial competitive ratio for the weighted problem. We study various regimes of the problem which permit non-trivial online algorithms. In particular, when weights are restricted to the interval $[1, U]$ we give a deterministic algorithm achieving competitive ratio $\Omega\left(2^{-2\sqrt{\log U}}\right)$. We also prove that deterministic online algorithms cannot achieve competitive ratio better than $O\left(2^{-\sqrt{\log U}}\right)$. Interestingly, we establish that randomization alone suffices to achieve competitive ratio 1/3 even when there are no restrictions on the weights. Additionally, if one allows an online algorithm to revoke acceptances, then one can achieve a competitive ratio ≈ 0.2862 deterministically for arbitrary weights. We also establish a lower bound on the competitive ratio of randomized algorithms in the unweighted setting, and improve the best-known bound on advice complexity to achieve a perfect matching.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, weighted matching problem, Euclidean plane, non-crossing constraints, competitive analysis, randomized online algorithms, online algorithms with advice, online algorithms with revoking

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.16

Funding The first and third authors were supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B. Other authors were supported by NSERC Canada.

Acknowledgements The authors thank the anonymous referees for their valuable suggestions.

[1] Corresponding author

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 16; pp. 16:1–16:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We introduce and study the following problem, which we call Online Weighted Non-Crossing Matching (OWNM). Suppose $2n$ points $p_1, \ldots, p_{2n}$ in Euclidean plane arrive online one-by-one. When p_i arrives, its positive weight $w(p_i) \in \mathbb{R}_{>0}$ is revealed and an algorithm has an option of matching p_i to one of the unmatched previously revealed points, or leave p_i unmatched. In the vanilla online model, the decisions of the algorithm are irrevocable. There is a non-crossing constraint, which requires that the straight-line segments corresponding to the edges of the matching do not intersect. Assuming that the points are in general position, the goal is to design an algorithm that maximizes the weight of matched points.

The interest in geometric settings, particularly the Euclidean plane setting, for the matching problem stems from applications in image processing [14] and circuit board design [20]. In such applications, one is often required to construct a matching between various geometric shapes, such as rectangles or circles, representing vertices, using straight-line segments or, more generally, curves. Geometry enters the picture due to constraints on the edges, such as avoiding intersections among the edges, as well as avoiding edge-vertex intersections. These constraints can have a significant impact on the offline complexity of the problem, often resulting in variants of problem that are $\mathcal{NP}$-hard (see the survey by Kano and Urrutia [22]).

The unweighted version of the Non-Crossing Matching problem (i.e., when $w(p_i) = 1$ for all $i \in \{1, \ldots, 2n\}$) has been studied both in the offline setting ([8, 19]) and the online setting ([11, 31, 21, 25]). We go over the history of the problem in detail in Section 2. For now, it suffices to observe that an offline algorithm that knows the locations of all the points in advance can match all the points while satisfying the non-crossing constraint. Thus, the value of offline OPT is always $W := \sum_{i=1}^{2n} w(p_i)$. Performance of an online algorithm is measured by its competitive ratio, which for our problem corresponds to the fraction of W that the algorithm can guarantee to achieve in the worst-case.

It is relatively easy to see that when there are no restrictions on the weights of points, no deterministic online algorithm can guarantee a non-trivial competitive ratio bounded away from 0 (in particular, this is an immediate corollary of Theorem 1). We study different regimes under which the problem admits algorithms achieving non-trivial competitive ratios. Our results can be summarized as follows:

- In the Restricted OWNM, we assume that the weights of points are restricted to lie in the interval $[L, U]$ for some $L \leq U \in \mathbb{R}_{>0}$ that are known to the algorithm at the beginning of the execution. Note that by scaling, we can assume that $L = 1$; thus, without loss of generality, we assume that all the weights are in the interval $[1, U]$ in Restricted OWNM. We show that the competitive ratio of any deterministic online algorithm is $O\left(2^{-\sqrt{\log U}}\right)$ (Theorem 1). We also present a deterministic online algorithm, Wait-and-Match (Wam), which has competitive ratio $\Omega\left(2^{-2\sqrt{\log U}}\right)$ (Theorem 5).
- We show, perhaps surprisingly, that randomization alone is enough to guarantee a constant competitive ratio for arbitrary weights. We present a simple randomized online algorithm, called Tree-Guided-Matching (Tgm), and prove that it has competitive ratio 1/3 (Theorem 7). We supplement this result by showing that no randomized online algorithm can achieve a competitive ratio better than 16/17, even for the unweighted version of the problem (Theorem 6).[2]

[2] Sajadpour [31] gave a proof that no randomized algorithm can achieve a competitive ratio better than 0.9262, which is stronger than our result $16/17 \approx 0.9411$. However, their argument has not been peer-reviewed at the time of this paper. Moreover, our argument is much simpler and shorter.

- We show that allowing revocable acceptances (see beginning of Section 6 for the definition of the model) permits one to obtain competitive ratio ≈ 0.2862 by a deterministic algorithm even when the weights of points are unrestricted (Theorem 10). We supplement this result by showing that no deterministic algorithm with revoking can achieve competitive ratio better than $2/3$ (Theorem 8).
- Lastly, we present a new algorithm, called Split-And-Match (Sam), that uses $\lceil \log C_n \rceil < 2n$ bits of advice (see beginning of Section 7 for the definition of the model) to achieve optimality (Theorem 12), where C_n is the n^{th} Catalan number. This improves upon the previously known bound of $3n$ on the advice complexity of the problem [25]. Since Sam achieves a perfect matching, it does not matter whether the given points are weighted or not.

2 Related Work

Given $2n$ points in $\mathbb{R}^2$ in general position, the basic non-crossing matching (NM) problem is to find a non-crossing matching with the largest possible number of edges. Observe that the minimum-length Euclidean matching is non-crossing, hence a perfect NM always exists. The NM problem and its variants have been extensively studied in the offline setting. Hershberger and Suri [19] gave an algorithm that finds a perfect NM in time $\Theta(n \log n)$. Atallah [8], and Dumitrescu and Steiger [15] gave efficient algorithms for the bichromatic version of the problem, where the points are divided into two subsets, and matching edges can only be formed between the two subsets. Other versions of the problem considered in the research literature include requiring the NM to be stable [30, 18], requiring two NMs to be compatible (edges in two NMs are also non-crossing, only sharing endpoints) [2], and requiring compatible NMs to satisfy certain diversity constraint [24].

Several studies considered optimization problems over all NMs. The objective functions include maximizing the sum of the Euclidean length of matching edges [5], minimizing the length of the longest matching edge [1], and other similar combinations of min and max [23].

Another line of research is to relax the non-crossing constraint and allow certain crossings. An important problem is to understand the size of a crossing family, that is, matching edges that are pairwise crossing. A recent breakthrough by Pach et al. [26] showed that the largest crossing family has linear size. Aichholzer et al. [4] studied the counting problem of k-crossing matchings.

At least two works [10, 32] considered weighted NM on n points, where every point has weight in $\{1, 2, \ldots, n\}$. Balogh et al. [10] considered the weight of an edge to be the sum of the weights of the two endpoints modulo n, and studied the typical size of NM with distinct edge weights (this is called non-crossing harmonic matching). Sakai and Urrutia [32] considered the weight of an edge to be the minimum weight of the two endpoints, and they studied the lower and upper bounds of the maximum weighted NM.

In pure mathematics, NM has been studied as a tool to understand the representation theory of groups [7, 27]. A tuple of NMs (so-called a necklace) satisfying a specific property is used to study the topology of harmonic algebraic curves associated with a polynomial over $\mathbb{C}$ [33]. Extremal graph problems where NM of size k plays the role of a forbidden subgraph are studied in [3, 17].

Besides the application in image processing and circuit design, as mentioned in the Introduction, NM has also found other applications. One major application is in computational biology. A restricted version of NM (e.g., points all on a circle), and k-non-crossing matching (no k edges pairwise intersecting, which reduces to the standard NM when $k = 2$) have been studied to understand RNA structures [9, 13, 34]. In applications that are related to visibility problems (such as in robotics) and geometric shape matching, one replaces all or a subset of points in question by geometric objects [28, 6]. For example, when the question is to match objects to objects, then an edge (p, q) between two objects A and B can be formed by choosing arbitrary points p from A and q from B, conditioned on that the edge (p, q) does not cross other objects.

The online NM has only been studied very recently. Bose et al. [11] initiated the study of online (unweighted) NM and showed that the competitive ratio of deterministic algorithms is $2/3$, while Kamali et al. [21] gave a randomized algorithm that matches in expectation about 0.6695 fraction of all points. The online bichromatic NM has also been studied in [11, 31]. Finally, the advice complexity was studied in [11, 25]. In particular, Lavasani and Pankratov [25] resolved the advice complexity of solving online bichromatic NM optimally on a circle and gave a lower bound of $n/3 - 1$ and an upper bound of $3n$ on the advice complexity of online NM on a plane.

3 Preliminaries

The input to the matching problems considered in this work is an online sequence $I = (p_1, \ldots, p_{2n})$ of points in general position, where p_i has a positive real-valued weight $w(p_i) \in \mathbb{R}_{>0}$. We use W to denote the total weight of all the points, i.e., $W = \sum_{i=1}^{2n} w(p_i)$. For the Restricted OWNM, the weights are assumed to lie in the interval $[1, U]$ for some known value of U, which is considered to be a hyper-parameter, and not part of the input. Upon the arrival of p_i, an online algorithm must either leave it unmatched or match it with an unmatched point p_j $(j < i)$, in which case the line segment between p_i and p_j, denoted by $\overline{p_i p_j}$, must not cross the line segments between previously matched pairs of points. The objective is to maximize the total weight of matched points. For an online algorithm ALG (respectively, offline optimal algorithm OPT), we use ALG(I) (respectively, OPT(I)) to denote the total weight of points matched by the algorithm on input I. By abuse of notation, the symbol $\overline{pq}$ is also used to denote the full line passing through the two points p and q, dividing a convex region into two sub-regions.

We say that a deterministic online algorithm ALG is ρ-competitive if there exists a constant c such that for every input sequence I we have

$$\mathrm{ALG}(I) \geq \rho \cdot \mathrm{OPT}(I) - c.$$

For a randomized ALG the above inequality is replaced by the following

$$\mathbb{E}(\mathrm{ALG}(I)) \geq \rho \cdot \mathrm{OPT}(I) - c.$$

If $c = 0$ then we call the competitive ratio ρ strict, and we say that ALG is strictly ρ-competitive. If $c \neq 0$ then, for emphasis, we shall sometimes say that the competitive ratio is asymptotic. Note that for the Restricted OWNM, we allow c to depend on the hyper-parameter U when considering asymptotic competitiveness. Thus, an algorithm achieving asymptotic competitive ratio ρ is allowed to leave a constant number of points unmatched (regardless of their weights) beyond the $(1 - \rho)$-fraction of W.

4 Deterministic Algorithms for Restricted OWNM

4.1 Point Classification

In both lower and upper-bound arguments, we use a point classification, based on parameters, $k \in \mathbb{N}$ and $U \in \mathbb{R}$, which we explain here. Let $k = \lceil \sqrt{\log U} \rceil$, and define values of $a_0, a_1, \ldots, a_k$ so that

$$a_0 = 1, a_k = U, \quad r = a_1/a_0 = a_2/a_1 = \ldots = a_k/a_{k-1},$$

which implies that $r = U^{1/k}$ and $a_i = r^i$. For a given value $w \in [1, U]$, define $\llbracket w \rrbracket$ as the largest a_i such that $a_i \leq w$. In what follows, a point with weight w is said to have type i if $\llbracket w \rrbracket = a_i$. Thus, there are $k+1$ distinct types, with type k containing only the value U. The type of a line segment between two matched points x and y is defined by the type of the end-point with larger weight, that is, $\overline{xy}$ has type i if one of its endpoints has type i and the other endpoint has type at most i.

4.2 Negative Result

▶ **Theorem 1.** *For a sufficiently large value of U, the asymptotic competitive ratio of any deterministic online algorithm for the Restricted OWNM problem is $O\left(2^{-\sqrt{\log U}}\right)$.*

Proof. Let ALG be any online deterministic algorithm. We use an adversarial argument. The adversary sends all points on a circle C, so any match the algorithm makes creates a chord in the circle, dividing a previous region into two. At any point in time, the adversary sends a point in an *active region* of C, which is formed by one or two *arcs*, the segments of the circle bounded by two consecutive points, in the boundary of C. Initially, the entire circle forms the active region. The adversary's strategy is to maintain a mapping from unmatched points to matched points to ensure the ratio between the total weight of matched points and unmatched points is $O\left(2^{-\sqrt{\log U}}\right)$. Note that this implies the ratio between the total weight of matched points and all points is also $O\left(2^{-\sqrt{\log U}}\right)$.

The adversary starts the input with an arbitrarily large number, m (this is required to guarantee that our bound is asymptotic). The adversary puts points of weight 1 in arbitrary positions on the circle until either the algorithm matches m pairs of points or it reaches $m2^k$ points on the circle. In the latter case, the competitive ratio is at most $O(2^{-k}) = O(2^{-\sqrt{\log U}})$.

Therefore, we may assume that ALG eventually matches m pairs of points, creating non-intersecting chords, and $m+1$ regions. Now, make each matched pair responsible for a distinct region created, though with the first matched pair being responsible for two regions, initially the first two regions. Suppose a new chord $\overline{xy}$ divides region R into two. Let $\{x_R, y_R\}$ be the responsible pair for R, R_1 be the side of R that has $\overline{x_R y_R}$ on its boundary and R_2 be the other side. Leave $\{x_R, y_R\}$ responsible for R_1 and make $\{x, y\}$ responsible for R_2. This ensures that each matched pair is responsible for at least one region.

For each region R, the adversary makes R the active region, runs the following procedure and continues with the next region until it covers all the regions. Let $\{x_R, y_R\}$ be the responsible pair of points for R. Consider the following two cases, depending on the number of unmatched points in R:

Case 1. If the number of unmatched points in R is $\geq 2^k - 1$, the adversary does not send any point in R and continues to the next region. In this case, we map the unmatched points in R to the matched pair $\{x_R, y_R\}$. Note that $2^k - 1$ points of weight 1 are mapped to a segment of total weight 2. The ratio between the weight of matched points to the unmatched points will be $\leq 2/(2^k - 1) \in O\left(2^{-\sqrt{\log U}}\right)$.

Case 2. If the number of unmatched points in R is $< 2^k - 1$, the adversary plans to send a sequence of points, $P = (p_1, p_2, \ldots, p_k)$, with weights $a_1, a_2, \ldots, a_k$ (respectively), one point from each weight, in the ascending order of their weights, in the following manner, (see Fig. 1). The point p_1 of weight a_1 appears in an arbitrary position in R (on the circle). Upon

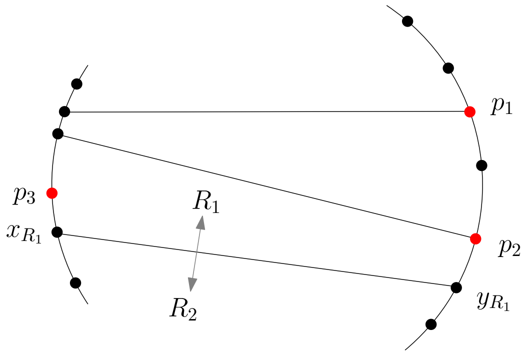


Figure 1 An illustration of the adversary's strategy for $k = 3$. The two arcs form the active region. Black points have weight 1. Suppose in the first phase ALG matched x_{R_1} and y_{R_1}, which became responsible for R_1 region. Note that the number of unmatched points (of weight 1) in R_1 is 6, which is less than $2^k - 1 = 7$. Thus, in the second phase, the adversary plans to send points p_1, p_2, p_3 of weights a_1, a_2, a_3 in R_1. Suppose ALG matches the point of weight a_1; then the adversary sends p_2, p_3 below the line segment between the matched pair (there are fewer unmatched points there). Similarly, after the point p_2 of weight a_2 is matched, the adversary sends p_3 to the side of the resulting segment with no unmatched points. This ensures that some point of weight a_i (here a_3) stays unmatched and is mapped to the matched pairs.

the arrival of a point p_i with weight a_i ($i \in \{1, \dots, k\}$), either ALG matches it with a point of weight 1 or leaves it unmatched. In the latter case, the adversary does not send more points in R and continues with the next region.

In the former case, when ALG matches a point p_i of weight a_i with a point q of weight 1, make the side of $\overline{p_i q}$ that contains at most half of the unmatched points, the active region. The adversary continues putting the remaining points of P in the active region. Thus the unmatched points on the opposite side of $\overline{p_i q}$ stay unmatched, since $\overline{p_i q}$ is between the new point and those unmatched points.

Therefore, after matching p_i and q, the number of unmatched points of weight 1 that can match with future points in P decreases by a factor of at least 2. Let p_j be the first point in P that the algorithm leaves unmatched. Given that the adversary can send up to k points, and there are initially less than $2^k - 1$ unmatched points in R, there exists such p_j of weight a_j. At this point, the adversary ends the procedure for R and continues with the next region.

The total weight of points in matched pairs in R before the arrival of p_j is:

$$M = \underbrace{2}_{\text{for}(x_R, y_R)} + \underbrace{j-1}_{\text{endpoints of weight 1}} + \underbrace{a_1 + a_2 + \dots + a_{j-1}}_{\text{endpoints with weight } a_i} \leq 2\frac{a_j - 1}{r - 1}.$$

Given that the unmatched point p_j is of weight a_j, the ratio between the weight of matched points and unmatched points is at most $M/a_j \in O(1/r) = O\left(2^{-\sqrt{\log U}}\right)$.

Given that each matched pair is responsible for at least one region, the above procedure creates a mapping of matched points to unmatched points with a weight ratio of $O(2^{-\sqrt{\log U}})$ in all cases, as desired. This finishes the proof. ◀

4.3 Positive Result: The Wait-and-Match Algorithm

We propose an algorithm called "Wait-and-Match" (WAM). Assume the points appear in a bounding box B. Throughout its execution, WAM maintains a "convex partitioning" of B. Initially, there is only one region formed by the entire B. As we will describe, the algorithm

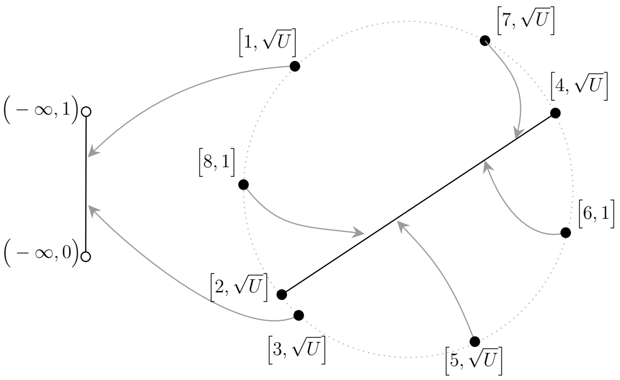


Figure 2 An illustration of the mapping used to analyze Wam. In this example, we have $k = 2$ and 8 points with weights in $\{1, \sqrt{U}, U\}$. Here, $[t, w]$ indicates the t^{th} point in the input sequence having weight w. Note that points 1 and 3 are mapped to the segment corresponding to the imaginary points $(-\infty, 0)$ and $(-\infty, 1)$ of weight U.

matches two points only if they appear in the same convex region. Whenever two points in a convex region R are matched, the line segment between them is extended until it hits the boundary of R, which results in partitioning R into two smaller convex regions. We use the same point classification as defined in Section 4.1.

Suppose a new point p appears, and let R denote the convex region of p. In deciding which point to match p to (if any), the algorithm considers all unmatched points in R in the non-increasing order of their weights. Let q be the next point being considered, and let i be the maximum of the type of p and the type of q. The algorithm matches p with q if there are at least $2^{k-i} - 1$ unmatched points on each side of $\overline{pq}$. If all points in R are examined, and no suitable q exists, p is left unmatched.

Example. Suppose $k = 2$. Then $a_0 = 1, a_1 = \sqrt{U}$, and $a_2 = U$. Let p be a point with weight 1. Upon the arrival of a point p, the algorithm matches p with any point q of weight U when there are at least $2^{2-2} - 1 = 0$ points on each side of $\overline{pq}$. That is, if there is an unmatched point of weight U in the region, the algorithm would match p to it unconditionally. Similarly, if there are no unmatched points of weight U in the region, the algorithm tries to match p with any point q of weight $[a_1 = \sqrt{U}, a_2 = U)$ provided there is at least $2^{2-1} - 1 = 1$ point on each side of $\overline{pq}$. Finally, if previous scenarios do not occur, the algorithm tries to match p with any point q of weight $[a_0 = 1, a_1 = \sqrt{U})$ provided there are at least $2^{2-0} - 1 = 3$ unmatched points on each side of $\overline{pq}$. This will happen if there were at least 7 unmatched points in the region.

To analyze the algorithm, we match each unmatched point into a matched pair. For the sake of analysis, we introduce two "imaginary" points $(-\infty, 0)$ and $(-\infty, 1)$ of weight U and treat them as if they were matched before the input sequence is revealed. Suppose a new point, p, arrives in a region R that is not matched. In this case, we map p to the most recent segment that forms a boundary of the region R. See Figure 2 for an illustration of this mapping.

▶ **Lemma 2.** *Every point of type i is mapped to a segment of type $j \geq i$.*

Proof. For the sake of contradiction, suppose a point p with type i arrives in the region R and gets mapped to $\overline{p_R, q_R}$ of type $\leq i - 1$.

Without loss of generality, assume p_R arrived after q_R. By the definition of the algorithm, at the time p_R appeared, there were at least $2^{k-i+1}-1$ unmatched points in R (otherwise, p_R would not have been matched with q_R). These unmatched points are still unmatched when p appeared (otherwise, R should have been partitioned, and p should have been mapped to some other segment). Thus, when p appeared, the algorithm could match it with the point that bisects these unmatched points, and there would be at least $(2^{k-i+1}-2)/2 = 2^{k-i}-1$ points on each side of the resulting line segment. This contradicts the fact that the algorithm left p unmatched. ◀

▶ **Lemma 3.** *Let s be any line segment between two matched points. For any i, at most $2^{k-i+2}-2$ unmatched points of type i are mapped to s.*

Proof. For the sake of contradiction, assume at least $2^{k-i+2}-1$ points of type i are mapped to s. Then, there must be at least $\lceil (2^{k-i+2}-1)/2 \rceil = 2^{k-i+1}$ points of type i in a convex region R formed by extending s. At the time the last of these points, say p, arrives, it could be matched to the point q that bisects the other points; there will be at least $(2^{k-i+1}-2)/2 = 2^{k-i}-1$ points on each side of $\overline{pq}$. Since $\overline{pq}$ is of type i, the algorithm must have matched p with q, which contradicts the fact that p and q are unmatched and mapped to s. ◀

▶ **Lemma 4.** *Assuming U is sufficiently large, the total weight of unmatched points mapped to a segment of type j is at most $a_{j+1}2^{k-j+3}$.*

Proof. Note that a point of type i has weight at most $a_{i+1} = r^{i+1}$. Hence, by Lemma 2 and Lemma 3, the total weight of unmatched points mapped to a segment of type j is at most

$$\sum_{i=0}^{j} r^{i+1}2^{k-i+2} = 2^{k+2}r\sum_{i=0}^{j}\left(\frac{r}{2}\right)^{i} = 2^{k+2}r\frac{(r/2)^{j+1}-1}{r/2-1} \le a_{j+1}2^{k-j+3}. \qquad ◀$$

Here, we assumed that $r \ge 4$, which holds for a sufficiently large U.

▶ **Theorem 5.** *The competitive ratio of the deterministic online algorithm Wam for the Restricted OWNM problem is $\Omega\left(2^{-2\sqrt{\log U}}\right)$.*

Proof. For every matched pair $\overline{pq}$ by Wam consider the set of points formed by p, q, and the unmatched points mapped to them. By Lemma 4, if $\overline{pq}$ has type j, the ratio of the weight of the matched pair over all the points in this set is at least $\frac{a_j}{2a_j+a_{j+1}2^{k-j+3}} \ge \frac{1}{r2^{k+4}}$.

Since the algorithm Wam guarantees that every unmatched point is mapped to some matched pair, the competitive ratio of Wam is at least $2^{-(2k+4)}$, where we used $k = \lceil \sqrt{\log U} \rceil$ and $r = U^{1/k} = 2^{(\log U)/k} \le 2^k$. ◀

5 Randomized Algorithms

5.1 Negative Result

To bound the competitive ratio of randomized algorithms, we will use Yao's minimax principle. We create a randomized unweighted input similar to what Lavasani and Pankratov [25] used for the advice model. We prove an upper bound on the competitive ratio of every deterministic algorithm on this input, and this gives us an upper bound for randomized algorithms in the adversarial setting. We consider a circle and generate points on the circumference of this circle. For a point p, let the left and the right arcs of p be the clockwise and counter-clockwise arcs that are bounded by p.

Put p_1 and p_2 on two arbitrary antipodals of the circle, creating two arcs. Make p_2 the current *active* point. At each step, we choose one of the arcs of the current active point randomly and then we put the next active point on that arc. To deceive the algorithm, sometimes we generate a *fake* point on one of the arcs of the active and then put the next active point on the other arc.

Consider two sequences $L_1, \dots, L_{2n}$ and $F_1, \dots, F_{2n}$ of Bernoulli i.i.d. random variables with parameter $1/2$. Iterate the following procedure to make $2n$ points. Let p_i be the current active point, L_i determines the position of p_{i+1}. If L_i is 1, put p_{i+1} in the middle of the left arc of p_i, and if L_i is 0, put it in the middle of the right arc of p_i.

Given p_i is an active point, if F_{i+1} is 1, the point p_{i+1} becomes fake point. Make p_{i+2} the next active point and put it in the middle of the other arc of p_i (e.g. if p_{i+1} is on the left arc of p_i, put p_{i+1} on the right arc of p_i). If F_{i+1} is 0, make p_{i+1} the new active point. Continue the procedure with the new active point.

▶ **Theorem 6.** *No randomized online algorithm can achieve a competitive ratio better than* $16/17$ *in expectation.*

Proof. We aim to bound the competitive ratio of any deterministic algorithm on the described input sequence. Fix a deterministic algorithm ALG. Segments of matched points by ALG divide the circle into convex regions. If an unmatched point is in a region that no new points arrive in, it cannot be matched anymore and we call it an *isolated* point. Given that ALG matches p_i upon its arrival, let X_i be the indicator random variable that p_i is an active point, and matching it causes at least one point to become isolated. Let A_i be the indicator random variable that p_i becomes an active point. For $i \geq 3$, p_i is a fake point if and only if p_{i-1} was an active point and F_i is 1. Thus we can write A_i as $1 - A_{i-1}F_i$.

Suppose p_i is an active point that arrives in a convex region R, that ALG matches upon its arrival, splitting R into R_L and R_R, which contain the left and right arcs of p_i, respectively. If R_L and R_R are both empty, meaning they do not contain any unmatched point, p_{i+1} becomes isolated if it is a fake point. If R_L and R_R are both non-empty and the point p_{i+1} becomes an active point, then the unmatched points of the opposite side of p_{i+1} become isolated. Now suppose R_L is empty and R_R is not empty and p_{i+1} arrives on the left arc of p_i. If p_{i+1} is a fake point, it becomes isolated, and if it is the new active point, unmatched points in R_R become isolated. Similarly, if R_R is empty and R_L is not empty and p_{i+1} arrives in the right arc of p_i, the segment $\overline{p_i p_j}$ creates isolated points. If $i = 2n$, there is no p_{i+1}, and matching p_i makes points isolated if R_L or R_R are not empty. Since we are interested in the asymptotic competitive ratio we can ignore this case. Therefore, given ALG matches p_i we can write X_i as follows.

$$X_i = \begin{cases} A_i F_{i+1} & \text{if } R_L \text{ and } R_R \text{ are empty} \\ A_i L_i & \text{if } R_L \text{ is empty and } R_R \text{ is not} \\ A_i(1 - L_i) & \text{if } R_R \text{ is empty and } R_L \text{ is not} \\ A_i(1 - F_{i+1}) & \text{if } R_L \text{ and } R_R \text{ are not empty} \end{cases}$$

Let the random variable M be the size of the matching made by ALG, and for each $1 \leq i \leq M$, let T_i be the step number in which ALG makes the i^{th} match. Thus, the algorithm is guaranteed to have at least $\sum_{i=1}^{M} X_{T_i}$ unmatched points at the end of the execution. In order to bound the expectation of M, it may be beneficial to view it in the context of the following game. Suppose that ALG has a budget of $2n$ points. The game proceeds in rounds. In round j the algorithm pays 2 points from the budget to make a guess (this corresponds to a pair of points getting matched) of a Bernoulli random variable outcome (which corresponds to ALG's match either resulting in an isolated point or not). If the guess is correct (this corresponds to $X_{T_j} = 0$, no isolated points are guaranteed to be created), then the algorithm does not pay any more points for this round. If the guess is incorrect (this corresponds to $X_{T_j} = 1$), then the algorithm pays one more point from the budget. ALG tries to maximize the total number of rounds before the budget is exhausted. Thus, in round j, the algorithm uses $X_{T_j} + 2$ points from the budget. Overall, M is the largest integer such that $\sum_{j=1}^{M}(X_{T_j} + 2) \leq 2n$. If X_{T_j} were i.i.d., we could use the renewal theorem to bound $\mathbb{E}(M)$. The issue is that X_{T_j} are not i.i.d., because X_i depends on A_i and F_{i+1}; thus there are correlations between X_i and X_{i+1}. The idea is to lower bound the expression $\sum_{j=1}^{M}(X_{T_j} + 2)$ by the sum of some i.i.d. random variables Z_i, compute the corresponding value of M' for the Z_i, and then relate it back to the value of M.

Now we define an auxiliary random variable sequence $Y_1, \ldots, Y_M$ as follows:

$$Y_i = \begin{cases} (1 - F_{T_i})F_{T_i+1} & \text{if } R_L \text{ and } R_R \text{ are empty} \\ (1 - F_{T_i})L_{T_i} & \text{if } R_L \text{ is empty and } R_R \text{ is not} \\ (1 - F_{T_i})(1 - L_{T_i}) & \text{if } R_R \text{ is empty and } R_L \text{ is not} \\ (1 - F_{T_i})(1 - F_{T_i+1}) & \text{if } R_L \text{ and } R_R \text{ are not empty} \end{cases}$$

By replacing A_i with $1 - A_{i-1}F_i$, we can see $Y_i \leq X_{T_i}$. Note that $Y_2, Y_4, \ldots, Y_{2\lfloor \frac{M}{2} \rfloor}$ are i.i.d. Bernoulli random variables with parameter 1/4. Thus for every $m \leq M$, we can bound $\sum_{j=1}^{m}(X_{T_j} + 2)$ as follows:

$$\sum_{j=1}^{m}(X_{T_j} + 2) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(X_{T_{2j-1}} + X_{T_{2j}} + 4) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(Y_{2j-1} + Y_{2j} + 4) \geq \sum_{j=1}^{\lfloor m/2 \rfloor}(Y_{2j} + 4)$$

Let us define yet another auxiliary random variable sequence $Z_1, Z_2, \ldots$ as follows. For $1 \leq i \leq \lfloor \frac{M}{2} \rfloor$, let $Z_i = 4 + Y_{2i}$ and for $i > \lfloor \frac{M}{2} \rfloor$ let $Z_i = 4 + Y_i'$ such that Y_i's are i.i.d. Bernoulli random variables with parameter 1/4. This makes the Z_i i.i.d. random variables that take on values of either 4 or 5 with probability 1/4 and 3/4, respectively.

Let the random variable M' be the maximum m such that $\sum_{i=1}^{m} Z_i < 2n$. Note that $\sum_{i=1}^{\lfloor M/2 \rfloor} Z_i = \sum_{i=1}^{\lfloor M/2 \rfloor}(Y_{T_j} + 4) \leq \sum_{i=1}^{M}(X_{T_j} + 2) \leq 2n$. Therefore $M' \geq \lfloor M/2 \rfloor$. Since the Z_i's are i.i.d. and $\mathbb{E}(Z_i) = 17/4$, by the renewal theorem $\mathbb{E}(M') = 8n/17$ and therefore $\mathbb{E}(M)$ is at most $16n/17$. By Yao's minimax principle, this shows an upper bound of 16/17 on the competitive ratio of randomized algorithms in the adversarial model. ◀

5.2 Positive Result: Tree-Guided-Matching Algorithm

We propose a randomized algorithm called "Tree-Guided-Matching" (TGM) that has the following uniform guarantee, regardless of the weights of the points: each point appears in a matching with probability at least 1/3.

The algorithm TGM uses a binary tree to guide its matching decisions. The binary tree is created online, with each node of the tree corresponding to an online point. Intuitively, the binary tree, as it grows, gives an online refining of the partition of the plane into convex regions, such that for each region there is some online point *responsible* for it. Initially, set p_1 as the root of the tree and p_2 the child of p_1. By an abuse of notation, we also use $\overline{pq}$ to denote the straight line determined by points p and q. Let R_1 and R_2 denote the two regions corresponding to the half-spaces created by $\overline{p_1 p_2}$. Let p_2 be responsible for both R_1 and R_2. In general, when p_i arrives into a region R for which p_j is responsible (of course, $j < i$), make p_i a child of p_j in the binary tree. The line $\overline{p_i p_j}$ divides the region R into two sub-regions R'

and R'', let p_i be responsible for both of them, and at this point the responsibility of p_j on R is lost as region R has been refined to R' and R''. Note that this implies every node of the tree has at most two children. Next, we describe how TGM chooses to match points. At the beginning, TGM matches p_2 with p_1 with probability $1/3$. After that, upon the arrival of p_i, let p_j be its parent in the tree. If p_j is unmatched and p_i is its first child, match p_i to p_j with probability $1/2$. If p_j is unmatched and p_i is its second child, match p_i to p_j deterministically. Note that TGM only tries to match an online point with its parent in the tree.

▶ **Theorem 7.** *Every point, regardless of its weight, is chosen into a matching by the randomized algorithm* TGM *with probability at least* $1/3$. *Hence,* TGM *achieves a strict competitive ratio at least* $1/3$.

Proof. Note that since TGM only matches a child to its parent in the binary tree, the matching is non-crossing. Indeed, by our construction of the tree, every child is a point *inside*[3] a convex region for which its parent is responsible, and its parent lies on the boundary of that region. Hence, the line segment formed by them does not cross any existing line segment.

Next, we show the claimed performance of TGM. By the definition of TGM, p_1 is matched (by p_2) with probability $1/3$. We will show that every p_i, $i \geq 2$, *upon its arrival* gets matched to its parent with probability exactly $1/3$, which implies the claim. To see this, proceed inductively. The base case is true for p_2. Let p be the currently arrived point and q be its parent. We consider two cases.

- If p is the first child of q, then by the induction hypothesis q at this moment is unmatched with probability $2/3$, hence according to TGM, p is matched (to q) with probability $(2/3) \cdot (1/2) = 1/3$.
- If p is the second child of q, then q at this moment is unmatched with probability $1 - 1/3 - 1/3 = 1/3$. By TGM, p is matched (to q) with probability $(1/3) \cdot 1 = 1/3$. ◀

6 Revocable Acceptances

In this section, we consider the revocable setting. When a new point p arrives, an algorithm has an option of removing one of the existing edges from the matching prior to deciding on how to match p. The decision to remove an existing edge is irrevocable. The benefit of making this decision is that the end-points of the removed edge, along with possible points on the other side of the edge (though our positive result does not use this possibility), become available candidates to be matched with p, provided the non-crossing constraint is respected.

6.1 Negative Result

Bose et al. [11] showed that a deterministic greedy algorithm without revoking can achieve $2/3$ competitive ratio in the unweighted version. In this section, we prove that in the unweighted version, no deterministic algorithm with revoking can beat the ratio $2/3$.

▶ **Theorem 8.** *No deterministic algorithm with revoking can achieve a competitive ratio better than* $2/3$ *even in the unweighted version.*

[3] Recall that online points are in general position.

Proof. Fix a deterministic algorithm ALG, an arbitrary large n, and a circle in the plane. The adversary adds at least $2n$ points, all of weight 1, on the circle, one by one, and let ALG match them into pairs. We maintain the invariants that there is always one active region of the circle, and that for each matched pair, there is always at least one unmatched point.

Initially, the entire circle is the active region. A phase consists of the adversary presenting points on the circle, in the active region, until ALG either matches a pair or revokes a matching, or until $2n$ points have been given. The adversary stops if there are $2n$ points and the last point is unmatched. Otherwise, if a match has just occurred, there are two cases.

In Case 1, the current point, p, is simply matched to a point, q, on the circle. The chord $\overline{pq}$ divides the active region into two sub-regions, R_1 and R_2. If neither region has any points, add a point, p', to R_1. Without loss of generality, assume that R_1 contains at least as many unmatched points as R_2. If p' is matched, ALG has revoked a matching; and we get the extra point from Case 2. Otherwise, R_2 becomes the active region, some unmatched point in R_1 is associated with the matched pair, and the phase ends.

In Case 2, ALG revokes a matching and either matches the current point, p, or leaves p unmatched. Removing the one match, removes a chord of the circle, joining two regions into a new convex region. This region is the active region if p is not matched. In either case, the number of matched points is not increased. However, the number of unmatched points is increased by at least 1, since at least one of the points, q, from the revoked match is now unmatched and p is only matched to one point. If there is a new match for p, the sub-region created by the match that does not contain q becomes the active region, and q is the unmatched point associated with the new matched pair. The current phase ends.

Inductively, the invariants hold after each phase, and the unmatched point associated with each matched pair ensures that no more than 2/3 of the points are matched. Although the number of points may be odd, this gives an asymptotic lower bound of 2/3 on the competitive ratio. ◀

Note that ignoring the revoking option, the above proof is a simpler alternative to bound the competitive ratio of the deterministic algorithm which was given by Bose et al. [11].

6.2 Positive Result: Big-Improvement-Match

We present a deterministic algorithm with revoking, called "Big-Improvement-Match" (Bim). This algorithm has a strict competitive ratio of ≈ 0.2862 even when weights of points are unrestricted. This shows that while revoking does not improve the competitive ratio in the unweighted version, it provides us with an algorithm with a constant competitive ratio, which is unattainable for a deterministic algorithm without revoking.

Bim maintains a partitioning of the Euclidean space into regions. Each region in the partition is assigned an edge from the current matching to be responsible for that region. Each edge can be responsible for up to two regions. Bim starts out by matching the first two points, p_1 and p_2 regardless of their weights, dividing the plane into two half-planes by $\overline{p_1p_2}$. Bim then assigns $\overline{p_1p_2}$ to be responsible for the two half-plane regions. Next, consider a new point p_i (for $i \geq 3$) that arrives in an existing region R. Suppose that $\overline{p_jp_{j'}}$ is the responsible edge for R. If there is at least one unmatched point in R, Bim matches p_i with an unmatched point p_k in R with the highest weight. Then $\overline{p_jp_{j'}}$ is no longer responsible for R, and the region R is divided into two new regions by $\overline{p_ip_k}$. The responsibility for both new regions is assigned to $\overline{p_ip_k}$. If p_i is the only point in R, then Bim decides to revoke the matching $(p_j, p_{j'})$ or not as follows. Without loss of generality, assume $w(p_j) \leq w(p_{j'})$. If $w(p_i) < rw(p_{j'})$, then Bim leaves p_i unmatched. Otherwise, Bim removes the matching $(p_j, p_{j'})$ and matches

p_i with $p_{j'}$. We note that r is a parameter that is going to be chosen later so as to optimize the competitive ratio. If R is the only region that $\overline{p_j p_{j'}}$ was responsible for when p_i arrived, then R is divided into two regions by $\overline{p_i p_{j'}}$, and $\overline{p_i p_{j'}}$ becomes responsible for the two new regions. (The regions on the other side of $\overline{p_j p_{j'}}$ from p_i keep their boundaries, even though $(p_j, p_{j'})$ is no longer in the matching.) Otherwise $\overline{p_j p_{j'}}$ was responsible for R' in addition to R when p_i arrived. In this case, after removal of the match $(p_j, p_{j'})$, regions R and R' are merged to give region $R'' = R \cup R'$, and R'' is divided by $\overline{p_i p_{j'}}$ into two regions, and Bim makes $\overline{p_i p_{j'}}$ responsible for both new regions.

▶ **Proposition 9.** *The following observations concerning Bim hold:*
1. *All responsible edges are defined by two currently matched points.*
2. *Each edge is responsible for at most two regions.*
3. *All regions are convex.*
4. *When a matched edge $(p_j, p_{j'})$ is replaced due to the arrival of a point p_i in region R, then edge $(p_i, p_{j'})$ is contained in R.*

Proof. (1) follows since an edge only becomes responsible when its endpoints become matched. When another edge becomes responsible for a region, the original edge is no longer responsible. (2) follows since the only two regions an edge is made responsible for are the two regions created when the endpoints of the edge were matched. When two points in one of the regions an edge is responsible for are matched, the edge is no longer responsible for that region, but will still be responsible for one region if it had been responsible for two up until that point. (3) follows inductively, since separating two convex regions by a line segment creates two convex regions. In addition, when Bim removes an edge, that edge was the last matching created in either of the two regions it was responsible for. (4) follows by (3). ◀

▶ **Theorem 10.** *Bim with $r \in (1, \sqrt{2}]$ has strict competitive ratio at least $\min\left(\frac{r^2-1}{r^3}, \frac{1}{1+2r}\right)$ for the OWNM with arbitrary weights.*

Proof. We consider for each region an edge is responsible for, the total weight of unmatched points in that region. These points come in two flavours: those that were matched at some point during the execution, but due to revoking became unmatched, and those that were never matched during the entire execution of the algorithm.

Consider any subsequence of all created edges, $\langle e_1, \ldots, e_k \rangle$, where e_1 was created when a second unmatched point arrived in some region, and the possible remaining edges were created via revokings, i.e., e_i caused e_{i-1} to be revoked for $2 \le i \le k$, and e_k is in Bim's final matching. Let $e_j = (p_{i_j}, p_{i_{j+1}})$ and $w(p_{i_j}) \le w(p_{i_{j+1}})$, so $e_{j+1} = (p_{i_{j+1}}, p_{i_{j+2}})$. Thus, for $3 \le j \le k+1$, p_{i_j} arrived after $p_{i_{j-1}}$. Every pair ever matched by Bim is included in some such sequence of edges. The points, $p_{i_1}, \ldots, p_{i_{k-1}}$ could be unmatched points in a region for which e_k is responsible.

Let $\alpha = w(p_{i_k})$, so for every $2 \le j \le k$, $w(p_{i_j})$ is at most $\alpha r^{-(k-j)}$ and $w(p_{i_1}) \le w(p_{i_2}) \le \alpha r^{-(k-2)}$. Let $\beta = w(p_{i_{k+1}})$. The total weight of points in this sequence is

$$\begin{aligned}\sum_{j=1}^{k+1} w(p_{i_j}) = w(p_{i_1}) + \sum_{j=2}^{k} w(p_{i_j}) + w(p_{i_{k+1}}) &\le \alpha r^{-(k-2)} + \alpha\left(\frac{r}{r-1}\right)(1 - r^{-(k-1)}) + \beta \\ &= \alpha\left(r^{-(k-1)}\frac{r(r-2)}{r-1} + \frac{r}{r-1}\right) + \beta.\end{aligned}$$

Now, we consider other points that were never matched, but were at some time in a region for which one of the e_j was responsible. After e_1 is created and before e_2, a first point q_1 could arrive in one of the regions for which e_1 is responsible. Note that q_1 is not matched if $w(q_1) < rw(p_{i_2})$. (Note that a second point arriving in that region will then be matched to q_1, dividing the region, and the sub-regions will not be considered part of the region for which e_k eventually becomes responsible.) Now, suppose that another point, q_2, arrives between when e_j and e_{j+1} are created for some $2 \leq j < k$, remaining unmatched in one of the regions for which e_k is responsible. Then, neither q_2 nor $p_{i_{j+2}}$ is in the same region as $p_{i_{j-1}}$ or one of them would have been matched to $p_{i_{j-1}}$ (or $p_{i_{j-1}}$ was already matched and the region divided). By Proposition 9.2, e_i is responsible for at most two regions, so $p_{i_{j+2}}$ arrives in the same region as q_2, while unmatched. This is a contradiction, since Bim would match them. Thus, other than q_1, the only never-matched point, q_2, in a region for which e_k is responsible, arrives after e_k and $w(q_2) < rw(p_{i_{k+1}})$.

Then, for $k \geq 2$, the total weight of unmatched points for which e_k is responsible is at most $\left(r^{-(k-3)} + r^{-(k-1)}\frac{r(r-2)}{r-1} + \frac{r}{r-1}\right)\alpha + (1+r)\beta$. If $r \leq \sqrt{2}$, then $r^{-(k-3)} + r^{-(k-1)}\frac{r(r-2)}{r-1}$ is at most zero and we can bound the total weight when $k \geq 2$ by: $(\frac{r}{r-1})\alpha + (1+r)\beta$. Thus, the ratio between the weight of matched points in sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_{k+1}}$ and the total weight of all points associated with this sequence for $k \geq 2$ is at least $\frac{\alpha+\beta}{(\frac{r}{r-1})\alpha+(1+r)\beta}$. Since $\frac{r}{r-1} > 1 + r$, for $1 < r \leq \sqrt{2}$, this ratio is minimized when β is minimized, which happens at $\beta = r\alpha$. Thus, the competitive ratio for $k \geq 2$ is at least $(1 + r)/(\frac{r}{r-1} + r(1 + r)) = \frac{r^2-1}{r^3}$.

Now, consider the case of $k = 1$, and let α and β have the same meaning as above. Then the sequence $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ consists of a single edge. Thus, the weight of the matched points is $\alpha + \beta$, and there could be two unmatched points q_1 and q_2 at the end of the execution of the algorithm charged to this edge. We have $w(q_i) < r\beta$, so the ratio between the weight of matched points and the total weight of all points associated with the sequence in case of $k = 1$ is at least $\frac{\alpha+\beta}{\alpha+(1+2r)\beta}$. Observe that this ratio is minimized when β goes to infinity and becomes $1/(1 + 2r)$.

Taking the worse ratio between the above two scenarios proves the statement of the theorem. ◀

▶ **Corollary 11.** *With the choice of parameter for* Bim*, r^*, defined as the positive solution to the equation $\frac{1}{1+2r} = \frac{r^2-1}{r^3}$, approximately* 1.2470*, we get a competitive ratio of $\frac{1}{1+2r^*}$, at least* 0.2862*.*

Proof. The value r^* is obtained by setting the two terms in the minimum in Theorem 10 equal to each other and solving for r, giving the lower bound on the competitive ratio.

To show that this result is tight, consider the following input: p_1 of weight α arrives at the north pole of the unit sphere, followed by p_2 of weight $\beta \geq \alpha$ at the south pole of the unit sphere, followed by p_3 of weight $r^*\beta - \epsilon$ at the west pole of the unit sphere, and followed by p_4 of weight $r^*\beta - \epsilon$ at the east pole of the unit sphere. The algorithm would end up matching p_1 with p_2, leaving p_3 and p_4 unmatched. Thus, in such an instance, the competitive ratio of the algorithm is $(\alpha + \beta)/(\alpha + \beta + 2r^*\beta - 2\epsilon)$. Taking β to ∞ and ϵ to 0 shows that the algorithm does not guarantee a competitive ratio better than $1/(1 + 2r^*)$ in the strict sense. ◀

7 Algorithms with Advice

In this section, we consider the OWNM problem in the tape advice setting. In the advice setting, a trustworthy oracle cooperates with an online algorithm according to a pre-agreed protocol. The oracle has access to the entire input sequence in advance. The oracle communicates with an online algorithm by writing bits on the advice tape. When an input item arrives, an algorithm reads some number of advice bits from the tape, and makes a decision for the new item based on the advice it read from the tape so far, and the items that have arrived so far. The question of interest is to bound the number of bits that need to be communicated between the oracle and the algorithm on the worst-case input to allow an online algorithm to solve the problem optimally. For an introduction to online algorithms with advice, an interested reader is referred to the survey [12] and references therein.

We propose an online algorithm with advice, which we call "Split-And-Match" (Sam), and show that it achieves optimality. For input sequences of size $2n$, Sam uses a family of C_n advice strings, where C_n is the n^{th} Catalan number. The oracle encodes each advice string using Elias delta coding scheme [16], which requires $\lceil \log C_n \rceil + \log n + O(\log\log(C_n))$ bits. We ignore the $O(\log\log(C_n))$ term for simplicity.

The Sam oracle and algorithm jointly maintain a partitioning of the plane into convex regions, and a responsibility relation, where a point can be assigned to be responsible for at most one region, and each region can have at most one point responsible for it. Each region defined will eventually receive an even number of points in total. No region which has not yet been divided into sub-regions contains more than one unmatched point. When a new point p arrives in a region R, if R does not have a responsible point, then p is assigned to be the responsible point for R, and p is left unmatched at this time. Otherwise, suppose that q is the responsible point for R at the time p arrived. In this case, the responsibility of q is removed, and the plane partition is refined by subdividing R into R_1 and R_2 – the sub-regions of R formed by $\overline{pq}$. If the total number of points (including future points, but excluding p and q) in R_i is even for each $i \in \{1, 2\}$, then p and q are matched (we refer to this event as a "safe match"), and R_1 and R_2 do not have any responsible points assigned to them. Otherwise, p and q are not matched, and q is made responsible for R_1, and p is made responsible for R_2. Note that when a region has a responsible point, that point is assumed to lie in the region by convention, though it can lie on the boundary.

To implement the above procedure in the advice model, the Sam oracle creates a binary string D of length $2n$, where the i^{th} bit indicates whether p_i arrives in a region which has some responsible point p_j assigned to it, and p_i and p_j form a safe match. The string D is encoded on the tape and is passed to Sam. The Sam algorithm reads the encoding of D from the tape (prior to the arrival of online points), recovers D from the encoding, and then uses the information in D to run the above procedure creating safe matches.

Observe that we aim to show the bound $\log C_n + \log n \sim 2n - \frac{1}{2}\log n$ on the advice complexity. The reason for the additive savings of $\frac{3}{2}\log n$ in the $\log C_n$, as compared to $2n$, is that not all binary strings of length $2n$ can be generated as a valid D. Thus, the oracle and the algorithm can agree beforehand on the ordering of the universe of possible strings D, which we call the advice family. Then the oracle writes on the tape the index of a string in this ordering that corresponds to D for the given input. The following theorem establishes the correctness of this algorithm, as well as the claimed bound on the advice complexity.

▶ **Theorem 12.** *Sam achieves a perfect matching with the advice family of size C_n.*

Proof. Since the request sequence contains $2n$ points, an even number of points eventually arrive. Inductively, a region that is divided always has an even number of points in both sub-regions, and each of these sub-regions is convex. Thus, any point arriving in a region can be matched to the point that is already there and responsible for the region. There are only two kinds of regions that occur during the execution of Sam, as a new point arrives:

- type I: this region does not have a responsible point, it is empty at the time of creation, and there are an even number of points arriving in this region in the future, and
- type II: this region has a responsible point, which is the only point in the region at the time of its creation, and there are an odd number of points arriving in this region in the future.

We argue inductively (on the number of future points arriving in a region) that the algorithm ends up matching all points inside a region, regardless of their type. The base case for a type I region is trivial: the number of future points is 0, and there is nothing to prove. The base case for type II region is easy: one point arrives in the region, then according to the algorithm it will be matched to the responsible point (since R_1 and R_2 are empty).

For the inductive step, consider a type I region R, and suppose that $2k$ points will arrive inside the region. The first point that arrives in the region becomes responsible for this region, changing its type to II. There are $2k-1$ future points arriving in this region, and the claim follows by the inductive assumption applied to the type II region. Now, consider a type II region R, and suppose that $2k-1$ points arrive inside the region. Let q be the responsible point for R, and let p be the first point arriving inside R. Note that $\overline{pq}$ partitions R into R_1 and R_2. There are two possible cases. Case 1: If R_1 and R_2 are both of type I, then p is matched with q and the inductive step is established for R by invoking induction on R_1 and R_2. If Case 2: R_1 and R_2 are both of type II, then inductive step is established for R by invoking induction on R_1 and R_2.

Observe that the entire plane is a region of type I at the beginning of the execution of the algorithm (prior to arrival of any points). Thus, correctness of the algorithm follows by applying the above claim to this region.

To establish the bound on advice complexity, observe that by the definition of the algorithm, Sam matches the most recent point whenever $D[i]$ is 1 and does not match otherwise. Thus, D has an equal number of zeros and ones and no prefix of D has more ones than zeros. This makes D a Dyck word and it is known that there are C_n Dyck words of size $2n$ [29]. ◀

8 Conclusion

We introduced the weighted version of the Online Weighted Non-Crossing Matching problem. We established that no deterministic algorithm can guarantee a constant competitive ratio for this problem. Then, we explored several ways of overcoming this limitation and presented new algorithms and bounds for each of the considered regimes. In particular, we presented the results for deterministic algorithms when weights of the points are restricted to lie in the range $[1, U]$, randomized algorithms without restrictions on weights, deterministic algorithms with revoking, and deterministic algorithms with advice. Many open problems remain. In particular, our bounds are not tight, and closing the gap in any of the settings would be of interest. It is also interesting to study the online setting of other versions of the problem that were considered in the offline literature. For example, one could allow an online algorithm to create some number of crossings up to a given budget, or one could consider the k-non-crossing constraint as inspired from understanding RNA structures. In this paper, we considered the vertex-weighted version, but one could also consider an edge-weighted version of the problem, where edge weights could be either abstract, or related to geometry.

References

1 A Karim Abu-Affash, Paz Carmi, Matthew J Katz, and Yohai Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457, 2014.

2 Oswin Aichholzer, Sergey Bereg, Adrian Dumitrescu, Alfredo García, Clemens Huemer, Ferran Hurtado, Mikio Kano, Alberto Márquez, David Rappaport, Shakhar Smorodinsky, et al. Compatible geometric matchings. *Computational Geometry*, 42(6-7):617–626, 2009.

3 Oswin Aichholzer, Sergio Cabello, Ruy Fabila-Monroy, David Flores-Penaloza, Thomas Hackl, Clemens Huemer, Ferran Hurtado, and David R Wood. Edge-removal and non-crossing configurations in geometric graphs. *Discrete Mathematics & Theoretical Computer Science*, 12(Graph and Algorithms), 2010.

4 Oswin Aichholzer, Ruy Fabila-Monroy, Philipp Kindermann, Irene Parada, Rosna Paul, Daniel Perz, Patrick Schnider, and Birgit Vogtenhuber. Perfect matchings with crossings. *Algorithmica*, pages 1–20, 2023.

5 Noga Alon, Sridhar Rajagopalan, and Subhash Suri. Long non-crossing configurations in the plane. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 257–263, 1993.

6 Greg Aloupis, Esther M Arkin, David Bremner, Erik D Demaine, Sándor P Fekete, Bahram Kouhestani, and Joseph SB Mitchell. Matching regions in the plane using non-crossing segments. *XVI Spanish Meeting on Computational Geometry*, 2015.

7 Drew Armstrong, Christian Stump, and Hugh Thomas. A uniform bijection between nonnesting and noncrossing partitions. *Transactions of the American Mathematical Society*, 365(8):4121–4151, 2013.

8 Mikhail J Atallah. A matching problem in the plane. *Journal of Computer and System Sciences*, 31(1):63–70, 1985.

9 Vineet Bafna, S Muthukrishnan, and R Ravi. Computing similarity between rna strings. In *Combinatorial Pattern Matching: 6th Annual Symposium, CPM 95 Espoo, Finland, July 5–7, 1995 Proceedings 6*, pages 1–16. Springer, 1995.

10 József Balogh, Boris Pittel, and Gelasio Salazar. Near-perfect non-crossing harmonic matchings in randomly labeled points on a circle. *Discrete Mathematics & Theoretical Computer Science*, Proceedings, 2005.

11 Prosenjit Bose, Paz Carmi, Stephane Durocher, Shahin Kamali, and Arezoo Sajadpour. Non-crossing matching of online points. In *32 Canadian Conference on Computational Geometry*, 2020.

12 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: a survey. *ACM Computing Surveys*, 50(32):1–34, 2017. Article No. 19.

13 William YC Chen, Hillary SW Han, and Christian M Reidys. Random k-noncrossing rna structures. *Proceedings of the National Academy of Sciences*, 106(52):22061–22066, 2009.

14 Scott Cohen. *Finding color and shape patterns in images.* PhD dissertation, Stanford University, 1999.

15 Adrian Dumitrescu and William Steiger. On a matching problem in the plane. *Discrete Mathematics*, 211(1-3):183–195, 2000.

16 Peter Elias. Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory*, 21(2):194–203, 1975.

17 András Gyárfás. Ramsey and turán-type problems for non-crossing subgraphs of bipartite geometric graphs. In *Annales Univ. Sci. Budapest*, volume 54, pages 47–56, 2011.

18 Koki Hamada, Shuichi Miyazaki, and Kazuya Okamoto. Strongly stable and maximum weakly stable noncrossing matchings. *Algorithmica*, 83(9):2678–2696, 2021.

19 John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32(2):249–267, 1992.

20 John Hershberger and Subhash Suri. Efficient breakout routing in printed circuit boards. In *Workshop on Algorithms and Data Structures*, pages 462–471. Springer, 1997.

21 Shahin Kamali, Pooya Nikbakht, and Arezoo Sajadpour. A randomized algorithm for non-crossing matching of online points. In *34th Canadian Conference on Computational Geometry*, pages 198–204, 2022.

22 Mikio Kano and Jorge Urrutia. Discrete geometry on colored point sets in the plane—a survey. *Graphs and Combinatorics*, 37(1):1–53, 2021.

23 Ioannis Mantas, Marko Savić, and Hendrik Schrezenmaier. New variants of perfect non-crossing matchings. *Discrete Applied Mathematics*, 343:1–14, 2024.

24 Neeldhara Misra, Harshil Mittal, and Saraswati Nanoti. Diverse non crossing matchings. In *34th Canadian Conference on Computational Geometry*, pages 249–256, 2022.

25 Ali Mohammad Lavasani and Denis Pankratov. Advice complexity of online non-crossing matching. *Computational Geometry*, 110:101943, 2023. doi:10.1016/j.comgeo.2022.101943.

26 János Pach, Natan Rubin, and Gábor Tardos. Planar point sets determine many pairwise crossing segments. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1158–1166, 2019.

27 Rebecca Patrias, Oliver Pechenik, and Jessica Striker. A web basis of invariant polynomials from noncrossing partitions. *Advances in Mathematics*, 408:108603, 2022.

28 David Rappaport. Tight bounds for visibility matching of f-equal width objects. In *Japanese Conference on Discrete and Computational Geometry*, pages 246–250. Springer, 2002.

29 Steven Roman. *An Introduction to Catalan Numbers*. Compact Textbooks in Mathematics. Brikhäuser, 2015.

30 Suthee Ruangwises and Toshiya Itoh. Stable noncrossing matchings. In *International Workshop on Combinatorial Algorithms*, pages 405–416. Springer, 2019.

31 Arezoo Sajadpour. Non-crossing matching of online points. Master's thesis, University of Manitoba, 2021.

32 Toshinori Sakai and Jorge Urrutia. Heavy non-crossing increasing paths and matchings on point sets. In *XIV Spanish Meeting on Computational Geometry*, pages 209–212, 2011.

33 David Savitt. Polynomials, meanders, and paths in the lattice of noncrossing partitions. *Transactions of the American Mathematical Society*, 361(6):3083–3107, 2009.

34 Alexander A Vladimirov. Non-crossing matchings. *Problems of Information Transmission*, 49(1):54–57, 2013.

A Omitted Pseudocode

Algorithm 1 *Split-And-Match* Oracle.

```
procedure Split-And-Match-Oracle
    $D \leftarrow [0]$
    make $p_1$ responsible for the plane
    for $i = 2$ to $2n$ do
        let $R$ be the region that $p_i$ arrives in
        if $R$ has a responsible point $p_j$ then
            revoke the responsibility of $p_j$
            divide $R$ into $R_1$ and $R_2$ by $\overline{p_i p_j}$
            if $R_L$ (and $R_R$) is going to contain an even number of points in total then
                $D.append(1)$
            else
                make $p_j$ and $p_i$ responsible for $R_1$ and $R_2$ respectively
                $D.append(0)$
        else
            make $p_i$ responsible for $R$
            $D.append(0)$
    pass $D$ to the algorithm
```

Algorithm 2 *Split-And-Match* Algorithm.

```
procedure Split-And-Match(D)
    while receive a new point $p_i$ do
        let $R$ be the region that $p_i$ arrives in
        if $R$ has a responsible point $p_j$ then
            revoke the responsibility of $p_j$
            divide $R$ into $R_1$ and $R_2$ by $\overline{p_i p_j}$
            if $D[i] == 1$ then
                match $p_i$ with $p_j$
            else
                make $p_j$ and $p_i$ responsible for $R_1$ and $R_2$ respectively
                leave $p_i$ unmatched
        else
            make $p_i$ responsible for $R$
            leave $p_i$ unmatched
```

Deterministic Cache-Oblivious Funnelselect

Gerth Stølting Brodal
Aarhus University, Denmark

Sebastian Wild
University of Liverpool, UK

Abstract

In the multiple-selection problem one is given an unsorted array S of N elements and an array of q query ranks $r_1 < \cdots < r_q$, and the task is to return, in sorted order, the q elements in S of rank $r_1, \ldots, r_q$, respectively. The asymptotic deterministic comparison complexity of the problem was settled by Dobkin and Munro [JACM 1981]. In the I/O model an optimal I/O complexity was achieved by Hu *et al.* [SPAA 2014]. Recently [ESA 2023], we presented a *cache-oblivious* algorithm with matching I/O complexity, named *funnelselect*, since it heavily borrows ideas from the cache-oblivious sorting algorithm *funnelsort* from the seminal paper by Frigo, Leiserson, Prokop and Ramachandran [FOCS 1999]. Funnelselect is inherently randomized as it relies on sampling for cheaply finding many good pivots. In this paper we present *deterministic funnelselect*, achieving the same optimal I/O complexity cache-obliviously without randomization. Our new algorithm essentially replaces a single (in expectation) reversed-funnel computation using random pivots by a recursive algorithm using multiple reversed-funnel computations. To meet the I/O bound, this requires a carefully chosen subproblem size based on the entropy of the sequence of query ranks; deterministic funnelselect thus raises distinct technical challenges not met by randomized funnelselect. The resulting worst-case I/O bound is $O\big(\sum_{i=1}^{q+1} \frac{\Delta_i}{B} \cdot \log_{M/B} \frac{N}{\Delta_i} + \frac{N}{B}\big)$, where B is the external memory block size, $M \geq B^{1+\varepsilon}$ is the internal memory size, for some constant $\varepsilon > 0$, and $\Delta_i = r_i - r_{i-1}$ (assuming $r_0 = 0$ and $r_{q+1} = N + 1$).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Multiple selection, cache-oblivious algorithm, entropy bounds

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.17

Funding *Gerth Stølting Brodal*: Independent Research Fund Denmark, grant 9131-00113B.
Sebastian Wild: Engineering and Physical Sciences Research Council grant EP/X039447/1.

1 Introduction

We present the first optimal deterministic cache-oblivious algorithm for the multiple-selection problem. In the multiple-selection problem one is given an unsorted array S of N elements and an array R of q query ranks in increasing order $r_1 < \cdots < r_q$, and the task is to return, in sorted order, the q elements of S of rank $r_1, \ldots, r_q$, respectively; (see Figure 1 for an example).

On top of immediate applications, like finding a set of quantiles in a numerical data set, the multiple-selection problem is of interest as it gives a natural common generalization of (single) selection by rank (using a single query rank $r_1 = r$) and fully sorting an array (corresponding

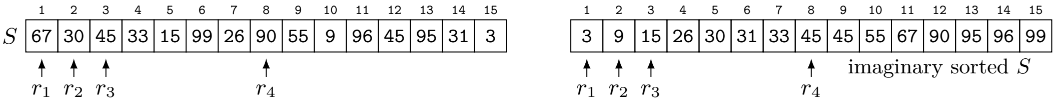


Figure 1 Example input with $N = 15$, $q = 4$ and $R[1..q] = [1, 2, 3, 8]$. The expected output `3`, `9`, `15`, `45` is obvious from the sorted array (right). (The sorted array is for illustration only; the goal of efficient multiple-selection algorithms is to avoid ever fully sorting the input.)

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 17; pp. 17:1–17:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to selecting every index as a query rank, i.e., $q = N$ and $r_i = i$ for $i = 1, \dots, N$). It thus allows us to quantitatively study the transition between these two foundational problems, which are of different complexity and each have their distinct set of algorithms. For example, the behavior of selection and sorting with respect to external memory is quite different: For single selection, the textbook median-of-medians algorithm [4] simultaneously works with optimal cost in internal memory, external memory, and the cache-oblivious model (models are defined below). For sorting, by contrast, the introduction of each model required a substantially modified algorithm to achieve optimal costs: Standard binary mergesort is optimal in internal memory, but requires $\approx M/B$-way merging to be optimal in external memory, where M is the internal memory size and B the external memory block size, measured in elements [1]; achieving the same cache obliviously, i.e., without knowledge of B and M, requires the judiciously chosen buffer sizes from the recursive constructions of funnelsort [12].

Since multiple selection simultaneously generalizes both problems, it is not surprising that also here subsequent refinements were necessary going from internal to external to cache-oblivious; the most recent result being our algorithm funnelselect [7]. However, all algorithms mentioned above for single selection and sorting are *deterministic*. By constrast, funnelselect is inherently relying on randomization and known deterministic external-memory algorithms [2, 15] are crucially relying on the knowledge of M and B. Prior to this work it thus remained open whether a single deterministic cache-oblivious algorithm exists that smoothly interpolates between selection and sorting without having to resort to randomization.

In this paper, we answer this question in the affirmative. Our algorithm *deterministic funnelselect* draws on techniques from cache-oblivious sorting (funnelsort) and existing multiple-selection algorithms, but it follows a rather different approach to our earlier randomized algorithm [7] and previous (cache-conscious) external-memory algorithms. A detailed comparison is given below.

1.1 Model of computation and previous work

Our results are in the cache-oblivious model of Frigo, Leiserson, Prokop and Ramachandran [13], a hierarchical-memory model with an infinite external memory and an internal memory of capacity M elements, where data is transferred between internal and external memory in blocks of B consecutive elements. Algorithms are compared by their I/O cost, i.e., the number of block transfers or *I/O*s (input/output operations). This is similar to the external-memory model by Aggarwal and Vitter [1]. Crucially, in the cache-oblivious model, no variables are allowed to depend on the model parameters M and B and I/Os are assumed to be performed automatically by an optimal (offline) paging algorithm. The algorithms are analyzed with respect to the parameters M and B. Cache-oblivious algorithms hence work for any parameters M and B, and they even adapt to multi-level memory hierarchies (under certain conditions [13]).

The multiple-selection problem was first formally addressed by Chambers [8], who considered it a generalization of quickselect [14]. Prodinger [17] proved that Chambers' algorithm achieves an optimal *expected* running time up to constant factors: $O(\mathcal{B}+N)$, where $\mathcal{B} = \sum_{i=1}^{q+1} \Delta_i \lg \frac{N}{\Delta_i}$ with $\Delta_i = r_i - r_{i-1}$, for $1 \le i \le q+1$, assuming $r_0 = 0$ and $r_{q+1} = N+1$, and lg denoting the binary logarithm. We call $\mathcal{B}$ the *(query-rank) entropy* of the sequence of query ranks [2]. It should be noted that $\mathcal{B} + N = O(N(1 + \lg q))$, but the latter bound does not take the location of query ranks into account; for example, if $q = \Theta\big(\sqrt{n}\big)$ queries are in a range of size $O(N/\lg N)$, i.e., $r_q - r_1 = O(N/\lg N)$, then the entropy bound is $O(N)$ whereas the latter $N(1 + \lg q) = \Theta(N \lg N)$.

Table 1 Algorithms for selection and multiple selection. CO = cache-oblivious, $\mathbb{E}$ = expected, wc = worst-case bounds. Note that Barbay *et al.* assume a tall cache $M \geq B^{1+\varepsilon}$, whereas Hu *et al.* do not.

Reference		Comparisons	I/Os	Comments
Single selection				
Hoare [14]	$\mathbb{E}$	$2\ln 2\mathcal{B} + 2N + o(N)$	$O(N/B)$	CO, randomized
Floyd & Rivest [11]	$\mathbb{E}$	$N + \min\{r, N-r\} + o(N)$	$O(N/B)$	CO, randomized
Blum *et al.* [4]	wc	$5.4305N$	$O(N/B)$	CO, deterministic
Schönhage *et al.* [18]	wc	$3N + o(N)$	?	CO, median, deterministic
Dor & Zwick [10]	wc	$2.95N + o(N)$	?	CO, median, deterministic
Multiple selection				
Chambers [8, 17]	$\mathbb{E}$	$2\ln 2\mathcal{B} + O(N)$	$O((\mathcal{B} + N)/B)$	CO, randomized
Dobkin & Munro [9]	wc	$3\mathcal{B} + O(N)$	$O((\mathcal{B} + N)/B)$	CO, deterministic
Kaligosi *et al.* [16]	wc	$\mathcal{B} + o(\mathcal{B}) + O(N)$	$O((\mathcal{B} + N)/B)$	CO, deterministic
Hu *et al.* [15]	wc	$O(N \lg(q))$	$O(N/B \cdot \log_{M/B}(q/B))$	deterministic
	wc	$O(\mathcal{B} + N)$	$O(\mathcal{B}_{\mathrm{I/O}} + N/B)$	(from closer analysis)
Barbay *et al.* [2]	wc	$\mathcal{B} + o(\mathcal{B}) + O(N)$	$O(\mathcal{B}_{\mathrm{I/O}} + N/B)$	online, determ., $M \geq B^{1+\varepsilon}$
Brodal & Wild [7]	$\mathbb{E}$	$O(\mathcal{B} + N)$	$O(\mathcal{B}_{\mathrm{I/O}} + N/B)$	CO, randomized, $M \geq B^{1+\varepsilon}$
This paper	wc	$O(\mathcal{B} + N)$	$O(\mathcal{B}_{\mathrm{I/O}} + N/B)$	CO, deterministic, $M \geq B^{1+\varepsilon}$

Dobkin and Munro [9] showed that $\mathcal{B} - O(N)$ comparisons are necessary to find all ranks $r_1, \ldots, r_q$ (in the worst case). Deterministic algorithms with that same $O(\mathcal{B} + N)$ running time are also known [9, 16], but as for single selection, the deterministic algorithms were presented later than the randomized algorithms and require more sophistication. Multiple selection in external-memory was studied by Hu *et al.* [15] and Barbay *et al.* [2]. Their algorithms have an I/O cost of $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$, where the *"I/O entropy"* $\mathcal{B}_{\mathrm{I/O}} = \frac{\mathcal{B}}{B \lg(M/B)}$. An I/O cost of $\Omega(\mathcal{B}_{\mathrm{I/O}}) - O\big(\frac{N}{B}\big)$ is known to be necessary [2, 7]. A more comprehensive history of the multiple-selection problem appears in [7]; Table 1 gives an overview.

Algorithms designed for a single level memory model (RAM model) can be analyzed in the cache-oblivious model. Analyzing existing time- and comparison-optimal multiple-selection algorithms [8, 9, 16, 17] in the cache-oblivious model, shows that they are not optimal with respect to the number of I/Os performed. The obtained I/O bounds are a factor $\lg(M/B)$ away from being optimal. To get a feeling for the I/O bounds in Table 1, consider for example the case where $B = \lg N$, $M = B^2$, and there are $q = \lg N$ evenly distributed queries ($r_i = \big\lfloor \frac{iN}{q+1} \big\rfloor$). In this case $\mathcal{B} = \Theta(N \lg\lg N)$. The cache-oblivious sorting bound (see Lemma 6) is $O\big(\frac{N}{B} \log_M N\big) = O\big(\frac{N}{B} \cdot \frac{\lg N}{\lg\lg N}\big)$, $O((\mathcal{B} + N)/B) = O\big(\frac{N}{B} \lg\lg N\big)$ and $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big) = O\big(\frac{N}{B}\big)$, where $O\big(\frac{N}{B}\big)$ is the number of I/Os required to scan the input.

1.2 Result

Our main result is the cache-oblivious algorithm *deterministic funnelselect* achieving the following efficiency (see Theorem 10 for the full statement and proof).

▶ **Theorem 1.** *There exists a deterministic cache-oblivious algorithm solving the multiple-selection problem using $O(\mathcal{B} + N)$ comparisons and $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os in the worst case, assuming a tall cache $M \geq B^{1+\varepsilon}$.*

At the high level, our algorithm uses the standard overall idea of a recursive partitioning algorithm and pruning recursive calls containing no rank queries, an idea dating back to the first algorithm by Chambers [8]. In the cache-aware external-memory model, I/O efficient algorithms are essentially obtained by replacing binary partitioning (as used in [8]) by an external-memory $\Theta(M/B)$-way partitioning [2, 15]. Unfortunately, in the cache-oblivious model this is not possible, since the parameters M and B are unknown to the algorithm. To be I/O efficient in the cache oblivious model, both our previous algorithm randomized funnelselect [7] and our new algorithm deterministic funnelselect apply a cache-oblivious multi-way k-partitioner to distribute elements into k buckets given a set of $k-1$ pivot elements, essentially reversing the computation done by the k-merger used by funnelsort [12]. The k-partitioner is a balanced binary tree of $k-1$ pipelined binary partitioners.

The key difference between our randomized and deterministic algorithms is that in our randomized algorithm we use a single $N^{\Theta(\varepsilon)}$-way partitioner using randomly selected pivots and truncate work inside the partitioner for subproblems that (with high probability) will not contain any rank queries. This is done by estimating the ranks of the pivots through sampling and pruning subproblems estimated to be sufficiently far from any query ranks. In our deterministic version, we choose k smaller and deterministically compute pivots, such that all elements are pushed all the way down through a k-partitioner without truncation (eliminating the need to know the (approximate) ranks of the pivots before the k-partitioning is finished), while we choose k such that the buckets with unresolved rank queries (that we have to recursive on) in total contain *at most half* of the elements. To compute k, we apply a linear-time *weighted*-median finding algorithm on $\Delta_1, \ldots, \Delta_{q+1}$. While randomized funnelselect can handle buckets with unresolved rank queries directly using sorting, deterministic funnelselect needs to recursively perform multiple-selection on the buckets to achieve the desired I/O performance.

2 Preliminaries

Throughout the paper we assume that the input to a multiple-selection algorithm is given as two arrays $S[1..N]$ and $R[1..q]$, where S is an unsorted array of N elements from a totally ordered universe, and R is a sorted array $r_1, \ldots, r_q$ of q distinct query ranks, where $1 \leq r_1 < \cdots < r_q \leq N$. The array S is allowed to contain duplicate elements. Our task is to produce/report an array of the q order statistics $S_{(r_1)}, \ldots, S_{(r_q)}$, where $S_{(r)}$ is the rth smallest element in S, i.e., the element at index r in an array storing S after sorting it.

Our new deterministic cache-oblivious multiple-selection algorithm makes use of the following three existing cache-oblivious results for single selection, weighted selection, sorting, and multi-way partitioning.

▶ **Lemma 2** (Blum, Floyd, Pratt, Rivest, Tarjan [4, Theorem 1]). *Selecting the k-th smallest element in an unsorted array of N elements can be done with $O(N)$ comparisons and $O\big(1+\frac{N}{B}\big)$ I/Os in the cache-oblivious model.*

▶ Remark 3 (Median of medians: I/O cost). Although the original paper by Blum *et al.* [4] predates the cache-oblivious model [12] by decades, analyzing the algorithm in the cache-oblivious model with a stack-oriented memory allocator gives a linear I/O cost, since the algorithm is based on repeatedly scanning geometrically decreasing subproblems.

▶ Remark 4 (Median of medians: duplicates). The original algorithm in [4] assumes that all elements are distinct. The algorithm can be extended to handle duplicates (by performing a three-way partition of the elements into those less-than, equal-to, and greater-than a pivot, respectively), and to return a triple $S_{\leq}, p, S_{\geq}$, that is a partition of S, where p is the element of rank k, $S_{\leq}$ are the elements of rank $1, \ldots, k-1$ in arbitrary order, and $S_{\geq}$ are the elements of rank $k+1, \ldots, |S|$ in arbitrary order (where duplicate elements are assigned consecutive ranks in an arbitrary order).

In the *weighted selection* problem we are giving an array of N elements, each with an associated non-negative weight, and a target weight W. The goal is find the smallest k, where the sum of the weights of the k smallest elements is at least W, and to return the k-th smallest element. A linear-time weighted-selection algorithm can be derived from the unweighted selection algorithm by Blum *et al.* [4] (Lemma 2) – as hinted by Shamos in [19] and spelled out in detail by Bleich and Overton [3] – by computing the weighted rank of the pivot. The weighted selection algorithm follows essentially the same recursion as [4], and it similarly follows that it is cache oblivious and performs $O\big(1 + \frac{N}{B}\big)$ I/Os.

▶ **Lemma 5** (Bleich, Overton [3])**.** *Weighted selection in an unsorted array of N weighted elements can be done with $O(N)$ comparisons and $O\big(1 + \frac{N}{B}\big)$ I/Os in the cache-oblivious model.*

▶ **Lemma 6** (Frigo, Leiserson, Prokop, Ramanchandran [13, Theorem 7], Brodal, Fagerberg [5, Theorem 2])**.** *Funnelsort sorts an array of N elements using $O(N \lg N)$ comparisons and $O\big(\frac{N}{B}(1+\log_M N)\big)$ I/Os in a cache-oblivious model with a tall-cache assumption $M \geq B^{1+\varepsilon}$, for constant $\varepsilon > 0$.*

▶ Remark 7 (Tall and taller). The original description of funnelsort by Frigo *et al.* [12] assumed the tall cache assumption $M = \Omega(B^2)$, whereas [5] observed that this could be relaxed to the weaker tall cache assumption $M = \Omega\big(B^{1+\varepsilon}\big)$. I/O optimality of funnelsort follows from a matching external-memory lower bound by Aggarwal and Vitter [1, Theorem 3.1].

The key innovation in our previous randomized algorithm funnelselect [7] is the k-*partitioner* (Figure 2), a cache-oblivious and I/O-efficient multi-way partitioning algorithm to distribute a batch of elements around $k-1$ given pivots into k buckets; the precise characteristics are summarized in the following lemma.

▶ **Lemma 8** (Brodal and Wild [7, Lemma 3])**.** *Given an unsorted array of $N \geq k^d$ elements and $k-1$ pivots $P_1 \leq \cdots \leq P_{k-1}$, a k-partitioner can partition the elements into k buckets $S_1, \ldots, S_k$, such all elements x in bucket S_i satisfy $P_{i-1} \leq x \leq P_i$. The algorithm is cache-oblivious and performs $O(N \lg k)$ comparisons and $O\big(k + \frac{N}{B}(1+\log_M k)\big)$ I/Os, provided a tall-cache assumption $M \geq B^{1+\varepsilon}$ and $d \geq \max\{1+2/\varepsilon, 2\}$. The working space for the k-partitioner (ignoring input and output buffers) is $O\big(k^{(d+1)/2}\big)$. This is also the time required to construct a k-partitioner (again ignoring input and output buffers).*

The k-partitioners are structurally similar to the k-mergers from funnelsort for merging k runs cache obliviously. In [7] we pipeline the partitioning by essentially reversing the computations done by funnelsort, and replace each binary merging node by a binary partitioning node.

3 Deterministic multiple-selection

In this section we present our deterministic cache-oblivious multiple-selection algorithm that performs optimal $O(\mathcal{B} + N)$ comparisons and $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os, under a tall-cache assumption $M \geq B^{1+\varepsilon}$. Detailed pseudo-code is given in Algorithm 1 and Algorithm 2, and the basic idea is illustrated in Figure 3.

Given a tall-cache assumption $M \geq B^{1+\varepsilon}$, we let $d = \max\{1 + 2/\varepsilon, 2\}$. The algorithm follows the general idea of making a recursive multi-way partition of the array of elements and to only recurse on subproblems with unresolved rank queries. For two consecutive query ranks r_{i-1} and r_i, we say that the $\Delta_i = r_i - r_{i-1}$ elements of rank $r_{i-1}+1, \ldots, r_i$ are in a

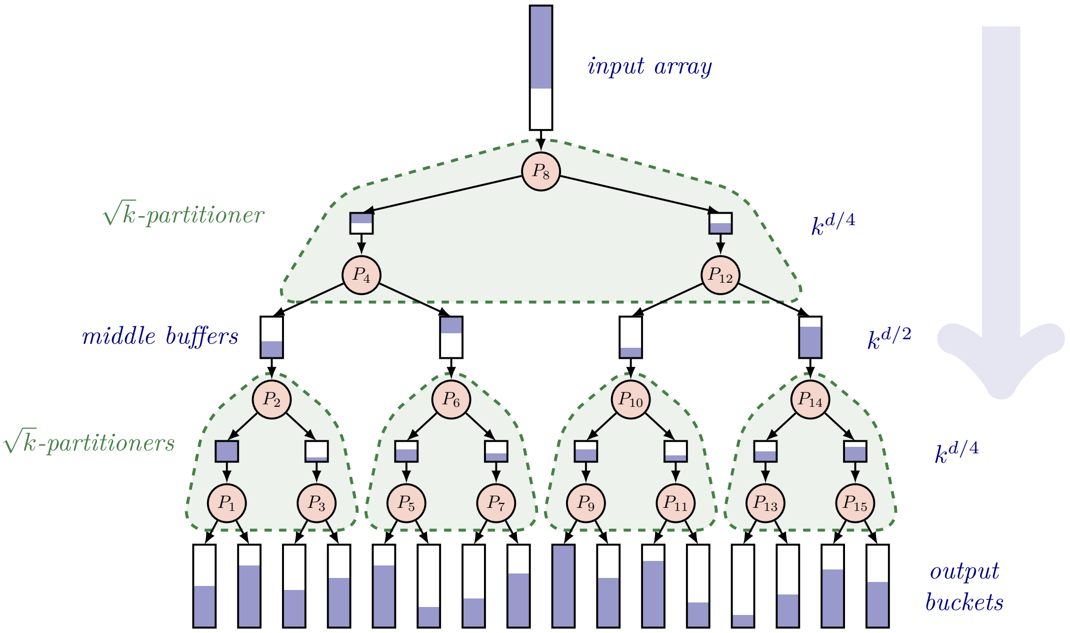


Figure 2 A k-partitioner for $k = 16$ buckets. Content in the buffers is shaded; buffers are filled bottom-to-top; when full, they are flushed and then consumed from the bottom. The figure shows the situation where the input buffer for P_6 is being flushed down to its children (by partitioning elements around pivot P_6). The flush at P_6 was triggered during flushing P_4's input buffer, which in turn has been called while flushing P_8 (the input).
Buffer sizes for the three internal levels are shown next to the buffers. k-partitioners are defined recursively from a $\sqrt{k}$-partitioner at the top, a collection of $\sqrt{k}$ middle buffers, and $\sqrt{k}$ further $\sqrt{k}$-partitioners, each partitioning from one middle buffer to $\sqrt{k}$ output buffers. (All sizes here ignore floors and ceilings; for the precise definition valid for all k, see [7].)

gap of size Δ_i. We choose a parameter Δ, such that at least half of the elements are in gaps of size $\leq \Delta$ and simultaneously at least half (rounded down) of the elements are in gaps of size $\geq \Delta$. To compute Δ (Algorithm 1, line 4), we compute $\Delta_i = r_i - r_{i-1}$ by a scan over the query ranks $r_1, \ldots, r_q$ (and $r_0 = 0$ and $r_{q+1} = N + 1$), and perform weighted selection (Lemma 5) among $\Delta_1, \ldots, \Delta_{q+1}$, where Δ_i has weight $w_i = \Delta_i$, and return the smallest Δ where $\sum_{\Delta_i \leq \Delta} w_i \geq N/2 + 1$.

For the case when Δ is small compared to N (formally, $(2N)^d \geq \Delta^{d+1}$ or $N^{1+\frac{1}{1+\varepsilon}} \geq \Delta^2$), we simply solve the multiple-selection problem by sorting the elements (cache-obliviously using funnelsort [13]), and report the elements with ranks $r_1, \ldots, r_q$ by a single scan over the sorted elements. The condition on Δ implies $\mathcal{B}_{\text{I/O}} = \Omega(\text{Sort}_{M,B}(N))$, where $\text{Sort}_{M,B}(N) = \Theta\big(\frac{N}{B}\big(1+\log_{M/B}\frac{N}{B}\big)\big)$ is the number of I/Os required to sort N elements in external memory [1], so this is within a constant factor of the I/O lower bound (detailed analysis in Section 4).

Otherwise, we create a k-partition, where $k = \Theta\big(\frac{N}{\Delta}\big)$ as follows (MultiPartition in Algorithm 2): We repeatedly distribute batches of Δ elements into a set of buckets separated by pivot elements. Initially we have one empty bucket and no pivot. Whenever a bucket reaches size $> \Delta$, the bucket is split into two buckets of size $\leq \Delta$ separated by a new pivot using the (cache-oblivious) linear-time median selection algorithm (Lemma 2). To distribute a batch of elements into the current set of buckets we use a cache-oblivious k-partitioner (Lemma 8, which depends on the tall-cache assumption parameter d) built using the current set of pivots. Note that we need to construct a new k-partitioner after each batch of Δ elements has been distributed, since the number of buckets and pivots can increase. For the

Algorithm 1 Deterministic cache-oblivious multiple-selection.

```
procedure DeterministicFunnelSelect($S[1..N]$, $R[1..q]$)
    if $q > 0$ then
        $\Delta_i \leftarrow R[i] - R[i-1]$ for $i = 1,\dots,q+1$, assuming $R[0] = 0$ and $R[q+1] = N+1$
        $\Delta \leftarrow \min\big\{\Delta_i \in \{\Delta_1,\dots,\Delta_{q+1}\} \bigm| \sum_{j\in\{1,\dots,q+1\}:\Delta_j\le\Delta_i} \Delta_j \ge N/2+1\big\}$
        if $(2N)^d \ge \Delta^{d+1}$ or $N^{1+\frac{1}{1+\varepsilon}} \ge \Delta^2$ then        ▷ $\mathcal{B}_{\mathrm{I/O}} = \Omega(\mathrm{Sort}_{M,B}(N))$
            $S \leftarrow$ FunnelSort($S$)
            Report $S[R[1]],\dots,S[R[q]]$
        else
            $(P_1,\dots,P_{k-1}),(S_1,\dots,S_k) \leftarrow$ MultiPartition($S,\Delta$)
            $\bar r_0 \leftarrow 0$
            for $i \leftarrow 1,\dots,k$ do
                $\bar r_i \leftarrow \bar r_{i-1} + |S_i| + 1$                        ▷ $\bar r_i$ is rank of $P_i$
                $R_i \leftarrow \{r \mid r \in R \wedge \bar r_{i-1} < r < \bar r_i\}$        ▷ Rank queries to bucket $S_i$
                if $|R_i| > 0$ then
                    $r_i^{\max} \leftarrow \max(R_i)$
                    $\bar S_i, p_{\max}, S_{\ge} \leftarrow$ Select($S_i, r_i^{\max} - \bar r_{i-1}$)
                    if $|R_i| > 1$ then
                        $r_i^{\min} \leftarrow \min(R_i)$
                        $S_{\le}, p_{\min}, \bar S_i \leftarrow$ Select($\bar S_i, r_i^{\min} - \bar r_{i-1}$)
                        Report $p_{\min}$
                        if $|R_i| > 2$ then
                            $\bar R_i \leftarrow \{r - r_i^{\min} \mid r \in R_i \setminus \{r_i^{\min}, r_i^{\max}\}\}$
                            DeterministicFunnelSelect($\bar S_i, \bar R_i$)
                    Report $p_{\max}$
                if $\bar r_i \in R$ then
                    Report $P_i$
```

Algorithm 2 Given an array S with N elements and a bucket capacity Δ, where $(2N)^{\frac{d}{d+1}} \le \Delta \le N$, partition S into k buckets $S_1,\dots,S_k$ separated by $k-1$ pivots $P_1,\dots,P_{k-1}$, where $\left\lfloor \frac{\Delta}{2} \right\rfloor \le |S_i| \le \Delta$.

```
procedure MultiPartition($S[1..N]$, $\Delta$)
    Requires $(2N)^{\frac{d}{d+1}} \le \Delta \le N$
    $k \leftarrow 1$, $S_1 \leftarrow \{\}$                ▷ Initially only one empty bucket and no pivots
    for $i \leftarrow 1$ to $N$ step $\Delta$ do
        $\bar S \leftarrow S[i..\min(i+\Delta-1,N)]$                ▷ Next batch to distribute to buckets
        Distribute $\bar S$ to buckets $S_1,\dots S_k$ using pivots $P_1,\dots,P_{k-1}$ with a $k$-partitioner
        while there exists a bucket $S_j$ with $|S_j| > \Delta$ do                ▷ Split bucket $S_j$
            $S_{\le}, p, S_{\ge} \leftarrow$ Select($S_j, \lceil |S_j|/2 \rceil$)
            Rename $S_{j+1},\dots,S_k$ to $S_{j+2},\dots,S_{k+1}$ and $P_j,\dots,P_{k-1}$ to $P_{j+1},\dots,P_k$
            $S_j \leftarrow S_{\le}$, $P_j \leftarrow p$, $S_{j+1} \leftarrow S_{\ge}$
            $k \leftarrow k+1$
    return $(P_1,\dots,P_{k-1}),(S_1,\dots,S_k)$
```

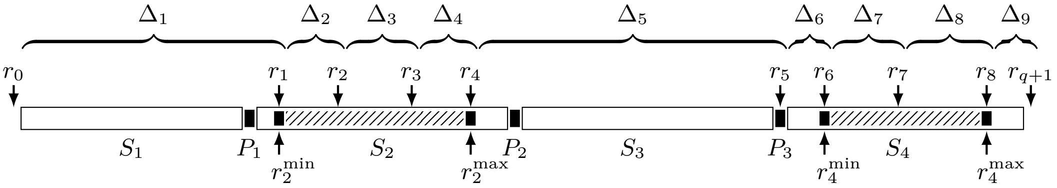


Figure 3 Deterministic multiple selection. The partition of an array S into buckets $S_1, \ldots, S_4$ separated by pivots $P_1, \ldots, P_3$, and query ranks $r_1, \ldots, r_8$. In the example the maximum allowed bucket size is $\Delta = \Delta_1$, since $\Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_6 + \Delta_7 + \Delta_8 + \Delta_9 \geq |S|/2 + 1$ and $\Delta_2 + \Delta_3 + \Delta_4 + \Delta_6 + \Delta_7 + \Delta_8 + \Delta_9 < |S|/2 + 1$. Black squares are pivots and the shaded regions in buckets are the subproblems to recurse on.

computation to be I/O efficient, we allocate in memory space for a $\lfloor \frac{2N}{\Delta} \rfloor$-partitioner followed by space for $\lfloor \frac{2N}{\Delta} \rfloor$ buckets of capacity 2Δ (in the proof of Lemma 9 we argue that the number of buckets created is at most $\frac{2N}{\Delta}$ and each bucket will never exceed 2Δ elements). The space for the partitioner is reused for each new batch, and whenever a bucket is split into two new buckets, one bucket remains in the old bucket's allocated space and the other bucket is placed in next available slot for a bucket. This ensures all buckets are stored consecutively in memory, albeit in arbitrary order.

After having constructed the buckets we compute the ranks of the pivots from the bucket sizes, and consider the buckets with at least one unresolved rank query. If the rank of a pivot coincides with a query rank, we report this pivot just after having considered the preceding bucket. Before recursing on the elements in a bucket, we first find the minimum and maximum query ranks $r^{\min}$ and $r^{\max}$ in the bucket by a scan over the bucket's query ranks, and find and report the corresponding elements in the bucket using linear-time selection (Lemma 2). Finally, we only recurse on the elements between ranks $r^{\min}$ and $r^{\max}$, provided there are any unresolved rank queries to the bucket. This ensures that when recursing on a subproblem of size $\bar{N}$, all elements in the subproblem are in gaps of size $< \bar{N}$ in the original input. By reporting the elements at the appropriate times during the recursion, elements will be reported in increasing order.

The partitioning of an array S into buckets is illustrated in Figure 3. The crucial property is that for a gap $\Delta_i \geq \Delta$, the two query ranks r_{i-1} and r_i defining the gap *cannot* be in the same bucket, implying that no element in this gap will be part of a recursive subproblem (see, e.g., gaps Δ_1 and Δ_5 in Figure 3).

Pseudocode for our algorithm is shown in Algorithm 1 and Algorithm 2. We assume $\textsc{Select}(S, k)$ is the deterministic linear-time selection algorithm from Lemma 2, and that it returns a triple $S_{\leq}, p, S_{\geq}$, that is a partition of S, where p is the element of rank k, $S_{\leq}$ are the elements of rank $1, \ldots, k-1$ in arbitrary order, and $S_{\geq}$ the elements of rank $k+1, \ldots, |S|$ in arbitrary order.

4 Analysis

We first analyze the number of comparisons and I/Os performed by MultiPartition in Algorithm 2, that deterministically performs a k-way partition of N elements into $k = O\big(\frac{N}{\Delta}\big)$ buckets separated by $k-1$ pivots, where each bucket has size at most Δ. The following lemma summarizes the precise properties of MultiPartition.

▶ **Lemma 9.** *For $N \geq \Delta$ and $\Delta^{d+1} \geq (2N)^d$, MultiPartition creates $k \leq \frac{2N}{\Delta}$ buckets and $k-1$ pivots, each bucket has size at most Δ, and performs $O(N \lg k)$ comparisons and $O\big(k^2 + \frac{N}{B}(1 + \log_M k)\big)$ I/Os.*

Proof. We first bound the sizes of the buckets created by MultiPartition. The algorithm repeatedly distributes batches of at most Δ elements to buckets and splits all overflowing buckets of size $> \Delta$ before considering the next batch. It is an invariant that before distributing a batch, all buckets have size at most Δ. Furthermore, as soon as the first bucket is split, all buckets have size at least $\lfloor \frac{\Delta}{2} \rfloor$, since whenever an overflowing bucket of size $s > \Delta$ is split the new buckets have initial sizes $\lfloor \frac{s-1}{2} \rfloor$ and $\lceil \frac{s-1}{2} \rceil$. Here "$-1$" is due to one element becomes a pivot. The smallest bucket size is achieved when $s = \Delta + 1$, where the smallest bucket size is $\lfloor \frac{\Delta+1-1}{2} \rfloor = \lfloor \frac{\Delta}{2} \rfloor$. Note that the buckets after the split have size at most Δ, since all buckets had at most Δ elements before the distribution of a batch of at most Δ elements to the buckets, i.e., $s \leq 2\Delta$. To bound the total number of buckets k created, observe that if $\Delta = N$ then no bucket will be split and $k = 1$. Otherwise, $\Delta < N$ and at least two buckets are created, and $k\lfloor \frac{\Delta}{2} \rfloor + k - 1 \leq N$, since all buckets have size at least $\lfloor \frac{\Delta}{2} \rfloor$ and there are $k - 1$ pivots. We have $N \geq k\big(\frac{\Delta}{2} - \frac{1}{2}\big) + k - 1 = \frac{k\Delta}{2} + \frac{k}{2} - 1 \geq \frac{k\Delta}{2}$, since $k \geq 2$, i.e., the total number of buckets created $k \leq \frac{2N}{\Delta}$.

To analyze the number of comparisons and I/Os performed, we need to consider the $\lceil \frac{N}{\Delta} \rceil$ distribution steps and at most $\frac{2N}{\Delta} - 1$ bucket splittings. Since each bucket splitting involves at most 2Δ elements, each bucket splitting can be performed cache-obliviously by a linear-time selection algorithm (Lemma 2) using $O(\Delta)$ comparisons and $O\big(1 + \frac{\Delta}{B}\big)$ I/Os, assuming each bucket is stored in a buffer of 2Δ consecutive memory cells. In total the $k - 1 = \Theta\big(\frac{N}{\Delta}\big)$ bucket splittings require $O(N)$ comparisons and $O\big(k + \frac{N}{B}\big)$ I/Os. A k-partitioner for partitioning Δ elements uses $O(\Delta \lg k)$ comparisons and $O\big(k + \frac{\Delta}{B}(1 + \log_M k)\big)$ I/Os (Lemma 8), assuming k is sufficiently small according to the tall-cache assumption (see below). This includes the cost of constructing the k-partitioner. The total cost for all $\lceil \frac{N}{\Delta} \rceil$ distribution steps becomes $O(N \lg k)$ comparisons and $O\big(k\frac{N}{\Delta} + \frac{N}{B}(1 + \log_M k)\big) = O\big(k^2 + \frac{N}{B}(1 + \log_M k)\big)$ I/Os.

By Lemma 8, the tall-cache assumption $M \geq B^{1+\varepsilon}$ implies that for a k-partitioner and an input of size Δ, it is required that $\Delta \geq k^d$ for the I/O bounds to hold (recall $d = \max\{1 + 2/\varepsilon, 2\}$). The input assumption $\Delta \geq \big(\frac{2N}{\Delta}\big)^d$ together with $k \leq \frac{2N}{\Delta}$ ensure that $\Delta \geq k^d$. ◀

We now prove our main result that DeterministicFunnelselect in Algorithm 1 is an optimal deterministic cache-oblivious multiple-selection algorithm. Crucial to the analysis is to show that the choice of Δ balances early pruning of buckets without queries with simultaneously achieving efficient I/O bounds.

▶ **Theorem 10.** DeterministicFunnelselect *performs $O(\mathcal{B} + N)$ comparisons and $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os cache-obliviously in a cache model with tall assumption $M \geq B^{1+\varepsilon}$, for some constant $\varepsilon > 0$.*

Proof. We first consider the consequences of the choice of Δ. By the choice of Δ, we have $\sum_{\Delta_i < \Delta} \Delta_i < N/2 + 1$. Since each bucket S_i has size at most Δ, and we only recurse on subsets that are (the union of) gaps where the two bounding rank queries of the gaps are *both* in the same bucket, we only recurse on gaps with $\Delta_i < \Delta$ elements (see Figure 3). A recursive subproblem between query ranks r_s and r_t, where $1 \leq s < t \leq q$, contains $r_t - r_s - 1 = \big(\sum_{i=s+1}^{t} \Delta_i\big) - 1$ elements. It follows that

(A) all recursive subproblems in total contain at most $\sum_{\Delta_i < \Delta} \Delta_i - 1 < N/2$ elements and each subproblem has size $\leq \Delta - 2$.

(B) $\sum_{\Delta_i \leq \Delta} \Delta_i \geq N/2 + 1$, i.e., at least $N/2$ elements are in gaps of size at most Δ.

To analyze the number of comparisons performed, we use a potential argument where one unit of potential can pay for $O(1)$ comparisons, and all comparisons performed can be charged to the released potential. We define the potential of an element x in a gap of size Δ_i to be $1 + \lg \frac{N}{\Delta_i}$, where N is the size of the current recursive subproblem x resides in. The total initial potential is at most $N + \sum_{i=1}^{q+1} \Delta_i \lg \frac{N}{\Delta_i} = O(\mathcal{B} + N)$.

We first consider the number of comparisons for the non-sorting case (Algorithm 1, lines 9–26). If an element x in a gap of size $\Delta_i \leq \Delta$ participates in a recursive call of size $< \Delta$, the potential released for x is at least $\bigl(1 + \lg \frac{N}{\Delta_i}\bigr) - \bigl(1 + \lg \frac{\Delta}{\Delta_i}\bigr) = \lg \frac{N}{\Delta}$. If an element x in a gap of size $\Delta_i \leq \Delta$ does not participate in a recursive call, the potential released for x is $1 + \lg \frac{N}{\Delta_i} \geq 1 + \lg \frac{N}{\Delta}$. Finally, elements in gaps of size $> \Delta$ will not participate in recursive calls, and will each release at least potential 1. It follows that the released potential is at least $\frac{N}{2} + \frac{N}{2} \lg \frac{N}{\Delta}$, since at least $N/2$ elements are in gaps of size $\leq \Delta$ (property (B), contributing the second summand) and at most $N/2$ elements are in gaps of size $< \Delta$ and participate in recursive calls (property (A)), i.e., at least $N/2$ elements are in gaps of size $\geq \Delta$ (contributing the first summand). By Lemma 9, MultiPartition requires $O(N \lg k)$ comparisons. Since $k = O(N/\Delta)$ (Lemma 9), these comparisons can be covered by the released potential. The additional comparisons required for computing Δ with a linear-time weighted section algorithm (Lemma 5) and performing Select (Lemma 2) at most twice on each bucket require in total at most $O(N)$ comparisons, and can also be charged to the released potential. It follows that for the non-sorting case the released potential can cover for all comparisons performed.

In the sorting case, a single call to Funnelsort is performed causing $O(N \lg N)$ comparisons (Lemma 6). No further recursive calls are made and the potential of all elements is released. At least $N + \frac{N}{2} \lg \frac{N}{\Delta}$ potential is released, since at least $N/2$ elements are in gaps of size $\leq \Delta$ (property (B)). In the sorting case, either $(2N)^d \geq \Delta^{d+1}$ or $N^{1+\frac{1}{1+\varepsilon}} \geq \Delta^2$. If $(2N)^d \geq \Delta^{d+1}$, we have $\Delta \leq (2N)^{\frac{d}{d+1}}$ and $\frac{N}{\Delta} \geq N/(2N)^{\frac{d}{d+1}} \geq \frac{1}{2} N^{\frac{1}{d+1}}$. It follows that the released potential is at least $N + \frac{N}{2} \lg\bigl(\frac{1}{2} N^{\frac{1}{d+1}}\bigr) \geq \frac{1}{2(d+1)} N \lg N$, covering the cost for the comparisons. Otherwise, $N^{1+\frac{1}{1+\varepsilon}} \geq \Delta^2$, i.e., $\Delta \leq N^{\frac{1}{2}\left(1+\frac{1}{1+\varepsilon}\right)}$ and we have $\frac{N}{\Delta} \geq N/N^{\frac{1}{2}\left(1+\frac{1}{1+\varepsilon}\right)} = N^{\frac{\varepsilon}{2(1+\varepsilon)}}$ and the potential released is at least $N + \frac{N}{2} \lg \frac{N}{\Delta} \geq N + \frac{\varepsilon}{4(1+\varepsilon)} N \lg N$ and can cover the cost for the comparisons. Note that the comparison bound depends on the tall-cache parameters ε and d.

To analyze the I/O cost we assign an I/O potential to an element x in gap of size Δ_i of $\frac{1}{B}\bigl(1 + \log_M \frac{N}{\Delta_i}\bigr)$, where N is the size of the current subproblem x resides in. Similar to the comparison potential, it follows that the non-sorting case releases I/O potential $\frac{1}{2}\bigl(\frac{N}{B} + \frac{N}{B} \log_M \frac{N}{\Delta}\bigr)$. The number of I/Os required is $O\bigl(1 + \frac{q}{B}\bigr) = O\bigl(1 + \frac{N}{B}\bigr)$ I/Os for scanning R and computing Δ using weighted selection (Lemma 5), $O\bigl(k + \frac{N}{B}\bigr)$ I/Os for selecting the minimum and maximum rank elements in each bucket (Lemma 2), and $O\bigl(k^2 + \frac{N}{B}(1 + \log_M k)\bigr)$ I/Os for the k-partitioning (Lemma 8), i.e., in total $O\bigl(k^2 + \frac{N}{B}(1 + \log_M k)\bigr)$ I/Os. It follows that the I/O cost can be charged to the released potential, provided $k^2 = O\bigl(\frac{N}{B}\bigr)$. To address this, we need to consider two cases depending on the size N of a subproblem. If the problem completely fits in internal memory together with all the geometric decreasing recursive subproblems, assuming a stack-oriented memory allocation, then considering this problem will in total cost $O\bigl(1 + \frac{N}{B}\bigr)$ I/Os, including all recursive subproblems. That means, there exists a constant $c > 0$ such that for $N \leq cM$, the I/O cost for handling such problems can be charged to the parent subproblem creating the subproblem. It follows that we only need to consider the I/O cost for subproblems of size $N \geq cM$. Since $M \geq B^{1+\varepsilon}$, we have $N \geq cM \geq cB^{1+\varepsilon}$, i.e., $B \leq \bigl(\frac{N}{c}\bigr)^{\frac{1}{1+\varepsilon}}$. Since $k = O\bigl(\frac{N}{\Delta}\bigr)$, to prove $k^2 = O\bigl(\frac{N}{B}\bigr)$ it is sufficient to prove $\left(\frac{N}{\Delta}\right)^2 = O\left(\frac{N}{(N/c)^{1/(1+\varepsilon)}}\right)$. This holds, e.g., when $N^{1+\frac{1}{1+\varepsilon}} \leq \Delta^2$, which is always fulfilled in the non-sorting case. For the sorting case, we have similarly to the comparison potential that $\Omega\left(\frac{N}{B}\log_M N\right)$ I/O potential is released, which can cover the I/O cost for cache-oblivious sorting (Lemma 6). ◀

5 Conclusion

With deterministic funnelselect, we close the gap left in previous work and obtain an I/O-optimal cache-oblivious multiple-selection algorithm that does not need to resort to randomization to achieve its performance. This settles the complexity of the multiple-selection problem in the cache-oblivious model (including the fine-grained analysis based on the query-rank entropy $\mathcal{B}$).

There are open questions left in other variants of the problem. Like randomized funnelselect [7], deterministic funnelselect cannot deal with queries arriving in an online fashion, one after the other. This problem has been addressed in the external-memory model [2], but no cache-oblivious I/O-optimal solution is known.

Concerning the transition from single selection by rank to sorting, which multiple selection allows us to study, some questions remain unanswered. For example, in the cache-oblivious model, it is known that sorting with optimal I/O-complexity is only possible under a tall-cache assumption [6] (such as the one made in this work); for single selection, however, such a restriction is not necessary. It would be interesting to study the transition between the problems and find out, how “sorting-like” a multiple-selection instance has to be to likewise require a tall cache for I/O-optimal cache-oblivious algorithms.

Another direction for future work are parallel algorithms for multiple selection that are also cache-oblivious and I/O efficient.

References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 Jérémy Barbay, Ankur Gupta, Srinivasa Rao Satti, and Jon Sorenson. Near-optimal online multiselection in internal and external memory. *Journal of Discrete Algorithms*, 36:3–17, January 2016. doi:10.1016/j.jda.2015.11.001.

3 Chaya Bleich and Michael L. Overton. A linear-time algorithm for the weighted median problem. Technical Report 75, New Yourk University, Department of Computer Science, April 1983. URL: https://archive.org/details/lineartimealgori00blei/.

4 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

5 Gerth Stølting Brodal and Rolf Fagerberg. Cache oblivious distribution sweeping. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 426–438. Springer, 2002. doi:10.1007/3-540-45465-9_37.

6 Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 307–315. ACM, 2003. doi:10.1145/780542.780589.

7 Gerth Stølting Brodal and Sebastian Wild. Funnelselect: Cache-oblivious multiple selection. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.25.

8 J. M. Chambers. Partial sorting [M1] (algorithm 410). *Commun. ACM*, 14(5):357–358, 1971. doi:10.1145/362588.362602.

9 David P. Dobkin and J. Ian Munro. Optimal time minimal space selection algorithms. *J. ACM*, 28(3):454–461, 1981. doi:10.1145/322261.322264.

10 Dorit Dor and Uri Zwick. Selecting the median. *SIAM Journal on Computing*, 28(5):1722–1758, 1999. doi:10.1137/s0097539795288611.

11 Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, March 1975. doi:10.1145/360680.360691.

12 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 285–298. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814600.

13 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

14 C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, 1961. doi:10.1145/366622.366647.

15 Xiaocheng Hu, Yufei Tao, Yi Yang, and Shuigeng Zhou. Finding approximate partitions and splitters in external memory. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*. ACM, June 2014. doi:10.1145/2612669.2612691.

16 Kanela Kaligosi, Kurt Mehlhorn, J. Ian Munro, and Peter Sanders. Towards optimal multiple selection. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2005. doi:10.1007/11523468_9.

17 Helmut Prodinger. Multiple Quickselect – Hoare's Find algorithm for several elements. *Information Processing Letters*, 56(3):123–129, November 1995. doi:10.1016/0020-0190(95)00150-b.

18 Arnold Schönhage, Mike Paterson, and Nicholas Pippenger. Finding the median. *J. Comput. Syst. Sci.*, 13(2):184–199, 1976. doi:10.1016/S0022-0000(76)80029-3.

19 Michael Ian Shamos. Geometry and statistics: Problems at the interface. In Joseph Frederick Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 251–280. Academic Press, 1976. URL: http://euro.ecom.cmu.edu/people/faculty/mshamos/1976Stat.pdf.

Dynamic L-Budget Clustering of Curves

Kevin Buchin
Faculty of Computer Science, TU Dortmund University, Germany

Maike Buchin
Faculty of Computer Science, Ruhr University Bochum, Germany

Joachim Gudmundsson
Faculty of Engineering, The University of Sydney, Australia

Lukas Plätz
Faculty of Computer Science, Ruhr University Bochum, Germany

Lea Thiel
Faculty of Computer Science, Ruhr University Bochum, Germany

Sampson Wong
Department of Computer Science, University of Copenhagen, Denmark

Abstract

A key goal of clustering is data reduction. In center-based clustering of complex objects therefore not only the number of clusters but also the complexity of the centers plays a crucial role. We propose L-Budget Clustering as unifying perspective on this task, optimizing the clustering under the constraint that the summed complexity of all centers is at most L. We present algorithms for clustering planar curves under the Fréchet distance, but note that our algorithms more generally apply to objects in metric spaces if a notion of simplification of objects is applicable. A scenario in which data reduction is of particular importance is when the space is limited. Our main result is an efficient $(8+\varepsilon)$-approximation algorithm with a $(1+\varepsilon)$-resource augmentation that maintains an L-budget clustering under insertion of curves using only $O(L\varepsilon^{-1})$ space and $O^*(L^3 \log L + L^2 \log(r^*/r_0))$ time where O^* hides factors of ε^{-1}.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases clustering, streaming algorithm, polygonal curves, Fréchet distance, storage efficiency, simplification, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.18

Supplementary Material *Software*: `https://github.com/NathenMat/SWAT24` [7]
archived at `swh:1:dir:c28806ab33cb4a564e1d492efd322b1dc32f7245`

Funding *Lukas Plätz*: The work was supported by the PhD School "SecHuman – Security for Humans in Cyberspace" by the federal state of NRW.

1 Introduction

Clustering is a key technique for reducing dataset size in big data and serves as a fundamental analysis task. The goal is to keep data characteristics as close as possible to the original while limiting the size and complexity. In particular, when dealing with very large data, such as live traffic data, this needs to be compressed regularly. This necessarily leads to an approximation of the dataset, and thus we always pay an approximation factor in each later analysis. This tradeoff between space-efficient data storage and accuracy poses a challenge for long-term analysis. To address this, we introduce a clustering strategy that approximately bounds the clustering error within our storage constraints.

If the objects to be clustered are simply points, k-center clustering directly provides a compact representation, since it is sufficient to store k points as cluster centers Gonzalez [13] and Hochbaum and Shmoys [15] give 2-approximation algorithms for k-center clustering,

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which is optimal assuming $P \neq NP$. These algorithms, though, require access to all points at once. Charikar et al. [12] present a dynamic k-center clustering algorithm for point sets, where points can be added to the clustering. Their doubling algorithm allows them to process the points one at a time, which gives an 8-approximation and can run using $O(k)$ space. This result was improved by McCutchen and Khuller [18] to a $(2+\varepsilon)$-approximation. A technique they use to achieve this is to run multiple instances with different radii to remove a factor from the approximation. This increases the runtime and memory usage by $O(\varepsilon^{-1}\log\varepsilon^{-1})$.

These techniques can only approximate storage requirements when all objects possess identical, small complexity. This assumption, however, may oversimplify the reality and prove too substantial for various types of data, such as libraries of images, movement-tracking trajectories, and networks. Even for seemingly simple objects like real numbers, their digital representation requires simplification, such as floating-point precision. Moreover, by adopting a simplified approach to clustering, they inadvertently constrain their options. Typically, this results in a binary decision: deleting or keeping the object as it is. In contrast, when we consider complex objects, they often bring a notion of simplifications.

In this work, we focus on polygonal curves as these occur naturally in many settings, and the literature provides a solid foundation of clustering, simplification, and similarity computation. They also show all the interesting features of complex objects. Nevertheless, our present techniques apply to a wider field of objects. The specific setting only influences the runtime analysis, and our clustering results hold in a more general setting. When we cluster curves, given some accuracy, we see that some may share a complicated center path, and others are similar in their position and have only a simple center path.

One option to deal with this setting is the (k,l)-*clustering problem.* It gives a clustering of complex objects by simplifying them to a uniform size of l. This saves the upper bound on the storage occupied by the clustering. Curves can be found in many applications and have different similarity metrics. We are interested in the Fréchet distance, which respects the traversal of a curve (cf. Hausdorff) but not its sampling (cf. dynamic-time-warping). For curves utilizing the Fréchet distance as the similarity metric, the centers are selected from the entire domain of Fréchet curves[1]. We choose the center curves freely i.e. under the unrestricted simplification setting. The selection process aims to minimize the maximum cluster radius. Buchin et al. [8] proposed a 3-approximation for this setting. Also for the setting of (k,l)-median clustering of curves, where not the maximum radius but the summed radii are considered, approximation algorithms have been developed [6, 10].

However, (k,l)-clustering may yield an imbalanced solution to the original problem, where some centers are inherently more complex than others. Additionally, determining the appropriate number of clusters in advance poses a challenge, often addressed by computing clustering for various values of k. Restricting the number of clusters can limit the full benefit of simplifications or the identification of cluster centers that genuinely represent the underlying data. Pre-determination of a suitable l may also be challenging and may depend on other considerations, such as the readability of centers.

We introduce L-*budget clustering* as a solution in this setting. The L-budget clustering problem is a center clustering problem with a complexity associated with each center. In a solution to it, the sum over the complexity of the centers is at most L and minimizes the maximum radius of the clusters. See Figure 1 for a small example.

[1] To establish the Fréchet Distance as a metric, we consider the equivalence classes of curves sharing the same trace, using the quotient space.

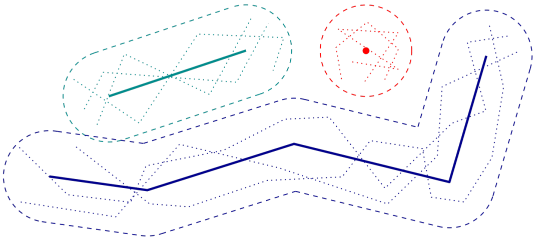


Figure 1 Small example of an L-budget clustering consisting of three clusters for $L = 8$.

Hence, we do not assume to know any meta parameters, nor do our guarantees rely on them, such as the number of clusters in our dataset or the objects' uniform complexity. Instead, we restrict ourselves to claims in the resource we are willing to spend: memory space. The strict size constraint allows us to optimize the representativeness using the given space. We assume that each input curve has complexity L. This can be achieved if necessary with a streaming simplification algorithm as described in Remark 9. To continuously build and analyze the data, we present a 1-pass streaming $(8+\varepsilon)$-approximation algorithm for L-budget clustering of curves with computing space in the output size and a $(1+\varepsilon)$-resource augmentation. That means it only needs to see the data once and does not require extra computation space, allowing us to handle large datasets. The approximation factor can be decreased to $(2+\varepsilon)$ but with an increase of $O(\varepsilon^{-1})$ in runtime and space requirements.

Overview. In Section 2, we summarize the simplification problems for curves and the current techniques to solve them. Here, we present an algorithm for a $(1+\varepsilon)$-approximation to find the minimum complex simplification with linear space. In Section 3, we present the static L-budget clustering problem and continue in Section 4 with the dynamic (k, l)-clustering. In Section 5, we then handle the dynamic L-budget clustering. We end with a short experimental evaluation in Section 6.

2 Simplification

Simplifying a curve is a natural and well-studied problem. It asks to reduce the complexity of a curve while keeping it as similar as possible to the original curve. For the similarity, different distance measures can be used. We will use the popular Fréchet distance $\mathbf{d}_F$ [4]. We denote the complexity of a curve z with $\mathbf{cplx}(z)$, which counts the number of points.

Simplifying a curve is then a bi-criterion optimization problem of the size and distance of the simplified curve (to the original curve). So, the literature discusses two subproblems. With min–#, we denote the simplification problem where we start with an upper bound on the Fréchet distance and minimize the complexity l of the simplification. The other variant is min–$\mathbf{d}$[2], where we start with an upper bound ℓ on the complexity and want to minimize the Fréchet distance $\mathbf{d}_F$. We are interested in the global unrestricted setting, the least-restricted simplification setting, to guarantee the best possible clustering. This allows us to get the least possible error given the size of the simplification, which in turn allows a better representation of the curves with the same budget. Global means that the Fréchet distance between the simplification and the curve is minimized [20]. This is in contrast to the local setting where the curve is partitioned. Then, the Fréchet distance is calculated

[2] We changed min–ε to min–$\mathbf{d}$ because it conflicts with the ε from full approximation schemas.

between the parts and its simplification, and the maximum over this is minimized [14]. In particular in earlier research on simplification, only curves with vertices of the original were considered as simplification [16]. This is the vertex-restricted case. Later, this was loosened to the unrestricted setting where the vertices can be from the whole metric space [14, 3]. See van de Kerkhof et al. [20] for a more detailed introduction.

We summarize relevant results on simplification in Table 1 and convert them to unrestricted global simplifications with Agarwal et al. [3]. This table contains offline and online algorithms which have different analysis settings. Offline algorithms are commonly analyzed with worst-case analysis and online algorithms with competitive ratios. However, simplification is a particularly hard setting and sometimes requires resource augmentation[3] [1, 20]. To unify the notation, we call the competitive ratio of the online algorithm just an approximation ratio but compare it to an offline algorithm that optimizes each instance individually.

Table 1 Results on Curve Simplification. In the min–**d** setting, first is the approximation ratio and then resource augmentation. In the min–# setting, it is reversed. The O^* hides terms in ε.

Authors and Paper	Setting	$(\mathbf{d}_F, \mathbf{cplx})$	Runtime	Space
Guibas et al. [14]	min–#, global, $\mathbb{R}^2$	$(1,1)$	$O(m^2 \log^2 m)$	$O(m)$
Agarwal et al. [3]	min–#, global	$(1,8)$	$O(m \log m)$	$O(l)$
van de Kerkhof et al. [20]	min–#, global	$(1+\varepsilon, 2)$	$O^*(m^2 \log m \log\log m)$	$O^*(m)$
Abam et al. [1]	min–**d**, global	$(16\sqrt{2}+\varepsilon, 2)$	$O^*(\ell)$	$O^*(\ell^2)$
Buchin et al. [8]	min–**d**, global	$(4,1)$	$O(m^2 \ell \log m)$	$O(m)$
this paper Thm. 3	min–**d**, global, $\mathbb{R}^2$	$(1+\varepsilon, 1)$	$O^*((\log m + \ell) m^2 \log m)$	$O(m)$

One can change the simplification criterion of an algorithm by applying a binary search, as described by Chan and Chin [11].

For our clustering algorithms in Section 3, we need to find solutions to the min–# problem. We will call the algorithm that does that $\mathbf{S}_{\#}$. Guibas et al. [14] gave in theorem 14 with definition 4 an algorithm to solve this for planar curves in $O(m^2 \log^2 m)$ time.

We are interested in the min–**d**-simplification problem in the next sections, which we describe as finding a curve's best ℓ-simplification. This will be used in the later clustering algorithms. To prevent approximation factors in the space as much as possible, as this may be a hard constraint, we want an algorithm that has an competitive ratio of 1. For this we use a known algorithm by Imai-Iri, which was previously analysed and used by Buchin et al. [8]. Here, we briefly show how to get the $O(m)$ space and improve the approximation factor in Fréchet distance for planar curves using the disk stabber from Guibas et al. [14].

Kreveld et al. [21] gave an exact algorithm for min–#-simplification with dynamic programming in $O(m^2)$ space [4]. But this needs too much space for our application.

Our goal is to get a constant factor approximation in $O(m)$ space and improve it with a binary search to a $(1+\varepsilon)$-approximation. In the constrained space, we implicitly build the shortcut graph from Imai and Iri [16] and compute the best simplification given the length with a dynamic program. So, we compute the distance from the current edge to the sub-curve with the Fréchet distance algorithm from Alt and Godau [4]. The sub-curve can have at most length m, and the algorithm of Alt and Godau can run in $O(m)$ space. We compute the minimum Fréchet distance r necessary to reach a node i with an l-simplification

[3] Resource augmentation compares offline against online algorithms with more resources.
[4] In the look up table we only consider $k-1$ and can delete after a full iteration.

with the Algorithm 1 (Apx. Simplification). We save these values in a table $z[i,l]$. The solution to our problem is the value of $z[m,\ell]$, where m is the length of the input curve. To compute a $z[i,l]$, we need to compute the maximum of the previous $z[j,l-1]$ and the Fréchet distance between the sub curve $c[j,i]$ and the shortcut $\overline{c[j,i]}$, which we notate as $\mathbf{d}_F(\overline{c[j,i]}, c[j,i]))$. We then can compute the minimum over all j. As we only need the last iteration over the length of the simplification, we only need space in the size of the number of points in the curve, which is $O(m)$.

Algorithm 1 Apx. Simplification.

```
Data: polygonal curve $z$, complexity of the simplification $\ell \in \mathbb{N}$
Result: subsequence of $z$ of length at most $\ell$ with minimal Fréchet distance to $z$
for $l \leftarrow 1$ to $\ell$ do
    for $i \leftarrow 0$ to $m$ do
        $z[i,l] \leftarrow \min(\max(\{z[j,l-1], \mathbf{d}_F(\overline{c[j,i]}, c[j,i]))\,|\, j < i\})$
    delete all $z[\cdot, l-1]$
```

▶ **Lemma 1** (By Imai and Iri [16]). *Algorithm 1 (Apx. Simplification) gives us an optimal vertex-restricted local Fréchet simplification.*

▶ Remark 2. Algorithm 1 (Apx. Simplification) gives a 4-approximation of the weak unrestricted simplification [3]. It has an $O(m^2 \ell \log m)$ runtime [8] and needs $O(m)$ space.

So, we have established an interval of constant size where the optimal value lies. We now do a binary search and decide with the algorithm from Guibas et at. [14] if an ℓ-simplification exists.

Algorithm 2 (1+ε)-Apx. Simplification.

```
Data: polygonal curve $z$, complexity of the simplification $\ell \in \mathbb{N}$
Result: subsequence of $z$ of length at most $\ell$ with minimal Fréchet distance to $z$
$s \leftarrow Apx.Simplification(z, \ell)$
$h \leftarrow \mathbf{d}_F(s, z)$
$l \leftarrow h/4$
while $h - l > \varepsilon$ do
    $m \leftarrow (h + l)/2$
    if len($\mathbf{S}_{\#}(z, m)$) $\leq \ell$ then $h \leftarrow m$ else $l \leftarrow m$
return $h$
```

▶ **Theorem 3.** *Algorithm 2 ((1+ε)-Apx. Simplification) computes a $(1+\varepsilon)$-simplification of the* min–***d*** *without resource augmentation. The runtime is in $O((\log \varepsilon^{-1} \log m + \ell) m^2 \log m)$ and the space is in $O(m)$.*

Proof. The initial approximation needs $O(m^2 \ell \log m)$ runtime. To improve the approximation factor from 4 to (1+ε), we need $\log(4\varepsilon^{-1})$ iterations of binary search. As the decision algorithm, we take [14] that needs $O(m^2 \log^2 m)$ time. This gives us the $O(\log \varepsilon^{-1} m^2 \log^2 m + m^2 \ell \log m)$ runtime. ◀

▶ Remark 4. We will later want to use the clustering algorithm for point sets from Charikar et al. [12]. It uses different radii r over its runtime. Storing the simplifications for each r will not be possible because of the space restriction of $O(L\varepsilon^{-1})$. Thus, we compute a *range of simplifications* that use $O(L\varepsilon^{-1})$ space and choose a simplification later when r increases.

The range of simplification are all simplifications of size $(1+\varepsilon)^{-i}L$ for all $i \in [-1, \log_{(1+\varepsilon)} L]$. The geometric series bounds the size of the range with $O(L\varepsilon^{-1})$. See Figure 2 for a small example.

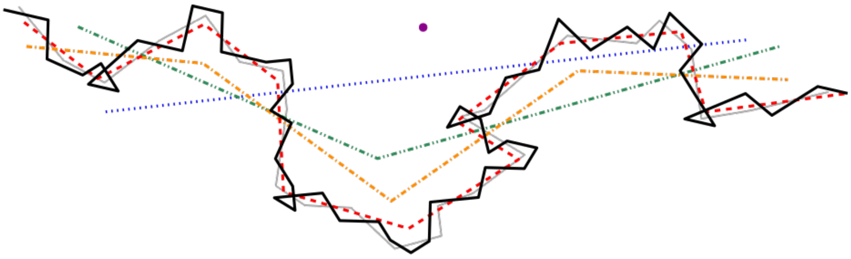


Figure 2 A non-optimal simplification range with factor 2. The colors and line styles indicate different stages of simplifications from full complexity (black line) down to 1 (purple point).

Algorithm 3 Simplification Range.

```
Data: polygonal curve $z$, base $b$, upper bound $\ell \in \mathbb{N}$
Result: sequence of simplifications $s$ with complexity $b^i$ and minimal Fréchet
        distance to $z$
$s, l \leftarrow [\,], 1$
while $b\ell > l$ do
    $s$.append($\mathbf{S}_{\#}(z, l)$)
    $l \leftarrow bl$
return $s$
```

▶ **Lemma 5.** *Computing the range of simplifications with Algorithm 3 (Simplification Range) has $O^*(m^3 \log m)$ runtime.*

Proof. Summing up the runtimes of Algorithm 2 ((1+ε)-Apx. Simplification), we get

$$O\left(\sum_{i=0}^{\log_{(1+\varepsilon)} m}\left(\log\varepsilon^{-1} m^2 \log^2 m\right) + \sum_{i=0}^{\log_{(1+\varepsilon)} m}\left(m^2 (1+\varepsilon)^i \log m\right)\right).$$

The first sum simplifies to $O(\varepsilon^{-1} \log \varepsilon^{-1} m^2 \log^3 m)$. The second sum is a simple geometric series. This saves a $\log m$ factor and resolves to $O(\varepsilon^{-1} m^3 \log m)$ time. ◀

3 Static L-budget clustering

We consider the setting of clustering curves with a given L space restriction. That is, we want to find a clustering $\mathcal{C}$ of curves c from the domain of Fréchet curves with $\sum_{c \in \mathcal{C}} \mathbf{cplx}(c) \leq L$ such that the balls $\mathbf{B}_r(c)$ with center at c and radius r cover the set of input curves Z and r is minimal. A similar setting is the k-weighted center clustering discussed by Hochbaum and Shmoys [15].

This also is a bi-criterion problem and leads to the co-problem where we have a maximum radius r and want to know the clustering with the minimal sum over the complexity of the centers. The co-problem will be used in the proofs of our clustering algorithm.

Starting with the static setting of computing a space-efficient clustering, we present an algorithm for L-budget clustering. We begin with the description of Algorithm 4 (Decide L-Budget Clustering). Here, we are given a radius r and want to decide if an L-budget clustering of the curves with $3r$ exists or if no cover with radius r exists. We now test if a clustering with radius r fits our storage in the following way. Given our r, we compute the best curve simplification with the function $\mathbf{S}_\#$ for every curve and store them in a heap. Then, we draw simplifications of uncovered curves, starting with the curve that has the least verticies. For our simplification s, we compute the ball $\mathbf{B}_{3r}(s)$ and mark all curves as covered. The newly covered curves form a cluster with s as the center. We repeat the drawing until all curves are covered or exceed the budget L. If we succeed, we get a 3-approximation; otherwise, we know that r is smaller than the optimum radius r^* and retry with a larger r.

Algorithm 4 Decide L-Budget Clustering.

```
Data: set of curves $Z$, budget $L \in \mathbb{N}$, radius $r$
Result: decision $r^* \leq 3r$ or $r^* > r$
$C, l \leftarrow [\,], 0$
heap $\leftarrow$ build_heap($\{(\mathbf{cplx}(s), s, z) \mid z \in Z, s = \mathbf{S}_\#(z, r)\}$)
while $Z \neq \emptyset$ do
    $\ell, c, z \leftarrow$ heap.pop()
    if $z \notin Z$ then continue
    $l \leftarrow l + \ell$
    if $l > L$ then return $r^* > r$
    $C \leftarrow C \cup \{c\}$
    $Z \leftarrow Z \setminus \mathbf{B}_{3r}(c)$
return $r^* \leq 3r$
```

▶ **Theorem 6.** *Algorithm 4 (Decide L-Budget Clustering) decides $r^* \leq 3r$ or $r^* > r$ for the L-budget clustering problem.*

Proof. If the algorithm returns $r^* \leq 3r$, then it covers the curves, and the complexity of the centers is lower or equal to the budget. Therefore, $3r \geq r^*$.

If the algorithm returns $r^* > r$, we look at the following auxiliary clustering problem. We start with an r and want the least complex clustering $\mathcal{C}_r^*$. With it, we show that $\mathbf{cplx}(\mathcal{C}_r^*)$ is larger than the interim clustering computed by the algorithm. The complexity of the auxiliary optimal clustering proves that given the budget L, the optimal clustering has a radius larger than r. The interim clustering with radius r covered a set of curves with balls of radius $3r$ and had a complexity l over the budget L. We charge the complexity of the interim clustering onto the optimal solution $\mathcal{C}_r^*$ of the auxiliary clustering problem to show that there cannot be a solution with the given r. We start with charging the complexity l of the interim clustering to the centers with their respective complexity. Now, every center c has a curve z we used to generate it. We call z the *witness* of c. So, we charge the complexity of the center to its witness curve. The witness curve z has to be part of a cluster in the optimal clustering $\mathcal{C}_r^*$. So, we can forward the charge of the witness curve z to the center c^* of the optimal cluster.

Now, two properties play a key role. First, because we only considered simplification with an uncovered witness, the distance between witnesses must be at least $2r$. Thus, each witness is in a different optimal cluster, and each optimal center got charged at most once. Secondly, the optimal center c^* each got charged their complexity or less, as each of our centers c is the minimal complexity curve in the r-ball $\mathbf{B}_r(z)$ around their witness z, which includes the optimal centers c^*. So, if we sum up the complexities of the charged optimal centers, we get that they are greater than the budget L. Hence, r was too small, and we get $r^* > r$. ◀

▶ **Theorem 7.** *Algorithm 4 (Decide L-Budget Clustering) with n curves of complexity m has runtime in $O((m \log^2 m + L)nm)$ where L is the complexity budget of the clustering.*

Proof. First, we compute all simplifications of Z for radius r. This takes $O(nm^2 \log^2 m)$ with the algorithm of Guibas et al. [14] in $\mathbb{R}^2$. Then we build a heap in $O(n)$ If we stop early because we reached the budget, we get that the summed complexity of the simplifications for which we computed the covering is $O(L)$. We also get that when we do not stop early and cover all curves. Therefore deciding which center covers which curve can be done naively in $O(Lnm)$ time. ◀

With the decider, we can search for the smallest r with $r^* \leq 3r$. Because we have no good upper bound, we start with an exponential search to find an upper bound. Here, we start with an initial guess r_0 and increase it by factors of 3. When we reach a cover of the curves, it also gives us a lower bound. With a binary search, we can find our smallest r up to an error of ε.

▶ **Theorem 8.** *For n curves of complexity m, the initial radius r_0, the optimal radius r^*, and L is the complexity of the clustering we can compute with Algorithm 4 (Decide L-Budget Clustering) in $O((m \log^2 m + L)nm(\log(r^*/r_0) + \log(\varepsilon^{-1})))$ time a $(3 + \varepsilon)$-approximation.*

Proof. We invoke the decider $O(\log(r^*/r_0) + \log(\varepsilon^{-1}))$ many times. This gives the runtime. The correctness follows from the decider's correctness and the binary search's precision. ◀

We gained two insights from this section, which we will use later in the dynamic setting.

- We used a witness z to prove that the center has minimal complexity in a ball of radius r around it.
- The centers have a minimum distance between them to guarantee that no optimal center gets charged twice.

4 Dynamic (k, ℓ)-clustering

Next, we introduce the *dynamic setting*, which allows us to incrementally add curves to our clustering. Formally, in the dynamic setting, we get a sequence of curves. At each step, we have the previous clustering and a new curve. We want to compute a constant factor approximation to the optimal solution of the static problem over the subset of processed curves. In other words, we want an approximation to the static clustering problem but cannot see all curves simultaneously.

Because every cluster has its complexity budget in the (k, ℓ)-clustering problem, we have no tradeoff in the complexity between clusters. We assume that all input curves are of complexity ℓ in the dynamic setting.

▶ Remark 9. This can be achieved with various simplification algorithms. For inputs up to the size of $O(k\ell)$, we can get an ℓ-simplification using the algorithm of Buchin et al. [8]. They need $O(k^2\ell^3 \log(k\ell))$ time and $O(k\ell)$ space and return an 4-approximation in the Fréchet distance with a no resource augmentation in the complexity. Depending on your setting, this is a valid simplification algorithm for larger input sizes but will need more space. In many clustering applications, the number of objects is much larger than their complexity. Hence, the complexity of an object is typically not larger than the complexity of the resulting clustering. However, if we want to process arbitrarily big input, Abam et al. [1] have a streaming algorithm for l-simplification with a competitive ratio of 2 and an $4\sqrt{2}+\varepsilon$-approximation in the Fréchet distance in $O(\ell^2/\sqrt{\varepsilon})$ space. But, of course, both simplification algorithms introduce an approximation factor in the complexity and the radius.

Assuming uniform complexity ℓ of the input curves allows us to use the "doubling algorithm" of Charikar et al. [12] in our setting of curve clustering. It computes a lower bound r and provides an upper bound αr of the optimal clustering radius r^*. McCutchen and Khuller [18] extended their algorithm to the "scaling algorithm" by allowing different approximation factors. They also devised a trick to improve the approximation factor. The "scaling algorithm" increases r multiplicatively by α only when there are more than k clusters. When r increases, the algorithm merges all clusters with a distance less than $2\alpha r$. Until the number of clusters is more than k, it tries to insert points into the existing clusters. Inserting a point into an existing cluster is possible if the distance between the center and the point is less than $\eta := 2\alpha^2/(\alpha-1)r$. So, when we merge clusters, we have to check that we do not immediately break the radius of the clusters. These constraints give us this inequality to be satisfied: $2\alpha r + \eta r \leq \eta\alpha r$. This simplifies to $1 < \alpha$. Charikar et al. also showed that the returned r is always smaller than the optimal radius r^*.

For the merging step, there are two variants. The first variant is to compute the threshold graph with a heap of all edges (using edge lengths as priority). This is the runtime-efficient implementation because there can be $O(k^2)$ many edges between the clusters. When we have a new cluster, we compute the distance of the center to all other centers and put an edge with the length as the priority into the heap. To build the t-threshold graph, we pop all edges from the heap with a length below or equal to t. The other variant computes the distance just in time. This only needs $O(k)$ space as each cluster can have that many edges but also requires re-computation of the distance each time. The algorithm would also work if we only had the nearest neighbor or range queries. There is some literature on this topic for the Fréchet distance [2, 5, 19], but we did not consider it because the bottleneck in the runtime analysis is the simplification.

We combine these properties and define a cluster as *valid* if and only if its radius is smaller than ηr. And clustering is valid if and only if all clusters are valid, $r \leq r^*$, the centers cover the pointset, the centers have distance $r\alpha$, and the clustering has at most k clusters.

We summarize the algorithm in pseudo-code as Algorithm 5 ((k,ℓ)-Scaling). We used the Python keyword *yield*. It behaves like a return and gives an output but allows the function to continue in a subsequent call at the point where it last yielded a result. Combined with *next*, this allows a nice notation of iteration over sequences or generators.

We start the runtime analysis with an update step.

▶ **Lemma 10.** *Algorithm 5 ((k,ℓ)-Scaling) computes an update step, excluding the insertions of the curves, in $O(k\ell^2\log\ell + k\log k)$ amortized time.*

Proof. The algorithm maintains a heap of the edges in order of their length. The graph can have almost $O(k^2)$ many edges. A new cluster introduces an edge to all the other clusters. To compute the Fréchet distances, we need $O(k\ell^2\log\ell)$ time. We charge each edge with $(\ell^2\log\ell + \log k)$ at the insertion into the heap. When the threshold t increases, we need all edges below t to build the threshold graph. Deletion of i edges from a heap takes $O(i\log k)$ time. However, we paid for the deletion at the insertion to get $O(k\ell^2\log\ell + k\log k)$ amortized time. ◀

We can then bound the runtime of the whole Algorithm 5 ((k,ℓ)-Scaling).

Algorithm 5 (k, ℓ)-Scaling.

```
Data: sequence of curves $Z$, number of clusters $k \in \mathbb{N}$, complexity of center $\ell \in \mathbb{N}$,
      approximation factor $\alpha$
Result: sequence of valid clustering $\mathcal{C}$ with their respected radius $r$
/* initialization and small value treatment                                   */
$\mathcal{C} \leftarrow \emptyset$
for $i \in [k]$ do
    $z \leftarrow$ next$(Z)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(z, \{z\})\}$
    yield $\mathcal{C}, 0$
$2\alpha r \leftarrow$ minimal distance of any pair in the first $k + 1$ curves
/* the core of the algorithm                                                   */
while $z \in Z$ do
    $z \leftarrow$ next$(Z)$
    if $B_{\eta r}(z) \cap \mathcal{C} =$ then $\mathcal{C} \leftarrow \mathcal{C} \cup \{z\}$
    /* merge step                                                              */
    while $|\mathcal{C}| > k$ do
        $r \leftarrow \alpha r$
        merge clusters with the $2\alpha r$-threshold graph
    yield $\mathcal{C}, r$   // generator notation from python
```

▶ **Theorem 11.** *Algorithm 5 ((k, ℓ)-Scaling) yields an 8-approximation and computes a clustering of n curves with cluster radius r^* in $O((k\ell^2 \log \ell + k \log k) \log(r^*/r_0) + kn\ell^2)$ amortized time.*

Proof. The correctness follows from McCutchen and Khuller [18] and the performance ratio is described by $\eta r/r^* \leq \eta$ and $\eta := 2\alpha^2/(\alpha - 1)$. This is at its smallest at $\alpha = 2$.

We get the number of update steps with $\log(r^*/r_0)$. This only needs to be multiplied by the runtime of the update step. After adding the number of decisions of the Fréchet distance, we need to check if a curve fits into an existing cluster. We get a runtime of $O((k\ell^2 \log \ell + k \log k) \log(r^*/r_0) + knl^2)$. ◀

We can improve the approximation factor with the trick from McCutchen and Khuller [18]. It uses multiple instances of the "scaling algorithm" with different start values. This leads to the approximation factor of $(2 + \varepsilon)$ and the runtime and space increases by $O(\varepsilon^{-1} \log \varepsilon^{-1})$.

5 Dynamic L-budget clustering

Now, we consider clustering with a fixed budget in the dynamic setting. We assume each input curve has at most complexity L, i. e. is an L-simplification if necessary. We provide an $\eta(1 + \varepsilon)$-approximation of dynamic L-budget clustering when using $(1 + \varepsilon)L$ as budget. In L-budget clustering, we can trade between cluster complexity and the number of clusters. The novel aspect of our setting is that we have differently complex centers, and the complexity of a simplification can change. Simplifying gives a new way to lower the complexity of a cluster by replacing the current center with a simplification of it. We show that there is a natural moment when we will simplify and when we will merge clusters.

For the construction, we need a witness of the minimal complexity for each cluster center and each r. We always have to guarantee a $(1+\varepsilon)$-approximation of the optimal simplification in the ball $\mathbf{B}_r$. However, we cannot compute these simplifications later and must work with the range of stored simplifications. This means we will have such simplification only up to a resource augmentation of $(1+\varepsilon)$. We build on the work of Charikar et al. [12] and McCutchen and Khuller [18], who introduce the approximation factor η. A clustering is valid if the maximum radius is smaller or equal to $\eta(1+\varepsilon)r^*$ and the sum of the complexity of the centers is smaller or equal to $(1+\varepsilon)L$. We will show that our incremental algorithm adds a curve and constructs a new valid clustering given a previously valid clustering.

In Algorithm 6 (Initialization), we get our first curve z and compute all the $(1+\varepsilon)$-simplifications with algorithm $\mathbf{S_d}$ (see Section 2). Our simplification algorithm gives the best guarantees, needing a resource augmentation of only $(1+\varepsilon)$ but works only for curves in 2D. We represent a cluster with a set of tuples of the center, the witness's influence, and the cluster radius's upper bound for each simplification. We define $\mathbf{d}_F(c_{-1}, z) := \infty$.

■ **Algorithm 6** Initialization.

```
Data: polygonal curve $z$, budget $L \in \mathbb{N}$
Result: valid clustering $\mathcal{C}$ with its respected radius $r$
/* (center, witness influence, upper bound on cluster radius)        */
$\mathcal{C} \leftarrow \{((c_i, \mathbf{d}_F(c_{i-1}, z), \mathbf{d}_F(c_i, z)) \mid c_i \in \mathit{SimplificationRange}(z, (1+\varepsilon), L)\}$
$r, l \leftarrow \mathbf{d}_F(c_0, z), L$
return $\mathcal{C}, r$
```

▶ **Lemma 12.** *Algorithm 6 (Initialization) gives a valid clustering, the runtime is in* $O(\varepsilon^{-1}(\log \varepsilon^{-1} \log^2 L + L)L^2 \log L)$ *and the space is in* $O((1+\varepsilon)L)$*.*

Proof. We get the complexity of the centers by construction. For the simplification algorithm, we already showed that this produces a $(1+\varepsilon)$-simplification to the optimal simplification. Because the distance between the curve and the simplification monotonically decreases in the number of nodes, we get that the highest complex simplification has to be as good as the optimal simplification. Thus, we get that $r \leq r^*(1+\varepsilon)$. From this, it then follows that the clustering is valid.

The runtime is dominated by the runtime of Algorithm 3 (Simplification Range). ◀

So, we can assume that we have a valid clustering for the main algorithm and show that we maintain it when we add a new curve w.

■ **Algorithm 7** L-Budget Main.

```
Data: sequence of curves $Z$, budget $L \in \mathbb{N}$
Result: sequence of valid clustering $\mathcal{C}$ with their respected radius $r$
$z \leftarrow$ next($Z$)
$\mathcal{C}, r \leftarrow$ Initialization($z, L$)
yield $\mathcal{C}, r$
for $z \in Z$ do
    $\mathcal{C} \leftarrow$ Insertion($z, \mathcal{C}, \eta r$)
    $\mathcal{C}, r \leftarrow$ Make Clustering Valid($\mathcal{C}, L, r$)
    yield $\mathcal{C}, r$
```

The structure of our Algorithm 7 (L-Budget Main) is that of the algorithm from Charikar et al. [12], but we changed the while loop to be controlled by the complexity of the clustering. The same two cases can happen.

In the first case, we can insert with Algorithm 8 (Insertion) the curve z without creating a new cluster. Then, l and r do not change.

Algorithm 8 Insertion.

```
Data: polygonal curve $z$, clustering $\mathcal{C}$, radius $r$, budget $L \in \mathbb{N}$
Result: updated clustering $\mathcal{C}$ containing $z$
/* inserting curve $z$ if possible                                   */
for $C \in \mathcal{C}$ do
    if $z \in \boldsymbol{B}_{\eta r}(C_{center})$ then
        $C \leftarrow (c_i, w_i, \max(r_i, \mathbf{d}_F(c_i, z)))$
        return $\mathcal{C}$
/* $z$ has distance $> r$ to all other cluster centers.              */
$C \leftarrow \{(c_i, \mathbf{d}_F(c_i, z), \mathbf{d}_F(c_i, z)) \mid c_i \in \mathit{SimplificationRange}(z, (1+\varepsilon), L)\}$
/* reduce the complexity of the centers and guarantee the witness    */
$C \leftarrow \mathit{ShortenCenterList}(C, r)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
return $\mathcal{C}$
```

When we insert z into a cluster, we compute the distance to all simplifications of the center and update the upper bound on the maximum radius. In the second case, we could not find a cluster for z, so it has to introduce a new one. For this, we do the same as in the initialization and remove all the centers with an upper bound lower than r up to the lowest complex one. This is done with Algorithm 9 (Shorten Center List). There has to be at least one simplification with the upper bound lower or equal to r because we assume the curve has complexity at most L and so, the curve itself can be a center. We also computed the radius r' of the by factor $(1+\varepsilon)$ more complex simplification. So, we can guarantee that the simplification is the best for any radius smaller than r' with the resource augmentation of $(1+\varepsilon)$.

Algorithm 9 Shorten Center List.

```
Data: center set $C$, radius $r$
Result: reduced center set $C$ with minimal complexity for the given $r$
$\mathit{idx} \leftarrow \max_{\{(c_i,w_i,r_i):=C[i], w_h \leq r\}} h$
return $C[\mathit{idx}:]$
```

▶ **Lemma 13.** *Algorithm 9 (Shorten Center List) reduces the list of centers to the minimal complexity given r and keeps the cluster valid, runs in place and in $O(\log\log L)$ time.*

Proof. The center list contains the optimal centers for the opening curve z up to a factor of $(1+\varepsilon)$ in complexity. Before the shortening, the curves in the cluster had at most ηr_{old} distance from the old center. The old center was in r_{old} of z, and z is in r_{new} of the new center. The radius r_{new} is at least α times the old r_{old}. Therefore, the distance to the new center is at most $(\eta + \alpha + 1)r_{old}$. This, of course, needs to be less than $\eta\alpha r_{old}$, which is true for all our $\alpha > 1$. This proves that the new center keeps the cluster valid. The next center would be farther than r to z, invalidating z as a witness.

The values of w_i have to be monotonically increasing. The list length is $O(\log L)$, and binary search inserts another log. ◀

We use Algorithm 10 (Make Clustering Valid) to deal with too complex clusterings. It starts with increasing the distance r, which allows us to simplify the cluster centers further. We finish if the total complexity is at most the budget, and the pairwise center distances are at least distance 2α. Otherwise, we merge clusters with the $2\alpha r$-threshold graph.

Algorithm 10 Make Clustering Valid.

```
Data: clustering $\mathcal{C}$, budget $L \in \mathbb{N}$, radius $r$
Result: valid clustering $\mathcal{C}$ with its respected radius $r$
while $l > (1+\varepsilon)L$ do
    $r \leftarrow \alpha r$
    /* simplifying the centers and changing the witness */
    for $C \in \mathcal{C}$ do
        Shorten Center List$(C, r)$
    /* reducing the numbers of clusters */
    merge clusters with the $2\alpha r$-threshold graph
    $l \leftarrow$ cplx$(\mathcal{C})$
return $\mathcal{C}$, $r$
```

▶ **Theorem 14.** *After Algorithm 10 (Make Clustering Valid), we have a valid clustering for the curves considered, and an outer loop needs $O(L^2)$ time.*

Proof. The algorithm covers the curves, and the complexity of the centers is, at most, the budget by construction. The maximum radius is ηr implying $r^* \leq \eta r$. We now show that $r \leq r^*$. See Figure 3 for a visualization of the flow of charge and the inter-cluster distance.

Before the last loop in the merge loop, we covered the curves with complexity over the budget and radius ηr_{old}. We will charge this onto the optimal solution to show that there cannot be a solution with the given r_{old}. We start with charging the complexity of the solution to the centers with their respective complexity. Now, every center c_i has a witness w_i who is part of its cluster and proves the minimality of the center up to the factor of $(1+\varepsilon)$. So, we charge the center's complexity times the factor of $(1+\varepsilon)$ to its witness and then to the optimal center c_i^*. We cannot charge two curves in the same optimal cluster as we merge clusters within $2\alpha r_{old}$. Hence, each optimal center got charged at most once. However, the optimal centers each got charged their complexity times the factor of $(1+\varepsilon)$ or less, as each of our centers is the minimal complexity curve in the range of the witness up to the factor of $(1+\varepsilon)$. If we now sum up the charges on the optimal centers, we get a lower bound of $(1+\varepsilon)$ times the complexity of the optimal solution. But our initial complexity is greater than $(1+\varepsilon)$ times the budget L.

Because of the size constraint, we cannot maintain the threshold graph in memory, so we need to decide each edge, which takes $O(L^2)$ time by deciding the Fréchet distance between every cluster. ◀

▶ **Lemma 15.** *Algorithm 7 (L-Budget Main) gives a valid solution for each iteration.*

Proof. If we insert a curve into a cluster, everything stays the same, inheriting the validity. If we must introduce a new cluster, the merge loop implies the solution's validity. ◀

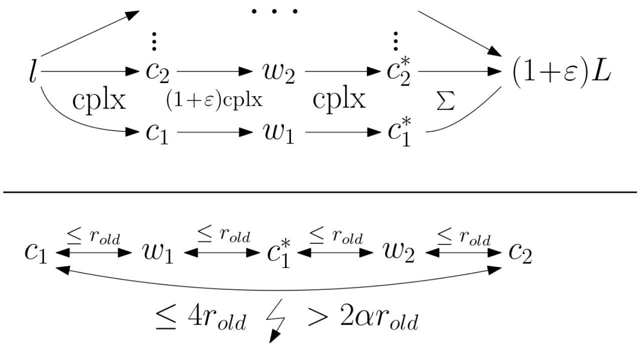

l
c_2
w_2
c_2^*
$(1+\varepsilon)L$
cplx
$(1+\varepsilon)$cplx
cplx
Σ
c_1
w_1
c_1^*
c_1 $\leq r_{old}$ w_1 $\leq r_{old}$ c_1^* $\leq r_{old}$ w_2 $\leq r_{old}$ c_2
$\leq 4r_{old}$ $> 2\alpha r_{old}$

Figure 3 On top is an illustration of the flow of charge and below inter-cluster distance requirement.

▶ **Theorem 16.** *Algorithm 7 (L-Budget Main) gives an $(8+\varepsilon)$-approximation ratio with a $(1+\varepsilon)$-resource augmentation, needs $O^*(L^3 \log L + L^2 \log(r^*/r_0))$ time and $O(L\varepsilon^{-1})$ space with r_0 being the inital and r^* the optimal radius.*

Proof. The performance ratio of the clustering is $\eta r/r^* \leq \eta$ with $\eta := 2\alpha^2/(\alpha-1)$ which is minimal at $\alpha = 2$. Multiplying the simplification error gives us $(8+\varepsilon)$.

The worst-case path through the algorithm first tests every existing cluster to see if the new curve fits. This needs multiple decisions of Fréchet distances, which take $O(L^2)$ time. Introducing a new cluster and computing the range of simplifications takes $O(\varepsilon^{-1}(\log \varepsilon^{-1} \log^2 L + L)L^2 \log L)$ time. Each outer loop of Algorithm 10 (Make Clustering Valid) needs $O(L^2)$ time. Because we will need at most $\log(r^*/r_0)$ many of them, we get $O(\varepsilon^{-1}(\log \varepsilon^{-1} \log^2 L + L)L^2 \log L + L^2 \log(r^*/r_0))$ time.

The space requirements of computing the simplification is $O(L)$, and the size of the data structure for the clusters is in $O(l\varepsilon^{-1})$. The complexity only increases if a new cluster is added, but $l \in O(L)$ because curves have at most complexity L. ◀

The trick from McCutchen and Khuller [18] can be applied because they rely on α getting big, which coincides with our restriction to not overcharge the optimal clusters – leading to a $(2+\varepsilon)$-approximation but runtime and space increases by $O(\varepsilon^{-1})$.

Reintroducing the simplification and using the algorithm from Abam et al. [1] of the curve to fit the $O(L)$ space assumption adds the space requirement of $O(L^2)$, adds the $(16\sqrt{2}+\varepsilon)$-approximation factor, and increases the resource augmentation to $(2+\varepsilon)$.

6 Experiments

We conducted experiments to compare the results of L-budget clustering to (k,ℓ)-center clustering and their respective runtimes. We use Dennis Rohde's C++ implementation for Fréchet distance, simplification, and (k,ℓ)-clustering[5], in which we added a decider for the Fréchet distance and used it in the implementation of the simplification of Agarwal et al. [3]. As dataset, we used the trajectories of homing pigeons [17] also used by [9] to demonstrate (k,ℓ)-center clustering. We computed clusterings for varying budget L and varying number n of curves from the data set. We compare our results to the best $(k,L/k)$-clustering found in a linear search over all $k \in [L]$. Figure 4 displays one result of a (k,ℓ)-center and a L-budget clustering on this data. The curves contain some outliers and a central cluster detected by

[5] `https://github.com/derohde/Fred`

both clusterings. The central cluster in the middle is quite straight with minor deviations. The outliers to the north and west make quite a detour and hence need a complex center to be represented well. Both algorithms found 5 clusters but static L-budget with half the radius ($r = 0.0097$). For this inhomogeneity in cluster complexity, L-budget clustering was designed. The (k, ℓ)-center clustering, in contrast, overfits the middle cluster without improving the clustering radius, but also has too little complexity to represent the outliers.

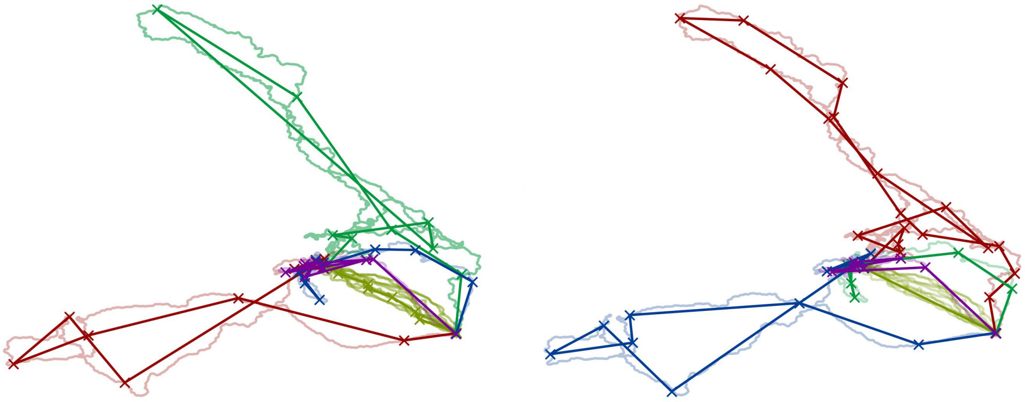


Figure 4 Example of a (k, ℓ)-center (left) and L-budget (right) clustering on a $n = 25$ curves using a budget of $L = 50$. Clusters are indicated by color and cluster centers have marked vertices.

Figure 5 shows the resulting runtimes.

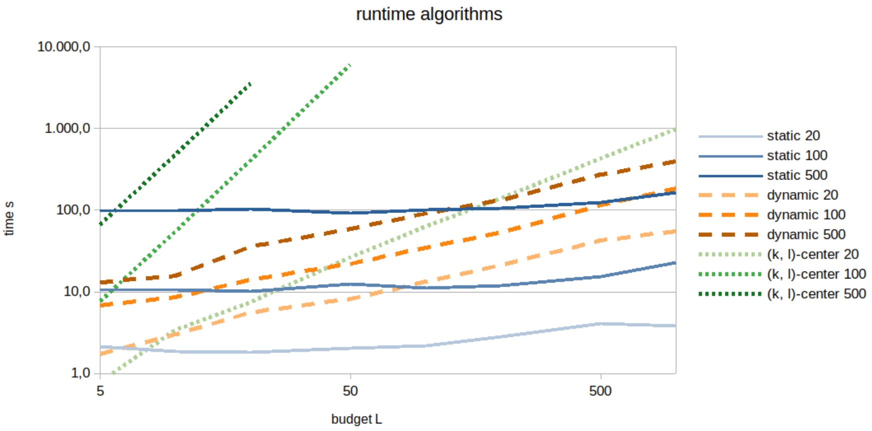


Figure 5 Shown are the runtimes of the algorithms in seconds. The runtime of some of the (k, l)-clustering experiments exceeded our time and did not finish.

The runtimes grow in L and n, seemingly slower than linear in L or n. This could be because the simplification algorithm finds long shortcuts quickly, and the clustering reduces most Fréchet distance decisions to clear-cut cases where at least one curve is of low complexity. For all interesting L and n, we also see that both implementations of L-budget clustering are much faster than using (k, ℓ)-center clustering. This is also the case for larger n or L if we compute only one instance of (k, ℓ) clustering.

To compare the results, we divide the radius found by the L-budget clustering by the best radius found by the (k, l)-center clustering in Figure 6.

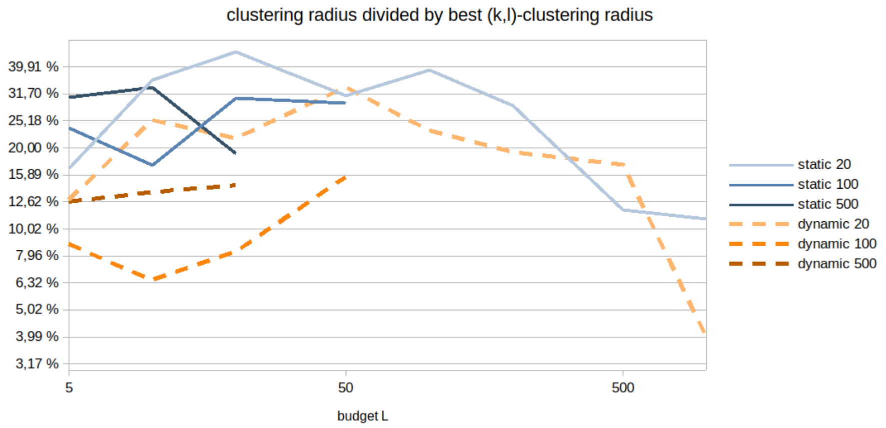


Figure 6 Shown is the radius size in percent against the best possible (k, ℓ)-center clustering. Some curves stop early because of the missing reference clustering.

These experiments suggest that L-budget clustering finds clusterings with significantly smaller clustering radius than a comparable (k, ℓ)-center clustering instance. The faster runtime enables the clustering of much larger instances.

7 Conclusion

We have addressed center clustering problems of complex objects under space constraints. We introduced the L-budget clustering problem to handle the issue and presented a 3-approximation for the static setting. We continued with the dynamic (k, ℓ)-clustering and then considered the space-restricted streaming setting. For the streaming setting, we presented an $(8 + \varepsilon)$-approximation algorithm with a $(1 + \varepsilon)$ resource augmentation over the budget L and gave a $(2 + \varepsilon)$-approximation with resource augmentation but with increased runtime and space requirements. We concluded with proof-of-concept experiments showing that the clustering is fast in practice and produces good results.

For L-budget clustering the objects to be clustered require a notion of simplification. While for curves streaming algorithms for simplification are known, it remains open is to find such an algorithm that works in $O(\ell)$ space. It would also be interesting to consider L-budget median clustering, where not the maximum but the summed radii are considered.

References

1 Mohammad Ali Abam, Mark de Berg, Peter Hachenberger, and Alireza Zarei. Streaming algorithms for line simplification. In *Proc. 23rd SoCG*, pages 175–183. ACM, 2007. doi:10.1145/1247069.1247103.

2 Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proc. 2018 SODA*, pages 898–917, 2018. doi:10.1137/1.9781611975031.58.

3 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42:203–219, 2005.

4 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995.

5 Julian Baldus and Karl Bringmann. A fast implementation of near neighbors queries for Fréchet distance (GIS cup). In *Proc. 25th ACM SIGSPATIAL*, SIGSPATIAL '17, 2017. doi:10.1145/3139958.3140062.

6 Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *Proc. 28th ACM SIGSPATIAL*, SIGSPATIAL '20, pages 99–110. ACM, 2020. doi:10.1145/3397536.3422245.

7 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Lukas Plätz, Lea Thiel, and Sampson Wong. L-Budget Clustering (SWAT 24). Software, swhId: swh:1:dir:c28806ab33cb4a564e1d492efd322b1dc32f7245, (visited on 27/05/2024). URL: https://github.com/NathenMat/SWAT24.

8 Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (k, ℓ)-center clustering for curves. In *Proc. 30th SODA*, pages 2922–2938, 2019.

9 Kevin Buchin, Anne Driemel, Natasja van de L'Isle, and André Nusser. klcluster: Center-based clustering of trajectories. In *Proc. 27th ACM SIGSPATIAL*, pages 496–499, 2019.

10 Maike Buchin, Anne Driemel, and Dennis Rohde. Approximating (k,l)-median clustering for polygonal curves. *ACM Trans. Algorithms*, 19(1), February 2023. doi:10.1145/3559764.

11 Wai-sum Chan and Francis Yuk Lun Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Int. J. Comput. Geom. Appl.*, 6:59–77, 1996.

12 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proc. 29th STOC*, pages 626–635, 1997.

13 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

14 Leonidas J Guibas, John E Hershberger, Joseph SB Mitchell, and Jack Scott Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *Int. J. Comput. Geom. Appl.*, 3(04):383–415, 1993.

15 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

16 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve — formulations and algorithms. In *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4.

17 Richard Mann, Robin Freeman, Michael Osborne, Roman Garnett, Chris Armstrong, Jessica Meade, Dora Biro, Tim Guilford, and Stephen Roberts. Objectively identifying landmark use and predicting flight trajectories of the homing pigeon using gaussian processes. *Journal of The Royal Society Interface*, 8(55):210–219, 2011.

18 Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 165–178, 2008. doi:10.1007/978-3-540-85363-3_14.

19 Majid Mirzanezhad. On approximate near-neighbors search under the (continuous) Fréchet distance in higher dimensions. *Information Processing Letters*, 183:106405, 2024. doi:10.1016/j.ipl.2023.106405.

20 Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In *Proc. 27th ESA*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:14, 2019. doi:10.4230/LIPIcs.ESA.2019.67.

21 Marc van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the hausdorff and fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020. doi:10.20382/jocg.v11i1a1.

Fixed-Parameter Tractable Certified Algorithms for Covering and Dominating in Planar Graphs and Beyond

Benjamin Merlin Bumpus
University of Florida, Gainesville, FL, USA

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Jaime Venne
Eindhoven University of Technology, The Netherlands

Abstract

For a positive real $\gamma \geq 1$, a γ-certified algorithm for a vertex-weighted graph optimization problem is an algorithm that, given a weighted graph (G, w), outputs a re-weighting of the graph obtained by scaling each weight individually with a factor between 1 and γ, along with a solution which is optimal for the perturbed weight function. Here we provide $(1+\varepsilon)$-certified algorithms for Dominating Set and H-Subgraph-Free-Deletion which, for any $\varepsilon > 0$, run in time $f(1/\varepsilon) \cdot n^{\mathcal{O}(1)}$ on minor-closed classes of graphs of bounded local tree-width with polynomially-bounded weights. We obtain our algorithms as corollaries of a more general result establishing FPT-time certified algorithms for problems admitting, at an intuitive level, certain "local solution-improvement properties". These results improve – in terms of generality, running time and parameter dependence – on Angelidakis, Awasthi, Blum, Chatziafratis and Dan's XP-time $(1+\varepsilon)$-certified algorithm for Independent Set on planar graphs (ESA2019). Furthermore, our methods are also conceptually simpler: our algorithm is based on elementary local re-optimizations inspired by Baker's technique, as opposed to the heavy machinery of the Sherali-Adams hierarchy required in previous work.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractability, certified algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.19

Funding Authors Bumpus and Jansen received funding by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

Acknowledgements We are grateful to the anonymous referees for pointing out Theorem 2.5 and making suggestions that improved the presentation of the paper.

1 Introduction

In many algorithmic contexts there is no tolerance for uncertainty. For instance, when lives are at stake (e.g. kidney exchanges [6, 15]), the difference between an approximate solution and a truly optimal one is staggering. However, finding exact optima only makes sense if the *objective function* which we are optimizing is known to accurately model the optimization problem at hand (and often this is not the case in e.g. clustering or vertex-optimization problems [13]). Indeed, if the objective function is only an *approximate* model, then there is no use in finding a true optimum relative to this objective function: after all, how could one tell whether the returned solution is "truly" optimal or if it is instead optimal simply due to the error, or noise in the objective function?

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 19; pp. 19:1–19:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Thus it is clear that, if we are optimizing an objective function which is subject to a certain degree γ of error, then it only makes sense to find *optimal* solutions when the inputs are *stable* under γ-perturbations: i.e. stable under small variations in the objective by factors which are at most our error γ. The precise formulation of the notion of γ-stability (which follows) is due to Bilu and Linial [3] and is a necessary prerequisite to the notion of *certified algorithms*, the focus of this paper.

▶ **Definition 1.1** (γ-perturbation). *For any $\gamma \in \mathbb{R}_{\geq 1}$ and set S, a* γ-perturbation *of a function $w\colon S \to \mathbb{R}$ is a function $w'\colon S \to \mathbb{R}$ satisfying $w(v) \leq w'(v) \leq \gamma \cdot w(v)$ for all $v \in S$.*

▶ **Definition 1.2** (γ-stable). *For any $\gamma \in \mathbb{R}_{\geq 1}$, a* γ-stable instance $(G, w\colon V(G) \to \mathbb{R})$ *of a vertex-minimization problem Π is an instance admitting a unique optimal solution S which remains optimal (though not necessarily unique) even under γ-perturbations of (G, w).*

Determining whether an instance is γ-stable or not can be a challenging computational task [13]. However, this is often beside the point: if we do not know whether the objective function we are optimizing has error or not, then it is enough to find a γ-approximate solution with the extra guarantee that the returned solution is optimal whenever the instance is γ-stable. Certified algorithms [3, 12, 13, 14] satisfy these requirements and more.

▶ **Definition 1.3** (Certified algorithm). *A* γ-certified solution *to an instance $(G, w\colon V(G) \to \mathbb{N})$ of a weighted vertex-optimization problem Π is a pair $(S, w'\colon V(G) \to \mathbb{R}_+)$ where w' is a γ-perturbation of w and S is an optimal solution on (G, w'). A* γ-certified algorithm *for Π is one mapping instances of Π to γ-certified solutions.*

Note that every γ-certified algorithm also serves as a factor-γ approximation algorithm [13, Thm. 5.11] for the problem, while the converse is false in general. For example, a γ-approximation for the Dominating Set problem may output a solution that fails to be inclusion-minimal, but this can never be the output of a γ-certified algorithm since there is no γ-perturbation for which such a solution is optimal.

Contributions. This paper is a foray into merging certified algorithms with parameterized complexity: here we develop FPT-time $(1 + \varepsilon)$-certified algorithms for *vertex-optimization problems* (Definition 2.2) parameterized by $1/\varepsilon$. Specifically we provide *certified* algorithms for H-Subgraph-Free-Deletion (for connected H) and Dominating Set which run in polynomial time on minor-closed classes of bounded local tree-width, which are exactly the apex-minor free graphs (Section 2). These results improve – in terms of generality, running time and parameter dependence – on Angelidakis, Awasthi, Blum, Chatziafratis and Dan's XP-time $(1 + \varepsilon)$-certified algorithm for Independent Set on planar graphs [1] which inspired the present paper.

Our results (Corollary 3.6) are obtained as by-products of our main theorem (Theorem 1.7). They draw inspiration from Baker's celebrated technique [2] and they establish FPT-time *certified* algorithms for any problem Π on such graph classes provided Π satisfies certain "local solution-improvement properties". The rest of this section will lead up to the formal statement of our main theorem by explaining precisely what these properties consist of.

The "local" nature of the "solution-improvement properties" mentioned above has to do with the operation of *m-stitching.* Intuitively, this operation consists of amending a given solution S_1 by "stitching" onto it a small, local portion of another solution S_2. In the following definition, $N_G^m[J]$ denotes the closed m-neighborhood of vertex set J (see Section 2).

▶ **Definition 1.4** (m-stitch operation). *For an integer $m \geq 0$ and vertex sets $J, S_1, S_2 \subseteq V(G)$ of a graph G, we define the m-*stitch *of S_2* onto *S_1* along *J as:*

$$S_2 \oplus^m_{G,J} S_1 := (S_1 \setminus J) \cup (S_2 \cap N^m_G[J]).$$

Naturally we refer to vertex-optimization problems whose set of feasible solutions is closed under the m-stitch operator as *m-stitchable.*

▶ **Definition 1.5** (m-stitchable). *A vertex-optimization problem Π is m-*stitchable *if, for any feasible solutions S_1 and S_2 to Π on a graph G and any vertex set $J \subseteq V(G)$, we have that $S_2 \oplus^m_{G,J} S_1$ is a feasible solution to Π on G.*

While the stitching operation seems natural, we are not aware of earlier work exploiting this idea. Our main theorem requires as a subroutine an algorithm for the following computational task for minimization problems. Roughly speaking, algorithms for the task below should be thought of as "local optimization" routines which improve any given solution S to produce solutions which are at least as good as any m-stitch onto S.

Π-m-STITCHING **Parameter:** $\mathbf{tw}(G[N^m_G[J]])$
Input: an instance $(G, w\colon V(G) \to \mathbb{N})$ to an m-stitchable vertex-optimization problem Π along with a solution S and a vertex set $J \subseteq V(G)$.
Task: find a feasible solution S' to Π on G, such that for all other feasible solutions S^*, we have $w(S') \leq w(S^* \oplus^m_{G,J} S)$.

Notice that Π-m-STITCHING is parameterized by the tree-width of the closed distance-m neighborhood of J; this restricts the exponential dependency of this local optimization task in terms of the tree-width of the closed m-neighborhood of J.

Finally, we can state our main result (Theorem 1.7) which, sweeping some details under the rug, can be thought of as a way of turning any algorithm for Π-m-STITCHING into an FPT-time *certified* algorithm for Π whenever we can quickly guess at least one feasible solution (Definition 1.6).

▶ **Definition 1.6** (Guessable). *We say that a vertex-optimization problem Π is* guessable *if there is an algorithm that outputs a feasible solution (with no requirement for optimality) in polynomial-time.*

▶ **Theorem 1.7** (main). *Let $\mathcal{G}$ be a minor-closed graph class whose local tree-width is bounded above by a linear function of the form $g\colon r \mapsto \lambda r$ (where $r \in \mathbb{N}$) for some given, fixed $\lambda \in \mathbb{R}$. If Π is a vertex-minimization problem such that:*

- *Π is guessable and m-stitchable for some $m \in \mathbb{N}$, and*
- *there exists an algorithm $\mathcal{A}$ which solves Π-m-STITCHING in time $f(t) \cdot |V(G)|^{\mathcal{O}(1)}$, where $t = \mathbf{tw}(G[N^m_G[J]])$ and f is some computable function;*

then, for each $\varepsilon > 0$ there is a $(1+\varepsilon)$-certified algorithm for Π which runs in time $f(\lambda m/\varepsilon) \cdot |V(G)|^{\mathcal{O}(1)}$ on any input $(G, w\colon V(G) \to \mathbb{N})$ with $G \in \mathcal{G}$ and polynomially-bounded weights.

We note that Theorem 1.7 also applies to the *complementary* maximization problem (see Section 5 for the formal definition) of any minimization problem Π as above. This observation will furthermore allow us to obtain a $2^{\mathcal{O}(1/\varepsilon)} \cdot n^{\mathcal{O}(1)}$-time certified algorithm for the *maximum independent set* problem (with polynomially bounded integer weights), which improves on the algorithm with running time $n^{\mathcal{O}(1/\varepsilon)}$ by Angelidakis, Awasthi, Blum, Chatziafratis and Dan [1]. Apart from being more efficient and more general, our algorithm is also conceptually simpler. It relies on repeated improvement of a solution in bounded-tree-width subgraphs, rather than the technical machinery of the Sherali-Adams hierarchy employed in earlier work.

Organization. After establishing some preliminary background and notation in Section 2, we will show in Section 3 how to apply our main theorem to obtain certified algorithms for H-Subgraph-Free-Deletion and Dominating Set. The main theorem itself (Theorem 1.7) is instead proved later on in Section 4. We discuss our algorithmic results and their application to complementary maximization problems in Section 5, which is also where we pose open questions as an invitation to further work.

2 Preliminaries

We follow the convention that zero is a natural number. We only consider finite, simple, and undirected graphs, which consist of a vertex set $V(G)$ and edge set $E(G) \subseteq \binom{V(G)}{2}$. For $m \in \mathbb{N}$, the *closed* m-neighborhood $N_G^m[X]$ of a vertex subset $X \subseteq V(G)$ in G is defined inductively as $N_G^m[X] := N_G[N_G^{m-1}[X]]$ where $N_G^1[X] = N_G[X] = \{y \in V(G) \mid \exists x \in X \colon \{x, y\} \in E(G)\} \cup X$. The *open* m-neighborhood $N_G^m(X)$ is defined as $N_G^m(X) := N_G^m[X] \setminus X$. The *tree-width* [8] of a graph G is denoted $\mathbf{tw}(G)$. The *diameter* of a connected graph G, which is defined as the maximum number of edges on any shortest path, is denoted by $\mathsf{diam}(G)$.

Throughout this paper we will always assume that weight functions are polynomially bounded in the size of the graph; i.e. we always consider weight functions of the form $w \colon V(G) \to \{0, \dots, |V(G)|^{\mathcal{O}(1)}\}$. This restriction is crucial to obtaining polynomial-time algorithms for the vertex-optimization problems (defined below) considered in this paper.

▶ **Definition 2.1.** *A* vertex-subset property $\mathcal{P}$ *assigns to each graph G the subset $\mathcal{P}(G) \subseteq 2^{V(G)}$ of vertex sets that satisfy property $\mathcal{P}$ on G. We say that a set $S \subseteq V(G)$ is* feasible *for $\mathcal{P}$ on G when $S \in \mathcal{P}(G)$.*

▶ **Definition 2.2** (vertex-optimization)**.** *A* vertex-optimization problem Π *is any pair of the form $(\mathcal{P}, \mathsf{goal})$ consisting of a vertex-subset property $\mathcal{P}$ and a function* $\mathsf{goal} \in \{\min, \max\}$*. The task of Π is to find some vertex subset $\hat{S} \in \mathcal{P}(G)$ such that $w(\hat{S}) = \mathsf{goal}_{S \in \mathcal{P}(G)} w(S)$. We call Π a* vertex-minimization problem *if* $\mathsf{goal} = \min$ *and a* vertex-maximization problem *otherwise.*

Our main algorithmic theorems concern algorithms running in minor-closed classes of (linearly) bounded local tree-width. We recall these notions below (where $d(x, y)$ denotes the usual shortest-paths distance metric on graphs).

▶ **Definition 2.3** (local tree-width)**.** *Given a graph G, the* local tree-width of G *is the map*

$$\mathbf{loctw}^G : \mathbb{N} \to \mathbb{N} \quad \textit{where} \quad \mathbf{loctw}^G : \delta \mapsto \max_{x \in V(G)} \mathbf{tw}\big(G[\{y \in V(G) : d(x, y) \leq \delta\}]\big).$$

▶ **Definition 2.4** (graphs of bounded local tree-width)**.** *A graph class $\mathcal{C}$ has* bounded local tree-width *if there is a function $f \colon \mathbb{N} \to \mathbb{R}$ such that $\mathbf{loctw}^G(r) \leq f(r)$ for all $(G, r) \in \mathcal{C} \times \mathbb{N}$. Furthermore, if there is a $\lambda \in \mathbb{R}$ such that the function f above can be defined as $f \colon r \mapsto \lambda r$, then we say that $\mathcal{C}$ has λ-linear local tree-width.*

An *apex graph* is a graph that can be made planar by removing a single vertex. Eppstein [9] proved that a minor-closed class of graphs has bounded local tree-width if and only if it excludes an *apex* graph as a minor. Demaine and Hajiaghayi [7, Theorem 4.1] proved that any apex-minor-free graph has *linear* local tree-width, thereby leading to the following equivalence.

▶ **Theorem 2.5** ([7, 9])**.** *A minor-closed graph class $\mathcal{C}$ has bounded local tree-width if and only if it has λ-linear local tree-width for some $\lambda \in \mathbb{R}$.*

For any graph-theoretic notation not defined here, we refer the reader to Diestel's textbook [8]; similarly for standard notation in parameterized complexity theory see Cygan et al.'s textbook [4].

3 Applications of Theorem 1.7

Here we will apply Theorem 1.7 to obtain FPT-time certified algorithms for Dominating Set and H-Subgraph-Free-Deletion. We recall the definitions of these problems below.

H-Subgraph-Free-Deletion (for a fixed connected graph H)
Input: a vertex-weighted graph $(G, w \colon V(G) \to \mathbb{N})$.
Task: find a minimum-weight subset $X \subseteq V(G)$ such that no subgraph of $G - X$ is isomorphic to H.

Dominating Set
Input: a vertex-weighted graph $(G, w \colon V(G) \to \mathbb{N})$.
Task: find a minimum-weight subset $X \subseteq V(G)$ such that $V(G) = N_G[X]$.

To apply our main theorem to these problems we need to show that they are guessable (which is trivially true: $V(G)$ is feasible solution), m-stitchable for some appropriate choices of m, and that there are FPT-time algorithms for the relevant stitching problems parameterized by tree-width. We begin with stitchability.

▶ **Lemma 3.1.** *Dominating Set is 2-stitchable while H-Subgraph-Free-Deletion is $\mathsf{diam}(H)$-stitchable for any connected graph H.*

Proof. Consider any three vertex sets $J, S_1, S_2 \subseteq V(G)$.

First we consider Dominating Set. If S_1 and S_2 are dominating sets, then so is $S_2 \oplus^2_{G,J} S_1$: any vertex of $V(G) \setminus N_G[J]$ is dominated by $S_1 \setminus J$ while vertices of $N_G[J]$ are dominated by $S_2 \cap N^2_G[J]$. Note that we need to consider the 2-neighborhood of J, since there might be vertices in $N_G(J)$ that S_1 dominates from within J but that S_2 dominates from $N^2_G(J)$.

Now we turn our attention to H-Subgraph-Free-Deletion. Let $h \colon H \hookrightarrow G$ be an H-subgraph of G. If S_1 and S_2 are H-hitting sets and $h(H)$ is not hit by $S_1 \setminus J$, then $V(h(H) \cap J) \neq \emptyset$. Hence $h(H)$ lies entirely in $N_G^{\mathsf{diam}(H)}[J]$, since H is connected. But then $h(H)$ is hit by $S_2 \cap N_G^{\mathsf{diam}(H)}[J]$. Thus $S_2 \oplus_{G,J}^{\mathsf{diam}(H)} S_1$ is an H-hitting set. ◀

Next we give algorithms for H-Subgraph-Free-Deletion-Stitching (Lemma 3.2) and Dominating Set-Stitching (Lemma 3.3).

▶ **Lemma 3.2.** *Let H be a fixed connected graph and $m := \mathsf{diam}(H)$. Given any algorithm $\mathcal{A}$ which solves H-Subgraph-Free-Deletion on any vertex-weighted instance $(G, w \colon V(G) \to \mathbb{N})$ in time $f(\mathbf{tw}(G)) \cdot |V(G)|^c$ for some function f and constant c, the following algorithm solves H-Subgraph-Free-Deletion-m-Stitching in time $f(\mathbf{tw}(Q)) \cdot |V(G)|^c$ where $Q = G[N^m_G[J]]$.*

- ***Algorithm Stitch-H-Del***
- ***Input:*** *a vertex-weighted graph $(G, w \colon V(G) \to \mathbb{N})$, a vertex set $J \subseteq V(G)$, and a feasible solution S_1 on G, i.e., graph $G - S_1$ has no subgraph isomorphic to H.*
- ***Output:*** *a feasible solution S' on G, such that for all other feasible solutions S^*, we have $w(S') \leq w(S^* \oplus^m_{G,J} S_1)$.*

1. *Let $F = G[N^m_G[J] \setminus (S_1 \setminus J)]$.*
2. *Let S_2 be the output of the algorithm $\mathcal{A}$ on input $(F, w|_{V(F)})$.*
3. *Return $S_2 \oplus^m_{G,J} S_1$ if $w(S_2 \oplus^m_{G,J} S_1) < w(S_1)$ and S_1 otherwise.*

Proof. The running time is clearly dominated by that of $\mathcal{A}$. Notice, towards proving correctness, that $S_2 \oplus^m_{G,J} S_1$ is feasible: the set $S_2' := S_2 \cup (V(G) \setminus V(F)) \cup (S_1 \cap N^m_G(J))$ is an H-deletion set in G and thus, by the m-stitchability of H-Subgraph-Free-Deletion and definition of F, we find that $S_2' \oplus^m_{G,J} S_1 = S_2 \oplus^m_{G,J} S_1$ is an H-deletion set.

Now assume by way of contradiction that there is a feasible solution S_3 such that $w(S_2 \oplus^m_{G,J} S_1) > w(S_3 \oplus^m_{G,J} S_1)$. Then we have that:

$$\begin{aligned}
&w|_{V(F)}(S_2) = w(S_2) = && \text{(since } S_2 \subseteq V(F))\\
&= w(S_2 \cap N^m_G[J]) && \text{(since } V(F) \subseteq N^m_G[J])\\
&= w(S_1 \setminus J) + w(S_2 \cap N^m_G[J]) - w(S_1 \setminus J) && \\
&= w\big((S_1 \setminus J) \cup (S_2 \cap N^m_G[J])\big) - w(S_1 \setminus J) && \text{(since } V(F) \cap (S_1 \setminus J) = \emptyset)\\
&= w(S_2 \oplus^m_{G,J} S_1) - w(S_1 \setminus J) && \text{(by def. of stitch)}\\
&> w(S_3 \oplus^m_{G,J} S_1) - w(S_1 \setminus J) && \text{(by assumption on } S_3)\\
&= w\big((S_1 \setminus J) \cup (S_3 \cap N^m_G[J])\big) - w(S_1 \setminus J) && \text{(by def. of stitch)}\\
&= w\big((S_3 \cap N^m_G[J]) \setminus (S_1 \setminus J)\big) && (w(A \cup B) - w(A) = w(B \setminus A))\\
&= w\big(S_3 \cap (N^m_G[J] \setminus (S_1 \setminus J))\big) && ((A \cap B) \setminus C = A \cap (B \setminus C))\\
&= w(S_3 \cap V(F)) && \text{(by def. of } F)\\
&= w|_{V(F)}(S_3 \cap V(F))
\end{aligned}$$

which contradicts the fact that S_2 was optimal on $(F, w|_{V(F)})$ since $S_3 \cap V(F)$ is an H-deletion set on F (because the property of being an H-deletion set is closed under induced subgraphs). ◀

Since – in contrast to H-deletion sets – the property of being a dominating set is not closed under taking induced subgraphs, our algorithm for Dominating Set-2-Stitching will require slightly different ideas from those in Lemma 3.2. Indeed, rather than finding a solution that is locally optimal after the removal of $S_1 \setminus J$ (as we did in the previous lemma), we will instead find a minimum-weight set that dominates all vertices which are not already dominated by $S_1 \setminus J$.

▶ **Lemma 3.3.** *Given any algorithm $\mathcal{A}$ which solves Dominating Set on any vertex-weighted instance $(G, w\colon V(G) \to \mathbb{N})$ in time $f(\mathbf{tw}(G)) \cdot |V(G)|^c$ for some function f and constant c, the following algorithm solves Dominating Set-2-Stitching in time $f(\mathbf{tw}(Q)) \cdot |V(G)|^c$ where $Q = N^2_G[J]$.*

- ***Algorithm Stitch-Dom-Set***
- ***Input:*** *a vertex-weighted graph $(G, w\colon V(G) \to \mathbb{N})$, a vertex set $J \subseteq V(G)$, and a dominating set S_1 on G.*
- ***Output:*** *a dominating set S' in G, such that, for all other dominating sets S^*, we have $w(S') \leq w(S^* \oplus^2_{G,J} S_1)$.*

1. *Define F to be the graph obtained from $G[N^2_G[J]]$ by adding a new vertex $\mathfrak{f}$ with $N_F(\mathfrak{f}) := N_G(N_G[J])$. (Vertex $\mathfrak{f}$ is adjacent to the vertices at distance exactly two from J in G.)*

2. *Define* $w_F : V(F) \to \mathbb{N}$ *as*

$$w_F\colon x \mapsto \begin{cases} 0 & \text{if } x \in (S_1 \setminus J) \cup \{\mathfrak{f}\} \\ w(x) & \text{otherwise.} \end{cases} \tag{1}$$

3. *Let* $S_2 = S_2' \setminus \{\mathfrak{f}\}$ *where* S_2' *is the output of algorithm* $\mathcal{A}$ *on input* (F, w_F)*.*
4. *Return* $S_2 \oplus^2_{G,J} S_1$ *if* $w(S_2 \oplus^2_{G,J} S_1) < w(S_1)$ *and* S_1 *otherwise.*

Proof. The proofs of the running-time bound and feasibility of $S_2 \oplus^2_{G,J} S_1$ are virtually identical to Lemma 3.2. Notice that we can assume that $S_1 \cap N^2_G(J) \subseteq S_2$ since, by its definition in the algorithm above, $w_F(S_1 \setminus J) = 0$. The rest of the proof will make use of the following auxiliary definition.

▶ **Definition 3.4.** *Given a vertex subset* X *of a graph* H*, an* $\overline{X}$-dominating set *in* H *is a set* $S \subseteq V(H)$ *such that* $y \in N_H[S]$ *for all* $y \in V(H) \setminus X$*.*

We claim that S_2 is a minimum-weight $\overline{N_F(\mathfrak{f})}$-dominating set in $G[N^2_G[J]]$ with respect to weight function w. To see this, first of all note that S_2 is a $\overline{N_F(\mathfrak{f})}$-dominating set since it dominates every vertex in $F - \mathfrak{f} - N_F(\mathfrak{f}) = G[N^2_G[J]] - N_F(\mathfrak{f})$: the vertex $\mathfrak{f}$ that is removed from the dominating set S_2' in F only dominates vertices of $\{\mathfrak{f}\} \cup N_F(\mathfrak{f})$, so the rest is dominated by $S_2' \setminus \{\mathfrak{f}\} = S_2$. Now suppose by way of contradiction that there is an $\overline{N_F(\mathfrak{f})}$-dominating set D in $G[N^2_G[J]]$ with $w(D) < w(S_2)$. Then, since $w_F(\mathfrak{f}) = 0$, $w_F(D \cup \{\mathfrak{f}\}) = w(D) < w(S_2) = w_F(S_2 \cup \{\mathfrak{f}\})$ which contradicts the fact that $S_2 \cup \{\mathfrak{f}\}$ is a minimum dominating set on (F, w_F).

Now take any dominating set S_3 in G. Observe that $(S_3 \oplus^2_{G,J} S_1) \cap N^2_G[J]$ is an $\overline{N_F(\mathfrak{f})}$-dominating set in $G[N^2_G[J]]$ and thus, by what we just showed,

$$w\big((S_3 \oplus^2_{G,J} S_1) \cap N^2_G[J]\big) \geq w(S_2). \tag{2}$$

Using the fact that $S_1 \setminus J = (S_1 \setminus N^2_G[J]) \cup (S_1 \cap N^2_G(J))$, we thus have:

$$\begin{aligned} w(S_3 \oplus^2_{G,J} S_1) &= w\big((S_1 \setminus J) \cup (S_3 \cap N^2_G[J])\big) && \text{(by def. of stitch)} \\ &= w(S_1 \setminus N^2_G[J]) + w\big((S_1 \cap N^2_G(J)) \cup (S_3 \cap N^2_G[J])\big) && \text{(by fact above)} \\ &= w(S_1 \setminus N^2_G[J]) + w\big((S_3 \oplus^2_{G,J} S_1) \cap N^2_G[J]\big) && \text{(by def. of stitch)} \\ &\geq w(S_1 \setminus N^2_G[J]) + w(S_2) && \text{(by Inequality (2))} \\ &= w(S_1 \setminus N^2_G[J]) + w\big((S_1 \cap N^2_G(J)) \cup S_2\big) && \text{(since } S_1 \cap N^2_G(J) \subseteq S_2\text{)} \\ &= w(S_1 \setminus N^2_G[J]) + w\big((S_1 \cap N^2_G(J)) \cup (S_2 \cap N^2_G[J])\big) && \text{(since } S_2 \subseteq V(F) \setminus \{\mathfrak{f}\}\text{)} \\ &= w\big((S_1 \setminus J) \cup (S_2 \cap N^2_G[J])\big) && \text{(since } N^2_G[J] \setminus N^2_G(J) = J\text{)} \\ &= w(S_2 \oplus^2_{G,J} S_1). && \text{(by def. of stitch)} \end{aligned}$$

◀

From what we've seen so far in this section, we have that both Dominating Set and H-Subgraph-Free-Deletion are stitchable (by Lemma 3.1). Thus, since these problems lie in FPT parameterized by tree-width (as we recall for convenience in Theorem 3.5 below), we can conclude by Lemmas 3.2 and 3.3 that both H-Subgraph-Free-Deletion-Stitching and Dominating Set-Stitching also lie in FPT.

▶ **Theorem 3.5** ([4, 5])**.** *Given any weighted graph* $(G, w\colon V(G) \to \mathbb{N})$ *with tree-width at most* k*, we can solve:*

- *H-Subgraph-Free-Deletion in time* $2^{O(k)} \cdot |V(G)|^{O(1)}$ *when* H *is a clique [5],*

- *H-Subgraph-Free-Deletion in time $2^{O(k)^{\mu^*(H)}\log k} \cdot |V(G)|^{O(1)}$ when H is a connected graph that is not a clique [5], and*
- *Dominating Set in $2^{O(k)} \cdot |V(G)|^{O(1)}$ [4, page 176],*

where $\mu^*(H)$ for a connected graph H denotes the maximum, over all connected vertex sets $A \subseteq V(H)$ satisfying $N_H(N_H[A]) \neq \emptyset$, of the quantity $|N_H(A)|$.

Furthermore, since both H-Subgraph-Free-Deletion and Dominating Set are guessable, we can apply Theorem 3.5 to obtain (Corollary 3.6) polynomial time, *certified* algorithms for H-Subgraph-Free-Deletion and Dominating Set on minor-closed classes with bounded local tree-width.

▶ **Corollary 3.6.** *For any minor-closed graph class $\mathcal{C}$ of λ-linear local tree-width and each $\varepsilon > 0$ there are $(1+\varepsilon)$-certified algorithms solving Dominating Set and H-Subgraph-Free-Deletion whenever the input is of the form (G, w) with $G \in \mathcal{C}$ and $w \colon V(G) \to \mathbb{N}$ a polynomially-bounded weight function. Furthermore these algorithms respectively admit the following worst-case running-time bounds:*

- *$2^{O(\lambda/\varepsilon)} \cdot |V(G)|^{O(1)}$ in the case of H-Subgraph-Free-Deletion when H is a clique,*
- *$2^{O(k)^{\mu^*(H)}\log k} \cdot |V(G)|^{O(1)}$ (where $\mu^*(H)$ is the constant of Theorem 3.5 and k equals $\mathsf{diam}(H)\lambda/\varepsilon$) in the case of H-Subgraph-Free-Deletion for a connected graph H that is not a clique, and*
- *$2^{O(2\lambda/\varepsilon)} \cdot |V(G)|^{O(1)}$ in the case of Dominating Set.*

4 Proving Theorem 1.7

The proof of Theorem 1.7 takes inspiration from Baker's technique [2] for designing polynomial-time approximation schemes on planar graphs. It will occur in three steps: we will outline the algorithm in Section 4.2.1, show that it has the desired running time in Section 4.2.2, and prove its correctness in Section 4.2.3. However, before doing so we shall briefly establish a few useful definitions in Section 4.1 which will streamline the presentation of what follows.

4.1 Definitions for Theorem 1.7

Throughout we assume all graphs are *connected* unless stated otherwise and we denote any interval $\{a, a+1, \dots, b\}$ in $\mathbb{Z}$ as $[a, b]$; furthermore we denote by $\iota_{a,b}$ the obvious inclusion $\iota_{a,b} : [a, b] \hookrightarrow \mathbb{Z}$ given by $\iota_{a,b}(i) = i$ for $a \le i \le b$. Often, we shall refer to $\iota_{a,b}$ itself as an *interval.*

▶ **Definition 4.1** (m-boundary of an interval). *Given any integer $m \ge 1$, we define the* left and right m-boundaries *of any interval $\iota_{a,b}$ to respectively be the intervals*

$$\delta^L_m(\iota_{a,b}) : [a-m, a-1] \hookrightarrow \mathbb{Z} \quad \text{and} \quad \delta^R_m(\iota_{a,b}) : [b+1, b+m] \hookrightarrow \mathbb{Z}.$$

We define the closed m-boundary *of $\iota_{a,b}$ as $\delta_m[\iota_{a,b}] : [a-m, a] \cup [a, b] \cup [b, b+m] \hookrightarrow \mathbb{Z}$ while the* open m-boundary *of $\iota_{a,b}$ is defined as $\delta_m(\iota_{a,b}) := \delta^L_m(\iota_{a,b}) \cup \delta^R_m(\iota_{a,b})$.*

Recall that, given any vertex v in a graph G, the *eccentricity* of v in G is the maximum length of a shortest path from v to any other vertex. Here we will denote this as $\hat{\epsilon}(v, G)$ or simply as $\hat{\epsilon}(v)$ if G is understood from context.

Ripples

When one drops a stone in a pond, an outward-radiating rippling ring of waves forms where the stone hit the surface of the water. In analogy to this phenomenon, we shall now define an r-*ripple*[1] in a graph as the sets of vertices (the waves, as it were) at fixed distances from some given vertex r.

▶ **Definition 4.2** (r-ripple). *Given a vertex r in a graph G, we call the function*

$$\rho_r : \mathbb{Z} \to 2^{V(G)} \quad \textit{where} \quad \rho_r : i \mapsto \{x \in V(G) : d(r,x) = i\}$$

the r-ripple *in G. The vertex-subsets that make up a ripple will be referred to as* waves*: for any integer i, we define the i-th wave in ρ_r to be the set $\rho_r(i)$.*

In Definition 4.2 above, we call the vertex r the *center* of the ripple. If the center of the ripple is understood from context, then we simply denote the ripple as ρ. Notice that the i-th wave of a ripple will always be empty if i is negative or if it is greater than the eccentricity $\hat{\epsilon}(r)$ of the center of the ripple; one should think of such as "dummy" indices. Our choice to represent ripples as functions with domain $\mathbb{Z}$ is simply for notational convenience; indeed, one could instead restrict these functions to simply view any r-ripple as a function with domain $\{0, \ldots, \hat{\epsilon}(r)\}$.

▶ **Definition 4.3** ((a,b)-subripple; see also Figure (1)). *Let ρ be an r-ripple in a graph G and $\iota_{a,b} : [a,b] \hookrightarrow \mathbb{Z}$ be an interval. We define the* (a,b)-subripple *in ρ to be the function $\rho_{a,b} : [a,b] \to 2^{V(G)}$ defined as the composite $\rho_{a,b} := \rho \circ \iota_{a,b}$. The* width *of a finite subripple is the number of waves it consists of (e.g. the width of an (a,b)-ripple is $|b - a + 1|$).*

For any graph G and (a,b)-subripple of an r-ripple ρ in G, the graph $G[\bigcup_{a \le i \le b} \rho(i)]$ is a subgraph of the graph G' obtained from G by contracting all vertices v with $d_G(r,v) < a$ into r. As $\bigcup_{a \le i \le b} \rho(i)$ is contained in $N_{G'}^{b-a+1}[r]$, the tree-width of $G[\bigcup_{a \le i \le b} \rho(i)]$ is bounded in terms of the local tree-width of G'. Whenever G comes from a minor-closed graph class $\mathcal{C}$ of bounded local tree-width, we have $G' \in \mathcal{C}$ which ensures a bound on its local tree-width. This yields the following observation.

▶ **Observation 4.4** ([11]). *Let $\mathcal{C}$ be a minor-closed class of graphs which has λ-bounded linear local tree-width. If ρ is an r-ripple in a graph G belonging to $\mathcal{C}$, then the tree-width of any (a,b)-subripple of ρ is upper-bounded by $\mathbf{tw}(G[\bigcup_{a \le i \le b} \rho(i)]) \le \lambda(|b - a + 1|)$.*

We note that one can of course use composition to generalize the notion of m-boundaries from intervals (Definition 4.1) to (sub)ripples. Indeed, we overload the notation so that, for example, the left m-boundary of any (a,b)-subripple $\rho_{a,b}$ is denoted $\delta_m^L(\rho_{a,b})$ and it is defined as the composite $\rho \circ \delta_m^L(\iota_{a,b})$. One can similarly define right, open and closed m-boundaries of any (a,b)-subripple.

In the rest of this section we shall make two further definitions related to simple constructions with ripples: *modular slices* of a ripple (Definition 4.5) and the *difference of a ripple and a modular slice* (Definition 4.6). Since both of these concepts are very easy to grasp visually, we defer their formal definitions and instead define them first "by picture" in Figure (1) below. The notation $S_2^{\bmod 4}(i)$ in Figure (1) denotes the *i-th modular slice* (an evenly spaced sequence of subripples with a given start-index i) and the *difference* $\rho \ominus S_2^{\bmod 4}(i)$ is simply the sequence of subripples that is "left-over" from ρ after we remove the modular slice $S_2^{\bmod 4}(i)$.

[1] This is sometimes referred to as a "*layering*" in the literature.

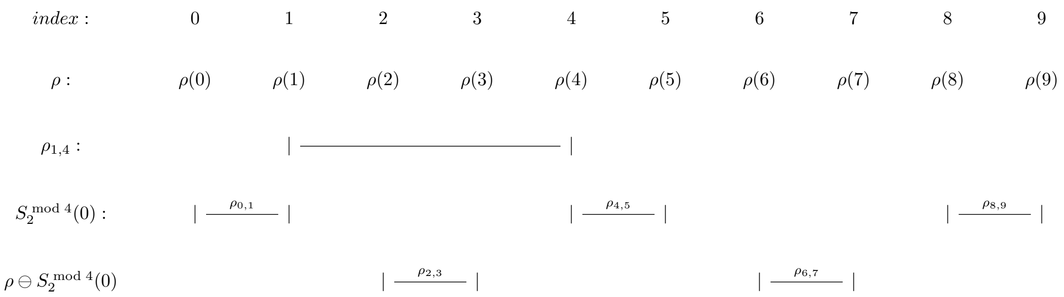


Figure 1 Illustration of subripples, modular slices, and remainders.

▶ **Definition 4.5** (modular slices). *Let ρ be an r-ripple in a graph G. For any integers s and k with $1 \leq s \leq k$ and any integer $i \in \{0, 1, \ldots, k-1\}$ define the set*

$$\mathbb{Z}_s^k(i) := \bigcup_{j \in \mathbb{Z}} \{j \cdot k + i, j \cdot k + i + 1, \ldots, j \cdot k + i + s - 1\}$$

and consider its obvious inclusion $\iota_{s,k,i} \colon \mathbb{Z}_s^k \hookrightarrow \mathbb{Z}$. We call the map $S_s^{\mathrm{mod}\ k}(i) \colon \mathbb{Z}_s^k \to 2^{V(G)}$ defined as the composite $S_s^{\mathrm{mod}\ k} := \rho \circ \iota_{s,k,i}$ the i-th modular k-slice of width s *in ρ.*

▶ **Definition 4.6** (Remainder). *Let s, k, and i be integers with $1 \leq s \leq k$ and $0 \leq i \leq k-1$. Let ρ be an r-ripple in a graph G and $S_s^{\mathrm{mod}\ k}(i) \colon \mathbb{Z}_s^k \hookrightarrow 2^{V(G)}$ be a modular k-slice of width s in ρ. Letting $\overline{\iota_{s,k,i}} \colon \mathbb{Z} \setminus \mathbb{Z}_s^k \hookrightarrow \mathbb{Z}$ be the obvious inclusion of $\mathbb{Z} \setminus \mathbb{Z}_s^k$ into $\mathbb{Z}$, we define the* remainder *of $S_s^{\mathrm{mod}\ k}(i)$ in ρ, denoted as $\rho \ominus S_s^{\mathrm{mod}\ k}(i)$, to be the composite $\rho \circ \overline{\iota_{s,k,i}}$.*

To ease legibility and conciseness, throughout this document, we shall treat any subripple (resp. modular slice or difference thereof) as the union of all of its constituent waves whenever performing set-theoretic operations. For example, for any subset X of $V(G)$ and any (a, b)-subripple $\rho_{a,b}$, we shall simply write $X \cap \rho_{a,b}$ instead of $X \cap (\bigcup_{a \leq i \leq b} \rho(i))$.

Pigeonhole arguments on ripples and their modular slices

We will conclude this preliminary section by proving two auxiliary lemmas (Lemmas 4.7 and 4.9) which will be of use to us in the proof of Theorem 1.7. Intuitively, the next lemma says that for any weighted vertex set X in a graph G, when considering the modular k-slice of width s of a ripple in G, there will be an offset i such that the vertices contained in its waves at offset i contribute at most an $\frac{s}{k}$ fraction of the total weight of X.

▶ **Lemma 4.7.** *Let ρ be an r-ripple in a weighted graph $(G, w \colon V(G) \to \mathbb{R}_+)$ and let $k \geq 1$ be an integer. For any vertex set $X \subseteq V(G)$ and integer $1 \leq s \leq k$, there exists an integer $i \in \{0, \ldots, k-1\}$ such that $S_s^{\mathrm{mod}\ k}(i)$ satisfies $w(X \cap S_s^{\mathrm{mod}\ k}(i)) \leq \frac{s}{k} w(X)$.*

Proof. Seeking a contradiction, assume no such index i exists and hence conclude, by summing over each index $0 \leq j \leq k-1$, that

$$\sum_{0 \leq j \leq k-1} w(X \cap S_s^{\mathrm{mod}\ k}(j)) > \sum_{0 \leq j \leq k-1} \frac{s}{k} w(X) = s \cdot w(X) \frac{1}{k} = s \cdot w(X). \tag{3}$$

However, since each non-empty wave of the ripple ρ is counted by exactly s out of k of the modular k-slices in the sum above, we can contradict the strictness of Inequality (3) by verifying that $\sum_{0 \leq j \leq k-1} w(X \cap S_s^{\mathrm{mod}\ k}(j)) = \sum_{j' \in \mathbb{Z}} s \cdot w(X \cap \rho(j')) = s \cdot w(X)$. ◀

Now consider, for example, the difference $\rho \ominus S_2^{\mathrm{mod}\ 4}(0)$ shown in Figure 1 above. It is easy to see (by inspection of the figure) that, after taking the closed 1-boundary of every subripple in $\rho \ominus S_2^{\mathrm{mod}\ 4}(0)$, the domains of the resulting subripples partition[2] the domain of ρ. In this case, we say simply that the extended subripples *partition* ρ. We can state this more generally as the following observation.

▶ **Observation 4.8.** *Let $0 \leq 2s \leq k$ be integers and ρ be an r-ripple in a graph G. If $S_{2s}^{\mathrm{mod}\ k}$ is a modular k-slice of width $2s$ in ρ, then $\left(\delta^s[\rho_{a,b}]\right)_{\rho_{a,b} \in \rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)}$ partitions ρ for any i.*

Observation 4.8 together with a pigeon-hole-like argument similar to that of the proof of Lemma 4.7 yields the following lemma. It applies to two vertex subsets I and S in a weighted graph G, which will later correspond to the solution S found by our algorithm and a solution I that is purported to be better. The lemma applies when the weight of $S \setminus I$ within the waves of a ripple ρ, on vertices outside the i-th modular k-slice of a certain width $2s$, is strictly larger than the weight of $I \setminus S$. It guarantees the existence of a *single* subripple $\rho_{a,b}$ with the following special property: the weight of $S \setminus I$ inside the subripple $\rho_{a,b}$ is strictly larger than the weight of $I \setminus S$ inside the *extended* subripple $\rho_{a-s,\ldots,b+s}$. We will later use this lemma to argue that under certain conditions, a local optimization step can strictly improve the solution.

▶ **Lemma 4.9.** *Let ρ be an r-ripple in a weighted graph $(G, w\colon V(G) \to \mathbb{R}_+)$ and $S_s^{\mathrm{mod}\ k}$ be the modular k-slice of width $2s$ in ρ. If there are sets $S, I \subseteq V(G)$ and an index i such that*

$$w((S \setminus I) \cap (\rho \ominus S_{2s}^{\mathrm{mod}\ k}(i))) > w(I \setminus S), \tag{4}$$

then there exists $\rho_{a,b} \in \rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)$ satisfying $w((S \setminus I) \cap \rho_{a,b}) > w((I \setminus S) \cap \delta_s[\rho_{a,b}])$.

Proof. Seeking a contradiction, assume no such subripple $\rho_{a,b}$ exists; i.e. assume that

$$w((S \setminus I) \cap \rho_{a,b}) \leq w((I \setminus S) \cap \delta_s[\rho_{a,b}]) \tag{5}$$

for all subripples $\rho_{a,b}$ in the difference $\rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)$. Then we have

$$\begin{aligned} w((S \setminus I) \cap (\rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)) &= \sum_{\rho_{a,b} \in \rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)} w((S \setminus I) \cap \rho_{a,b}) && \text{(by definition)} \\ &\leq \sum_{\rho_{a,b} \in \rho \ominus S_{2s}^{\mathrm{mod}\ k}(i)} w((I \setminus S) \cap \delta_s[\rho_{a,b}]) && \text{(by Equation (5))} \\ &= w(I \setminus S) \end{aligned}$$

where the last equality holds because $\left(\delta^s[\rho_{a,b}]\right)_{\rho_{a,b} \in \rho \ominus S_s^{\mathrm{mod}\ k}(i)}$ is a partition of ρ (by Observation 4.8). However, this contradicts Inequality (4) as desired. ◀

4.2 Proof of Theorem 1.7

We are now finally ready to prove Theorem 1.7: we will first describe (Section 4.2.1) the algorithm mentioned in the statement of Theorem 1.7; then we shall establish its running time guarantees (Section 4.2.2) and finally its correctness (Section 4.2.3). Using the existence of an algorithm for Π-m-Stitching, it will be easy to describe our $(1+\varepsilon)$-certified algorithms; the main challenge lies in the proof that the solution it outputs is optimal for a $(1+\varepsilon)$-perturbation of the input.

[2] Notice that, although it is not drawn, $\rho(0)$ is in the 1-boundary of the element $\rho_{-1,-2}$ of $\rho \ominus S_2^{\mathrm{mod}\ 4}(0)$.

4.2.1 The algorithm

Throughout the rest of the proof of Theorem 1.7 we shall let Π be a vertex-optimization problem as given in the statement of Theorem 1.7, i.e. it satisfies the following:

1. Π is guessable and m-stitchable for some given constant $m \in \mathbb{N}$, and
2. there exists an algorithm $\mathcal{A}$ that solves Π-m-Stitching in time $f(t) \cdot |V(G)|^{\mathcal{O}(1)}$, where $t = \mathbf{tw}(G[N_G^m[J]])$ and f is some computable function.

In what follows, given any feasible solution S on an instance (G, w) of Π and any (a, b)-subripple $\rho_{a,b}$ of an r-ripple ρ, we denote by $\mathcal{A}((G, w), S, \rho_{a,b})$ the output of running the algorithm $\mathcal{A}$ for Π-m-Stitching on inputs (G, w), the vertex set J of $\rho_{a,b}$, and the solution S. The algorithm is defined as follows.

- **Algorithm StitchAndCertify**
- **Input:** a (connected) vertex-weighted graph $(G, w \colon V(G) \to \mathbb{N})$ and $\varepsilon > 0$.
- **Output:** a vertex set $\hat{S} \subseteq V(G)$ and $(1 + \varepsilon)$-perturbation w' of w such that $\hat{S}$ is an optimal solution for Π on (G, w').
  1. Let $r \in V(G)$ be an arbitrary vertex and let ρ be the r-ripple in G.
  2. Let $\hat{S}$ be a feasible solution for Π on G (obtained by leveraging the guessability of Π; c.f. Definition 1.6).
  3. Let $k = \lceil \frac{2m}{\varepsilon} \rceil + 2m$.
  4. **While** there exists an (a, b)-subripple $\rho_{a,b}$ of width $k - 2m$ such that
  $$w\big(\mathcal{A}((G, w), \hat{S}, \rho_{a,b})\big) < w(\hat{S}) \tag{6}$$
     - then replace $\hat{S}$ with $\mathcal{A}((G, w), S, \rho_{a,b})$.
  5. **Otherwise**, return $(\hat{S},\ w' : V(G) \to \mathbb{R}_+)$ where w' is defined as
  $$w' \colon x \mapsto \begin{cases} w(x) & \text{if } x \in \hat{S} \\ (1 + \varepsilon) w(x) & \text{otherwise.} \end{cases} \tag{7}$$

4.2.2 Running time

Recall that the parameter for Π-Stitching is the tree-width of the closed m-neighborhood of the vertex set J along which we stitch. Each call to the algorithm $\mathcal{A}$ for Π-Stitching (Inequality 6) made inside StitchAndCertify runs on a subripple of width $k - 2m$. Hence the closed m-neighborhood of the subgraph along which we stitch is contained in $\delta^m[\rho_{a,b}]$, which is a subripple of width $k - 2m + 2m = k$. By Proposition 4.4 we know that $G[\delta^m[\rho_{a,b}]]$ has tree-width at most λk. These observations allow us to upper-bound the running time of each call to $\mathcal{A}$ by $f\big(\mathbf{tw}(G[\delta^m[\rho_{a,b}]])\big) \cdot |V(G)|^{\mathcal{O}(1)} \le f\big(\lambda k\big) \cdot |V(G)|^{\mathcal{O}(1)}$.

Now, since all other lines of the algorithm clearly take polynomial time (recall that Π is guessable by Definition 1.6), the calls to $\mathcal{A}$ dominate the running time. Note that, for $W = \max_{x \in V(G)} w(x)$, the number of iterations in which we find a strictly better solution is bounded by $W \cdot n$: the weight of the initial solution is at most $W \cdot n$, all weights are integers, and the value cannot improve to below 0. Thus, since $k \in \mathcal{O}(m/\varepsilon)$ the entire algorithm runs in time at most $W \cdot f(\mathcal{O}(\lambda m/\varepsilon)) \cdot |V(G)|^{O(1)}$, as desired.

4.2.3 Proof of correctness

To prove that StitchAndCertify is indeed a $(1 + \varepsilon)$-certified algorithm, we must show that the output $(\hat{S}, w' \colon V(G) \to \mathbb{R}_+)$ consists of an optimal solution $\hat{S}$ for Π on the instance (G, w') (which is clearly a $(1 + \varepsilon)$-perturbation of the input (G, w)). It is easy to see that $\hat{S}$ is indeed a solution to Π, since it is initialized as a feasible solution and is only replaced by the output of $\mathcal{A}$, which is also a feasible solution by definition. Hence it suffices to prove optimality.

Assume, by way of contradiction, that $\hat{S}$ is not optimal. Then there is a solution I for Π on (G, w') such that

$$w'(\hat{S}) > w'(I) \implies w'(\hat{S} \setminus I) > w'(I \setminus \hat{S}) \implies w(\hat{S} \setminus I) > (1+\varepsilon) w(I \setminus \hat{S}) \tag{8}$$

(by the definition of w'; see Equation (7)). The remainder of this proof will rest on the following claim which states that not only is $w(\hat{S} \setminus I) > (1+\varepsilon)w(I \setminus \hat{S})$, but moreover, there exists an index of the modular slice whose intersection with $\hat{S} \setminus I$ has greater weight than that of $I \setminus \hat{S}$.

▷ Claim 4.10. For the given $\hat{S}$ and I, there exists an index $0 \leq i \leq k-1$ such that the preconditions of Lemma 4.9 are met for $s = m$; stating this explicitly, there is a choice of i such that $w((\hat{S} \setminus I) \cap (\rho \ominus S_{2m}^{\text{mod } k}(i))) > w(I \setminus \hat{S})$.

Claim 4.10 (whose proof we defer to the end of this section) enables us to apply Lemma 4.9 in order to find a "heavy" subripple; i.e. a subripple $\rho_{a,b} \in \rho \ominus S_{2m}^{\text{mod } k}(i)$ satisfying

$$w((\hat{S} \setminus I) \cap \rho_{a,b}) > w((I \setminus \hat{S}) \cap \delta_m[\rho_{a,b}]). \tag{9}$$

To aid the upcoming derivation, we now argue that the sets $A := \hat{S} \setminus \rho_{a,b}$, $B := (I \setminus \hat{S}) \cap \delta_m[\rho_{a,b}]$, and $C := I \cap \hat{S} \cap \rho_{a,b}$, form a partition of $D := (\hat{S} \setminus \rho_{a,b}) \cup (I \cap \delta_m[\rho_{a,b}])$. To see this, observe first that A, B, C are disjoint: $A \cap C = \emptyset$ since C lives inside $\rho_{a,b}$ but A outside; $A \cap B = \emptyset$ since A lives inside $\hat{S}$ but B outside; and $B \cap C = \emptyset$ since C lives inside $\hat{S}$ but B outside. To establish that A, B, C partition D, it therefore suffices to argue their union covers D. For this, the crucial insight is that those vertices of $I \cap \hat{S} \cap \delta_m[\rho_{a,b}]$ that are not contained in $I \cap \hat{S} \cap \rho_{a,b}$, belong to $\hat{S} \setminus \rho_{a,b}$ and therefore to A.

Using this property we now deduce

$$\begin{aligned}
w(\hat{S}) &= w\big(\hat{S} \setminus \rho_{a,b}\big) + w\big(\hat{S} \cap \rho_{a,b}\big) \\
&= w\big(\hat{S} \setminus \rho_{a,b}\big) + w\big((\hat{S} \setminus I) \cap \rho_{a,b}\big) + w\big(I \cap \hat{S} \cap \rho_{a,b}\big) \\
&> w\big(\hat{S} \setminus \rho_{a,b}\big) + w\big((I \setminus \hat{S}) \cap \delta_m[\rho_{a,b}]\big) + w\big(I \cap \hat{S} \cap \rho_{a,b}\big) && \text{(by Inequality (9))} \\
&= w\big((\hat{S} \setminus \rho_{a,b}) \cup (I \cap \delta_m[\rho_{a,b}])\big) && (A, B, C \text{ partition } D) \\
&\geq w\big((\hat{S} \setminus \rho_{a,b}) \cup (I \cap N_G^m[\rho_{a,b}])\big) && (\delta_m[\rho_{a,b}] \supseteq N_G^m[\rho_{a,b}]) \\
&= w(I \oplus_{G,\rho_{a,b}}^m \hat{S}) && \text{(by Definition 1.4).}
\end{aligned}$$

But then this means that the m-stitch of I onto $\hat{S}$ along the subripple $\rho_{a,b}$ of width $k-2m$ yields a solution (since Π is m-stitchable) whose weight under w is strictly better than $\hat{S}$. By definition of Π-m-Stitching, the output of $\mathcal{A}$ for the subripple $\rho_{a,b}$ is at least as good, thus satisfying Inequality (6) of StitchAndCertify. We conclude that the algorithm cannot possibly have terminated. Thus, since we have found our desired contradiction, all that remains to be done is to prove Claim 4.10.

Proof of Claim 4.10. Applying Lemma 4.7 on (G, w) and ρ with $X = (\hat{S} \setminus I)$ we obtain an index i such that modular k-slice $S_{2m}^{\text{mod } k}(i)$ satisfies

$$w((\hat{S} \setminus I) \cap S_{2m}^{\text{mod } k}(i)) \leq \frac{2m}{k} w(\hat{S} \setminus I). \tag{10}$$

Now observe that, by the definition of $\ominus$ (Definition 4.6), we have

$$\begin{aligned} w((\hat{S} \setminus I) \cap (\rho \ominus S_{2m}^{\bmod k}(i))) &= w(\hat{S} \setminus I) - w((\hat{S} \setminus I) \cap S_{2m}^{\bmod k}(i)) \\ &\geq w(\hat{S} \setminus I) - \frac{2m}{k} w(\hat{S} \setminus I) && \text{(by Inequality (10))} \\ &= \frac{k-2m}{k} w(\hat{S} \setminus I) \\ &> \frac{k-2m}{k}(1+\varepsilon) w(I \setminus \hat{S}) && \text{(by Inequality (8)).} \end{aligned}$$

Notice that, since we defined k as $k = \lceil \frac{2m}{\varepsilon} \rceil + 2m$ (Line 3 of StitchAndCertify), we must have $\varepsilon \geq \frac{2m}{k-2m}$. Combining this observation with our derivation above, we obtain precisely the desired inequality of Claim 4.10 as follows.

$$\begin{aligned} w((\hat{S} \setminus I) \cap (\rho \ominus S_{2m}^{\bmod k}(i))) &> \frac{k-2m}{k}(1+\varepsilon) w(I \setminus \hat{S}) \qquad \text{(from the derivation above)} \\ \geq \frac{k-2m}{k}\left(1 + \frac{2m}{k-2m}\right) w(I \setminus \hat{S}) &= w(I \setminus \hat{S}). \end{aligned}$$

This concludes the proof of Claim 4.10 and hence also the proof of Theorem 1.7. ◁

5 Discussion

Our main theorem allows us to obtain FPT-time *certified* algorithms for vertex-minimization problems such as H-Subgraph-Free-Deletion and Dominating Set (Corollary 3.6 of Theorem 1.7). However, as mentioned in Section 1, our results also apply to the *complementary* maximization problems simply by virtue of being certified algorithms. Inspired by Makarychev and Makarychev's notation [13], we define the notion of the *complementary* problem as follows.

▶ **Definition 5.1.** *Fix a vertex-minimization (resp. maximization) problem* Π *as in Definition 2.2. The* complementary vertex-maximization *(resp.* minimization*) problem is obtained by equipping the vertex-subset property that encodes feasibility for* Π *with following maximization (resp. minimization) objective: for any given vertex-weighted instance* $(G, w\colon V(G) \to \mathbb{N})$ *find a set* $S \subseteq V(G)$ *such that* $w(V(G) \setminus S)$ *is maximum (resp. minimum) subject to the requirement that* S *be feasible with respect to* Π.

Makarychev and Makarychev [13] discuss many examples of complementary problems; perhaps the prototypical example pair is Vertex Cover and Independent Set: every minimum vertex cover S in a graph G corresponds to a maximum independent set $V(G) \setminus S$.

Notice that one can deduce [13, Theorem 5.11] that any certified algorithm $\mathcal{A}$ for some problem Π is also an approximation algorithm for the complementary problem to Π; this is recalled below for completeness. Furthermore, it is easy to show a polynomial-time equivalence between certified algorithms for a problem and its complementary problem.

▶ **Theorem 5.2** ([13])**.** *If* $\mathcal{A}$ *is a* γ*-certified algorithm for a vertex-minimization (resp. maximisation) problem* Π*, then* $\mathcal{A}$ *is a* γ*-approximation algorithm for both* Π *and its complementary vertex-maximization (resp. minimization) problem.*

Recalling that Vertex Cover is just K_2-Deletion, we find that Corollary 3.6 yields a polynomial-time $(1+\varepsilon)$-certified algorithm for Independent Set on minor-closed graph classes of bounded local tree-width. This improves – in terms of generality and running time – on the XP-time $(1+\varepsilon)$-certified algorithm for Independent Set on planar graphs which was due to Angelidakis, Awasthi, Blum, Chatziafratis and Dan [1].

Further questions

As we mentioned in Section 1, any certified algorithm $\mathcal{A}$ for a problem Π happens to also be an approximation algorithm for both Π and its complementary problem $\Pi^{\mathfrak{c}}$. Thus a natural direction for future work is to seek $(1+\varepsilon)$-certified algorithms for other problems that admit efficient polynomial-time approximation schemes. In contrast, by Theorem 5.2, whenever either Π or $\Pi^{\mathfrak{c}}$ do not admit any EPTAS, then the question that we just posed is clearly not a viable direction for further work. Thus for such problems such as WEIGHTED PLANAR FEEDBACK VERTEX SET (for which, for instance, bidimensional techniques [10] do not apply) even simply finding XP-time certified algorithms can be a fruitful direction of research.

Another interesting direction for future research concerns the range of weight values. Our running-time analysis crucially relies on the assumption that the weights are non-negative integers of value at most $n^{\mathcal{O}(1)}$: this property ensures that the local search terminates after $n^{\mathcal{O}(1)}$ improvements. The algorithm by Angelidakis et al. [1] also requires polynomially bounded weights. Is it possible to give FPT-time $(1+\varepsilon)$-certified algorithms on inputs whose weights are encoded in $n^{\mathcal{O}(1)}$ bits, but may have value $2^{\Omega(n)}$?

References

1 H. Angelidakis, P. Awasthi, A. Blum, V. Chatziafratis, and C. Dan. Bilu-linial stability, certified algorithms and the independent set problem. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.7.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, January 1994. doi:10.1145/174644.174650.

3 Y. Bilu and N. Linial. Are stable instances easy? *Comb. Probab. Comput.*, 21(5):643–660, September 2012. doi:10.1017/S0963548312000193.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

5 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Information and Computation*, 256:62–82, 2017. doi:10.1016/j.ic.2017.04.009.

6 M. Delorme, S. García, J. Gondzio, J. Kalcsics, D. Manlove, and W. Pettersson. New algorithms for hierarchical optimisation in kidney exchange programmes. *Technical report ERGO 20–005, Edinburgh Research Group in Optimization*, 2020. URL: https://optimization-online.org/2020/10/8058/.

7 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 840–849. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982919.

8 R. Diestel. *Graph theory*. Springer, 2010. ISBN:9783642142789.

9 David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/S004530010020.

10 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 748–759. SIAM, 2011. doi:10.1137/1.9781611973082.59.

11 M. Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23:613–632, 2000. doi:10.1007/s00493-003-0037-9.

12 T. Hazan, G. Papandreou, and D. Tarlow. *Bilu-Linial Stability*, pages 375–400. The MIT Press, 2016.

13 Konstantin Makarychev and Yury Makarychev. Perturbation resilience. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 95–119. Cambridge University Press, 2020. doi:10.1017/9781108637435.008.

14 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu-linial stable instances of max cut and minimum multiway cut. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 890–906. SIAM, 2014. doi:10.1137/1.9781611973402.67.

15 David Manlove. *Algorithmics of matching under preferences*, volume 2. World Scientific, 2013.

Sparsity-Parameterised Dynamic Edge Colouring

Aleksander B. G. Christiansen
Technical University of Denmark, Lyngby, Denmark

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Juliette Vlieghe
Technical University of Denmark, Lyngby, Denmark

Abstract

We study the edge-colouring problem, and give efficient algorithms where the number of colours is parameterised by the graph's arboricity, α. In a dynamic graph, subject to insertions and deletions, we give a deterministic algorithm that updates a proper $\Delta + O(\alpha)$ edge colouring in $\text{poly}(\log n)$ amortized time. Our algorithm is fully adaptive to the current value of the maximum degree and arboricity.

In this fully-dynamic setting, the state-of-the-art edge-colouring algorithms are either a randomised algorithm using $(1+\varepsilon)\Delta$ colours in $\text{poly}(\log n, \epsilon^{-1})$ time per update, or the naive greedy algorithm which is a deterministic $2\Delta - 1$ edge colouring with $\log(\Delta)$ update time.

Compared to the $(1+\varepsilon)\Delta$ algorithm, our algorithm is deterministic and asymptotically faster, and when α is sufficiently small compared to Δ, it even uses fewer colours. In particular, ours is the first $\Delta + O(1)$ edge-colouring algorithm for dynamic forests, and dynamic planar graphs, with polylogarithmic update time.

Additionally, in the static setting, we show that we can find a proper edge colouring with $\Delta + 2\alpha$ colours in $O(m \log n)$ time. Moreover, the colouring returned by our algorithm has the following local property: every edge uv is coloured with a colour in $\{1, \max\{\deg(u), \deg(v)\} + 2\alpha\}$. The time bound matches that of the greedy algorithm that computes a $2\Delta - 1$ colouring of the graph's edges, and improves the number of colours when α is sufficiently small compared to Δ.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases edge colouring, arboricity, hierarchical partition, dynamic algorithms, amortized analysis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.20

Related Version This paper is based on the master's thesis of Juliette Vlieghe, supervised by Eva Rotenberg and Aleksander B. G. Christiansen and defended on 30th June 2023.
Master's Thesis: `https://doi.org/10.13140/RG.2.2.18471.52648`

Funding This work was supported by the VILLUM Foundation grant VIL37507 "Efficient Recomputations for Changeful Problems".

Acknowledgements We thank Jacob Holm for his interest in this work, and for comments and improvements to an earlier version of this manuscript. We also thank an anonymous reviewer for their highly detailed feedback and suggestions for improvement.

1 Introduction and related work

When working on rapidly evolving, large scale graphs, algorithms need to adapt to the change in data quickly. The dynamic model is interested in maintaining some property in a graph undergoing edge insertions and/or deletions, and has led to many fast algorithms, with polylogarithmic update and query time, in particular through the use of amortized analysis.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 20; pp. 20:1–20:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Graph colouring is a family of fundamental problems with many applications in computer science. We study the edge-colouring problem: the goal is to assign edges colours such that edges sharing an endpoint are coloured differently. This problem has implications in resource allocation and scheduling, for example to allocate bandwidth in an optical network [17].

A *C-edge colouring* of a graph $G = (V, E)$ can be represented by a function $f : E \rightarrow \{1, ..., C\}$, and the smallest palette size C for which there exists a proper C edge colouring is called the edge chromatic number of G, denoted χ'. If Δ is the maximum degree of G, then the edge chromatic number is clearly at least Δ. Vizing [30] proved that G can always be coloured with $\Delta + 1$ colours. On the other hand, Holyer showed that it is NP-complete to determine the edge chromatic index of an arbitrary graph [21], and the problem remains NP-complete even for cubic graphs. A $(\Delta(uv) + C)$-edge colouring is a proper colouring of the graph where each edge uv receives a colour from $\{1, ..., \Delta(uv) + C\}$. Here $\Delta(uv) = \max\{d(u), d(v)\}$.

Vizing's proof is constructive and suggests a way to extend a proper partial colouring to a larger subgraph by recolouring $O(\Delta + n)$ edges. Furthermore, the colour changes can be performed in polynomial time. However, so far the fastest algorithms for statically $(\Delta + 1)$ edge colouring a graph spend $O(m\sqrt{n})$ [28] or $\tilde{O}(m\Delta)$ [18] time. For certain graphs, faster algorithms are known [4, 8].

It is interesting to see whether one can reduce the running time by slightly increasing the palette size. This line of research has been pursued before. In particular, the problem of $2\Delta - 1$ edge colouring can be solved greedily, yielding static algorithms running in near-linear time [14] and dynamic algorithms with $O(\log \Delta)$ update time [6]. In this dynamic setting, there are known algorithms [13, 16] that achieve a randomized $(1 + \epsilon)\Delta$ colouring in $\text{poly}(\log n, \epsilon^{-1})$ time, with $\epsilon > 0$ by Duan, He, and Zhang [16], and later Christiansen [13].

In the distributed setting, Chang, He, Li, Pettie and Uitto [11] designed a randomized $\Delta + O(\sqrt{\Delta})$ edge colouring in $\text{poly}(\log n)$ rounds based on the Lovasz Local Lemma and Barenboim, Elkin and Maimon [2] describe a simple deterministic distributed algorithm for $\Delta + O(\alpha)$ colouring in polylogarithmic time. There are many more papers achieving different trade-offs between time and palette size. See for instance [3, 15, 19, 29] for different trade-offs in the distributed setting, or the papers [11, 19] for a more extensive discussion.

For some algorithmic problems, especially ones where recourse is an important part of the running time of an algorithm, or the recourse is of interest on its own, the best known analysis follows a specific proof strategy; *"solution oblivious analysis"*. By solution oblivious, we mean that we do not only give guarantees against worst case input graphs at each step of the algorithm, we furthermore always, when analysing the next step, are robust against an adversary changing the solution to the worst-possible solution before every update. Examples of such analysis include the analysis of the SAP protocol for maintaining maximum matchings in bipartite graphs of Bernstein, Holm and Rotenberg [5], the analysis of recourse in the edge-colouring algorithms of Bernshteyn [3], Christiansen [13], and Duan, He and Zhang [16], and the analysis of the fully-dynamic out-orientation scheme due to Brodal & Fagerberg [10], in which a potential function bounds the reorientations by comparing to an *existing* (but not necessarily efficient) algorithm that augments paths whose lengths are bounded in an oblivious manner.

For the edge-colouring problem, an interesting lower bound has been proved by Chang, He, Li, Pettie and Uitto: they show that there exists a graph, and a partial colouring of this graph with a single uncoloured edge, such that to colour this edge, one needs to recolour a subgraph of diameter $\Omega(\frac{\Delta}{c}\log(\frac{cn}{\Delta}))$. This is then also a lower bound on the number of edges that need to be recoloured. This means that if we restrict ourselves to solution oblivious analysis, a dynamic algorithm with polylogarithmic update time will need a palette of size at least $\Delta + O\left(\frac{\Delta}{poly(\log n)}\right)$. The analysis of Christiansen [13], and Duan, He and Zhang [16] for their $(1+\epsilon)\Delta$ dynamic edge-colouring algorithms are solution oblivious, which results in algorithms that have a polynomial dependency on ϵ^{-1}.

Whether one can design an algorithm with poly log update time that only uses $\Delta+O(\Delta^{1-\varepsilon})$ colours for some constant $\varepsilon > 0$ remains a fundamental open problem. Improved results for special classes of graphs, like forests, planar graphs or sparse graphs, also receive attention in the community. In the static setting, it was shown by Bhattacharya, Costa, Panski and Solomon [8] that one can compute a $(\Delta + 1)$ edge colouring in $\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$-time. Here α is the *arboricity* of the graph, and it is equal to the smallest number of forests needed to cover the edges of a graph. It is within a factor of 2 of other sparsity measures like the maximum subgraph density and the degeneracy. Many other problems like, for instance, maintaining dynamic matchings [22, 26], maintaining a dynamic data structure that can answer adjacency queries efficiently [10] and maintaining a maximal independent set [25] also have solutions that run faster in graphs with low arboricity.

Our contribution. A natural question is therefore to ask if one can further reduce the palette size in dynamic graphs that are at all times sparse. In this paper, we show that this is the case. More specifically, we show that there exists a dynamic algorithm that can maintain an edge colouring with only $\Delta + O(\alpha)$ colours in poly-logarithmic update time. Since the arboricity can be as large as $\frac{\Delta}{2}$, this is not always an improvement, however for many graph classes like forests, planar graphs, and graphs with constant arboricity, the number of colours used is significantly reduced compared to other efficient dynamic edge-colouring algorithms.

Tables 1, 2 and 3 summarise the results mentioned above and are not a comprehensive overview of the state of the art.

Table 1 A comparison of static edge-colouring algorithms.

Palette size	Time	Notes	Reference
Δ	$O(m \log \Delta)$	bipartite multigraph	[14]
$2\Delta(uv) - 1$	$O(m \log \Delta)$		[6]
$\Delta + 1$	$O(m\sqrt{n})$	randomised	[28]
$\Delta + 1$	$\tilde{O}(m\Delta)$		[18]
$\Delta + 1$	$\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$		[8]
$\Delta(uv) + 1$	$O(n^2\Delta)$		[13]
$\Delta(uv) + 2\alpha - 2$	$O(m \log \Delta)$		new

Independent work. In independent and concurrent work, Bhattacharya, Costa, Panski, and Solomon also maintain a $\Delta + O(\alpha)$ edge colouring in amortised polylogarithmic time per insertion or deletion [7].

Table 2 A comparison of dynamic edge-colouring algorithms. If G goes through a sequence of insertions and deletions $G_1 ... G_T$, $\Delta_{max} = \underset{1 \leq t \leq T}{max} \Delta(G_t)$ is the maximum Δ on all graphs in the sequence. α_{max} is defined similarly.

Palette size	Update time	Notes	Reference
$2\Delta(uv) - 1$	$O(\log \Delta)$	worst case	[6]
$(1+\epsilon)\Delta$	$O(\log^9 n \log^6 \Delta/\epsilon^6)$	worst-case, randomised	[13]
$(1+\epsilon)\Delta$	$O(\log^8 n/\epsilon^4)$	amortized, randomised $\Delta \in \Omega(\log^2 n/\epsilon^2)$	[16]
$\Delta_{max} + O(\alpha_{max})$	$O(\log n \log \Delta_{max})$	amortized	new
$\Delta(uv) + O(\alpha)$	$O(\log^2 n \log \alpha_{max} \log \alpha \log \Delta_{max})$	amortized	new

Table 3 State-of-the-art for edge-colouring algorithms in the LOCAL model.

Palette size	Rounds	Notes	Reference
$\Delta + O(\alpha)$	$O(\sqrt{\alpha} \log n)$	LOCAL model	[2]
$\Delta + 1$	$poly(\Delta, \log n)$	LOCAL model	[3]

1.1 Notations

In this paper, we focus on simple graphs. According to many definitions of edge colouring, an edge from a vertex to itself can not be coloured.

Let $G = (E, V)$ be the undirected input graph and H a subgraph of G. Define $\Gamma_H(v)$ to be the neighbourhood of v with respect to a graph H, and given a subset of vertices $U \subseteq V$, define $\Gamma_U(v)$ to be the neighbourhood of v with respect to the subgraph induced by U in G. Later, we use $N_H(v)$ to refer to a data structure containing $\Gamma_H(v)$. Define $\deg_H(v)$ to be the degree of v with respect to a graph H. Formally, given $H \subseteq G$ and $U \subseteq V$:

$$\begin{aligned} \Gamma_H(v) &= \{u \in V \mid (u,v) \in E(H)\} \\ \Gamma_U(v) &= \{u \in U \mid (u,v) \in E(G)\} \\ \deg_H(v) &= |\Gamma_H(v)| \\ \deg_U(v) &= |\Gamma_U(v)| \end{aligned}$$

Let G go through a sequence of insertions and deletions $G_1 ... G_T$. At any iteration t, we denote by $\Delta_t(uv)$ the maximum degree of the endpoints of the edge uv and Δ_t the maximum degree of the graph considered. Δ_{max} is the maximum Δ on all graphs in the sequence. Formally, at iteration t:

$$\begin{aligned} \Delta_t(uv) &= \max\{\deg_{G_t}(u), \deg_{G_t}(v)\} \\ \Delta_t &= \Delta(G_t) \\ \Delta_{max} &= \underset{1\leq t\leq T}{max} \Delta(G_t) \end{aligned}$$

If the context is not ambiguous, we drop the subscript and write $\Delta, \Delta(uv)$ to refer to the maximum degree and the maximum degree between u and v in the current iteration.

Arboricity. The arboricity α of a graph $G = (V, E)$ is defined as:

$$\alpha = \underset{U\subseteq V,\ |U|>1}{max} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$$

On a more intuitive level, the arboricity can also be defined as the smallest number α such that the edges of the graph can be partitioned in α forests. The two definitions are equivalent by Nash-Williams theorem [24]. A relevant consequence is that there exists an orientation of G where each vertex has at most α out-neighbours.

H-partition. Let $H = \{H_1, ...H_k\}$ be a partition of the vertex set $V(G)$. If a vertex v is in H_i, we say that the level of v is $l(v) = i$. We denote by $Z_i = \bigcup_{j\geq i} H_j$ the vertices in levels i and above (Figure 1). We may abuse these notations and use H_i, Z_i to refer to the subgraph induced in G by those sets.

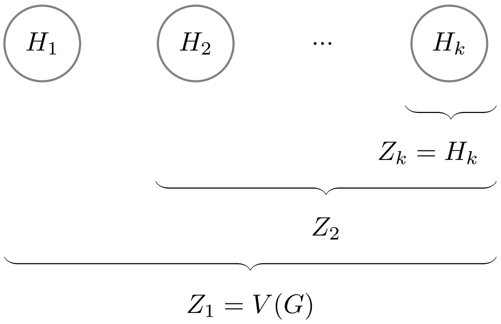


Figure 1 Hierarchical partition of the vertex set of a graph G.

Orientation. Consider the following orientation of the graph: if $l(u) < l(v)$, we orient the edge from u to v (Figure 2). If u and v are on the same level, we have an edge in both directions. This orientation enables us, given a vertex v, to refer to the neighbours of v in $Z_{l(v)}$ as the out-neighbours of v. We denote by $\deg^+(v)$ the out-degree of v.

$$\deg^+(v) = \deg_{Z_{l(v)}}(v)$$

1.2 Palettes

In the following, a palette is a data structure that keeps track of the colours that are used or not at a vertex. They cover colours $[1, 2^{\lceil \log(2\Delta-1)\rceil}]$, where the value of Δ can change. In the dynamic algorithms, we will use the following result:

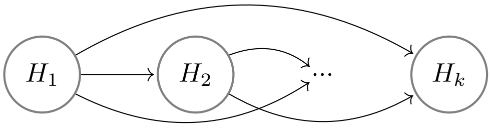


Figure 2 Hierarchical partition. In the static setting, the out degree of a vertex is bounded by d.

▶ **Theorem 1** (Palettes). *Consider two palettes P, Q such that there are a colours used in P and b colours used in Q. We can find a colour that is available in $[1, a+b+1]$ in $\log \Delta$ time.*

Proof. The proof is in the full version of the paper [12], along with the description of the data structure, which is generalised from the palettes in [6]. ◀

1.3 Roadmap

Baremboim, Elkin and Maimon describe a simple distributed algorithm for $\Delta + O(\alpha)$ colouring [2]. They form a hierarchical partition (or H-partition) of the graph, colour each set greedily, then colour the edges going out of the sets in an appropriate order. In Section 2, we show that this technique can easily be adapted into an efficient algorithm in the static setting. We also show a simpler algorithm that yields a $\Delta + 2\alpha - 2$ edge colouring within the same time by building a degeneracy order of the graph instead of an H-partition.

In Sections 3 and 4, we maintain a valid edge colouring in poly logarithmic time. We first present a simplified version of our algorithm that maintain a fully dynamic $\Delta_{max} + O(\alpha_{max})$ edge colouring in Section 3. This relies on two ideas: first, we maintain a dynamic H-partition, which only requires simple changes from the decomposition of Bhattacharya, Henzinger, Nanongkai, and Tsourakakis [9], namely, we need to maintain two palettes of available colours at each vertex, one for its neighbours and one for its out-neighbours. Then it is easy to colour an edge in a valid partition.

Then we present an algorithm that is adaptative to the maximum degree and arboricity and maintains a $\Delta(uv) + O(\alpha)$ edge colouring. Our data structure is derived from the Level Data Structure of Henzinger, Neumann and Wiese [20], which has the following property: instead of having an out-degree that depends on α, the levels of the partition have an increasingly large out-degree, and the sequence of maximum out-degree is such that it will be bounded by the current value of the arboricity, within a constant factor. Adapting to the current maximum can be done by updating the few problematic neighbours of a vertex when its degree decreases.

In the full version of the paper [12], we study the constants, and show that with small modifications of the data structure, we can get $\Delta_{max} + (4+\epsilon)\alpha_{max}$ and $\Delta(uv) + (8+\epsilon)\alpha$ colours in the dynamic setting.

2 Static $\Delta(uv) + O(\alpha)$ colouring

In this section, we describe a static edge-colouring algorithm. We arrange the vertices in a hierarchical partition that results in a $O(\alpha)$ out-orientation, then we colour vertices from right to left, so that for one of the endpoints, only the out-edges may already have colours. As the other endpoint has at most Δ coloured edges, the algorithm results in a $\Delta + O(\alpha)$ edge colouring.

The partition as it will be a crucial part of the dynamic algorithm. However, in the static setting, we could perform this algorithm with any α out-orientation. We show how this leads to a $\Delta + 2\alpha - 2$ edge colouring within the same asymptotic running time. We describe this algorithm first.

Arboricity and degeneracy. A graph G of arboricity α can be partitioned into α forests has at most $\alpha(n-1)$ edges, therefore it has a vertex of degree at most $2m/n \leq 2\alpha - 1$: the degeneracy of G is less than twice the arboricity. Therefore, the degeneracy is a 2-approximation of the arboricity. We can compute a degeneracy order of the graph, in linear time [23], and colour the edges as follows: we colour the out-edges of the vertices from right to left. We describe this in more details in the following proof.

▶ **Theorem 2.** *Given a graph of arboricity α, we can compute a $\Delta(uv) + 2\alpha - 2$ colouring of the graph in $O(m \log \Delta)$ time.*

Proof. For this proof only, let $d \leq 2\alpha - 1$ denote the degeneracy of the graph. We compute a degeneracy ordering, which can be done in linear time [23]. We greedily colour the edges from v_i to $v_{j>i}$ for $i = n-1, ..., 1$. Let us prove by induction that we can always find an available colour in a palette of size $\Delta(uv) + d - 1$. At the first iteration, we can greedily colour a potential edge from v_{n-1} to v_n with one colour.

Assume that for $i < n-1$, we have coloured any edge $v_j v_{j'}$ s.t. $j, j' > i$ with at most $\Delta(v_j v_{j'}) + d - 1$ colours. Consider an uncoloured edge $e = v_i v_j$ with $j > i$. v_j is incident to at most $\deg(v_j) - 1$ coloured edges and v_i is adjacent to at most $d-1$ coloured edges, therefore e sees at most $\deg(v_j) + d - 2$ colours and can find an available colour in a palette of size $\deg(v_j) + d - 1$.

If we maintain a binary tree over the palette at each vertex, we can find an available colour for an edge in $O(\log \Delta)$ time according to Theorem 1. Therefore, colouring all the edges takes $O(m \log \Delta)$ time. ◀

This could conclude the static version. However, this algorithm does not translate into an efficient dynamic algorithm, as far as we can say; therefore we introduce the notion of hierarchical partition, which will be crucial in our dynamic algorithms. The rest of the section proves a slightly weaker theorem through the use of such a partition:

▶ **Theorem 3.** *We can compute a static $\Delta(uv) + O(\alpha)$ edge colouring in $O(mlog\Delta)$ time.*

H-partition. In the following, we want to compute a partition such that in the corresponding orientation, each vertex has out-degree at most $O(\alpha)$. Barenboim, Elkin and Maimon H-partition from [1] describe a distributed algorithm which translates well to the static setting: we call a vertex *active* if it does not have a level assigned yet. Initially, all the vertices are active. A vertex that is active at iteration i will be part of the set Z_i. We call active degree the number of active neighbours of a vertex. Initially, the active degree of a vertex is its degree. Then at iterations $i = 1...k$, we group all the vertices of active degree at most $d = 4\alpha$ into a set H_i. For each vertex in H_i, we decrement the active degree of its neighbours and continue.

▶ **Lemma 4.** *We can compute a H-partition $H = \{H_1, ...H_k\}$ of a graph G of arboricity α such that the size of the partition is at most $k = \lfloor \log n \rfloor + 1$ and for all i, for all $v \in H_i$, the degree of v in $G(Z_i)$ is at most 4α.*

Proof. Consider a set H_i. The average degree is:

$$2\frac{|E(Z_i)|}{|V(Z_i)|} \leq 2\alpha \leq \frac{4\alpha}{2}$$

At most half of the vertices have a degree more than 4α, otherwise we would have an average degree greater than 2α, leading to a contradiction. Therefore:

$$\begin{aligned}|Z_{i+1}| &\leq |Z_i|/2\\ |Z_i| &\leq n/2^{i-1}\\ |Z_{\lceil \log n \rceil + 1}| &< 1\end{aligned}$$

Therefore, we can safely set $k = \lceil \log n \rceil$ ◀

▶ **Lemma 5.** *We can compute the H-partition described in lemma 4 in $O(m)$ time.*

Proof. The sum of the degrees is $2m$, so we cannot decrement the active degrees more than $O(m)$ time in total, and decrementing a degree takes constant time. Then, at iteration i, we can search the vertices of low active degree in $O(n/2^i)$ time. Therefore, the running time to compute the H-partition is $O(m + n)$. ◀

▶ **Lemma 6.** *Given a H-partition, we can compute a $\Delta(uv) + O(\alpha)$ colouring of the graph in $O(m \log \Delta)$ time.*

Proof. Let $d = 4\alpha$.

We greedily colour the edges from H_i to $H_{j \geq i}$ for $i = k, ..., 1$. Let us prove by induction that we can always find an available colour in a palette of size $\Delta(uv) + d - 1$. At the first iteration, we can greedily colour H_k with $2d - 1$ colours.

Assume that for $i < k$, we have coloured $G(H_{i+1} \cup ...H_k)$ with at most $\Delta + d - 1$ colours. Consider an uncoloured edge $e = uv$ such that $u \in H_i, v \in H_i \cup ...H_k$. v is incident to at most $\deg(v) - 1$ coloured edges and u is adjacent to at most $d - 1$ coloured edges, therefore e sees $\deg(v) + d - 2$ colours and can find an available colour in a palette of size $\deg(v) + d - 1$.

If we maintain a binary tree over the palette at each vertex, we can find an available colour for an edge in $O(\log \Delta)$ time according to theorem 1. Therefore, colouring all the edges given the H-partition takes $O(m \log \Delta)$ time using theorem 1. ◀

3 Dynamic $\Delta_{max} + O(\alpha_{max})$ colouring

In this section, we show how to update a dynamic edge colouring. We first discuss how we can recolour an edge within a valid H-partition, then we show how we can maintain such a partition. The algorithm requires α_{max} and Δ_{max} to be known in advance.

3.1 Data structure

We consider a H-partition that maintains the following invariants, with $d = 4\alpha_{max}$:

1. Each vertex v such that $l(v) > 1$ has at most βd neighbours in $Z_{l(v)}$ (out-neighbours). In this section, we choose $\beta = 5$.
2. Each vertex v has at least d neighbours in $Z_{l(v)-1}$

For each vertex $v \in H_i$, we store the following:

- For each $j < i$, we store the neighbours of v at level j, $\Gamma_{H_j}(v)$, in a linked list $N_{H_j}(v)$.
- We store the out-neighbours of v, $\Gamma_{Z_i}(v)$, in a linked list $N_{Z_i}(v)$.
- We store the length of each linked list.
- For each edge uv, we store a pointer to the position of u in the appropriate neighbour list $N_{\cdot}(v)$ and conversely.
- We store two palettes: one for the neighbours of v and one for its out-neighbours. We refer to those palettes as $P_G(v)$ and $P_{Z_i}(i)$.

3.2 Recolouring an edge

Let us start with a key sub problem: given an uncoloured edge in an otherwise valid data structure, what is the cost of colouring the edge? We colour the edge uv as we would have in section 2.

Let u be the leftmost vertex, i.e. $i = l(u) \leq l(v)$. We ignore the neighbours of u that have a lower level, which can be done in practice by searching for an available colour in $P_{Z_i}(u) \cap P_G(v)$. The colour picked may conflict with a single edge from a neighbour of u on a lower level. If that is the case, we recolour that edge in the same way.

▶ **Lemma 7.** *An uncoloured edge uv in an otherwise valid data structure can be coloured in* $O(\min(l(u), l(v)) \cdot \log \Delta)$ *time.*

Proof. Assume wlog. $l(u) \leq l(v)$. If uv is not coloured, at most $\beta d - 1$ colours are represented at u in the palette $P(Z_{l(u)})$, and at most $\deg(v) - 1$ colours are represented at v. Therefore, there exists an available colour in the palette $[\deg(v) + \beta d - 1]$. If there exists w such that uw conflict with uv, then $l(w) < l(u)$. Therefore, the level of the left endpoint of the conflicting edge decrease at each iteration, which can happen at most $l(u) - 1$ times (Figure 1). Therefore, recolouring an edge and recursively resolving conflicts take $O(l(u) \cdot \log \Delta)$ time. ◀

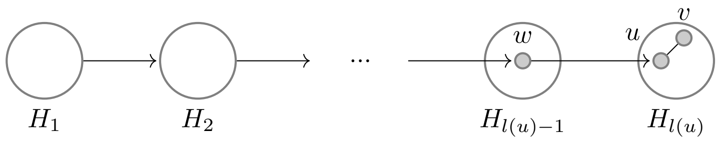


Figure 3 We may need to recolour at most $l(u)$ edges.

Algorithm 1 Recolour.

```
procedure RECOLOUR($uv$)
    if $l(u) > l(v)$ then
        $u, v = v, u$
    $i = l(u)$
    // Pick a colour available in $G(Z_i)$ in the palette $[\deg(v) + \beta d - 1]$.
    COLOUR($uv, \deg(v) + \beta d - 1, P_{Z_i}(u), P_G(v)$)
    if $c(uv)$ is represented at $u$ then
        Use the pointer $C(u)[c]$ to find the conflicting edge $wu$
        RECOLOUR($wu$)
```

3.3 Updating the hierarchical partition and full algorithm

Updates. The algorithm for the updates is the following: let us say that a vertex that violates Invariant 1 or 2 is *dirty.* As long as we have a dirty vertex v: if v violate the first invariant, we increment its level. If it violates Invariant 2, we decrement it. when doing so, we need to update our linked lists of neighbours and our tree palettes. Updating the trees is the limiting factor. The details of the updates are described in Algorithm 5. The algorithm terminates, which will be justified later, as we will define a positive potential that strictly decreases at each step.

▶ **Lemma 8.** *Increasing the level of a vertex takes* $O\left(\deg^+(v)\log\Delta\right)$ *time. Decreasing the level of a vertex takes* $O\left(d\log\Delta\right)$ *time.*

Proof. When we increment the level of a vertex, we first update the lists of neighbours of v: we traverse $N_{Z_i}(v)$ to split it in $N_{H_i}(v)$ and $N_{Z_{i+1}}(v)$ in $O(\deg^+(v))$ time. Then we discard $P_{Z_i}(v)$ and create $P_{Z_{i+1}}(v)$: we traverse the linked list $N_{Z_{i+1}}(v)$ and insert the $\deg_{Z_{i+1}}(v) = O(\deg^+(v))$ elements in the tree, which takes $O(\deg^+(v)\log\Delta)$ time. We also need to update the data structures of at most $\deg^+(v)$ neighbours, which takes constant time per neighbour for the linked lists and $\log\Delta$ time per neighbour for trees.

Finally, we need to check which vertices became dirty as a result of the operation. The set Z_{i+1} has one more element, and for $j \neq i+1$, Z_j is unchanged. Therefore, we check if v itself, or any vertex in $N_{Z_{i+1}}(u)$, breaks the first invariant, which takes $O(\deg_{Z_{i+1}}(v))$ time.

The procedure for decrementing a level is similar. If we decrement the level of a vertex, we know that the second invariant was not respected, i.e. $\deg_{Z_{i-1}}(v) < d$. After the operation, Z_i does not include v any more and for $j \neq i$, Z_j is unchanged. To update the lists of neighbours of v, we merge $N_{H_{i-1}(v)}$ and $N_{Z_i}(v)$, which takes $O(\deg_{Z_{i-1}}(v)) = O(d)$ time. We create the palette of v for the set Z_{i-1} in $O(\deg_{Z_{i-1}}(v))\log\Delta = O(d\log\Delta)$ time. Then we need to update the neighbour lists and the palettes of the neighbours of u in Z_i, which also takes constant time per neighbour for the lists and $O(\log\Delta)$ time per neighbour for the palettes. Finally, we check if v or any of its neighbours in Z_i is dirty, which takes constant time per vertex. ◀

Algorithm 2 Insert the edge uv.

```
procedure ADD($uv$)
    if $l(u) > l(v)$ then
        ADD($vu$)
        End procedure.
    Add $v$ to the out-neighbours of $u$ and store the corresponding pointer.
    if $l(u) <$ l(v) then
        Add $u$ to $N_{H_{l(u)}}(v)$, the neighbours of $v$ at level $l(u)$
        Store the corresponding pointer.
    else
        Add $u$ to the out-neighbours of $v$ and store the corresponding pointer.
    // The palettes will be updated when $uv$ gets a colour.
    Check if $u$ or $v$ became dirty.
    RECOVER
    RECOLOUR($uv$)
```

▶ **Theorem 9.** *We can maintain a dynamic* $\Delta_{max} + O(\alpha_{max})$ *edge colouring of a graph in* $O(\log n \log \Delta_{max})$ *amortized update time.*

Proof. We define the following potential.

$$B = \log\Delta_{max}\sum_{v\in V}\phi(v) + \log\Delta_{max}\sum_{e\in E}\psi(e)$$

$$\phi(v) = \sum_{j=1}^{l(v)-1}\max(0, \beta d - \deg_{Z_j}(v))$$

$$\psi(u,v) = 2(k - \min(l(u), l(v))) + \mathbf{1}_{l(u)=l(v)}$$

Algorithm 3 Delete the edge uv.

```
procedure Delete($uv$)
    if $l(u) > l(v)$ then
        Delete($vu$)
        End procedure.
    Remove the colour of $uv$ from the palettes of neighbours of $u$ and $v$.
    Remove the colour of $uv$ from the palette of out-neighbours of $u$.
    if $l(u) == l(v)$ then
        Remove the colour of $uv$ from the palette of out-neighbours of $v$.
    Remove $v$ from the neighbours of $u$ using the corresponding pointer.
    Remove $u$ from the neighbours of $v$ using the corresponding pointer.
    Check if $u$ or $v$ became dirty.
    Recover
```

Algorithm 4 Recursively fix the invariants.

```
procedure Recover
    while there exists a dirty vertex $v$ do
        $i = l(v)$
        if $\deg^+(v) > \beta d$ then Increment(v)
        else if $\deg_{Z_{i-1}}(v) < d$ then Decrement(v)
```

When we insert an edge, we create a potential $\psi(u, v) \leq 2k$. The potential of the other edges do not change and the potential of a vertex can only decrease, therefore the potential B increase by at most $2k \log \Delta_{max}$. When we delete an edge, $\phi(u)$ and $\phi(v)$ increase by at most k each, $\psi(u, v)$ is deleted, and the other potentials are not affected, therefore the potential increase by at most $2k \log \Delta_{max}$.

When a dirty vertex increment its level, the cost of the operation will be paid by the drop in potential from the edges. When a dirty vertex decrements its level, the cost of the operation is paid by the drop in potential from the vertex, despite the increase in potential from the edges. For completeness, we repeat the details of the analysis that follows closely that of [9]. Let i be the level of v before the operation.

Incrementing the level of a dirty vertex

- When $l(v)$ increases, we get $l(v) = i + 1$, so we add $\max(0, \beta d - \deg_{Z_i}(v))$ to $\phi(v)$. As invariant 1 was violated, we must have had $\deg_{Z_i}(v) > \beta d$ and therefore $\max(0, \beta d - \deg_{Z_i}(v)) = 0$: the potential of v is unchanged.
  The potential of the other vertices can not increase (though it might decrease for a neighbour of v).
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v)$ edges whose potential drop by one or two.

The total drop in potential is at least:

$$\deg^+(v) \log \Delta_{max} = \Omega(\deg^+(v) \log \Delta)$$

Algorithm 5 Incrementing / decrementing the level of a vertex.

```
procedure INCREMENT($v$)
    $i = l(v)$.
    Split $N_{Z_i}(v)$ in $N_{H_i}(v)$ and $N_{Z_{i+1}}(v)$.
    Create the palette of $v$ for $Z_{i+1}$.
    Discard the palette of $v$ for $Z_i$.
    for $u \in N_{Z_{i+1}}(u)$ do
        Using the pointer in $uv$, remove $v$ from $N_{H_i}(u)$, add $v$ to $N_{Z_{i+1}}(u)$ or $N_{H_{i+1}}(u)$.
        Update the pointers accordingly.
        if $l(u) = i + 1$ then
            Update the palette of $u$ for the set $Z_{i+1}$: add colour $c(uv)$.
    Increment $l(v)$
procedure DECREMENT($v$)
    $i = l(v)$.
    Merge $N_{Z_i}(v)$ and $N_{H_{i-1}}(v)$ into $N_{Z_{i-1}}(v)$.
    Create the palette of $v$ for $Z_{i-1}$.
    Discard the palette of $v$ for $Z_i$.
    for $u \in N_{Z_i}(v)$ do
        Move $v$ from $N_{H_i}(u)$ or $N_{Z_i}(u)$ to $N_{H_{i-1}}(u)$.
        Update the pointers accordingly.
        if $l(u) = i$ then
            Update the palette of $u$ for the sets $Z_i$: remove colour $c(uv)$.
    Decrement $l(v)$
```

Decrementing the level of a dirty vertex

- We must have $\deg_{Z_{i-1}}(v) < d \Rightarrow \max(0, \beta d - \deg_{Z_{i-1}}(v) \geq (\beta - 1)d = 4d$. Therefore, the $\phi(v)$ drops by at least $4d$.
- If u is a neighbour of v, $\deg_{Z_j}(v)$ is decremented if $j = i$ and is unchanged otherwise. This affects the potential of u if $l(u) > i$. Therefore, $\sum_{u \in \Gamma(v)} \phi(u)$ increase by at most $\deg_{Z_{i+1}}(v) < \deg_{Z_{i-1}}(v) < d$.
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v) \leq \deg_{Z_{i-1}}(v) \leq d$ edges whose potential increase by one or two.

The total drop of potential is at least:

$$\log \Delta_{max} (4d - d - 2d) = d \log \Delta_{max} = \Omega(d \log \Delta) \qquad ◀$$

4 Dynamic $\Delta(uv) + O(\alpha)$ colouring

The data structure from the previous section could maintain a dynamic $\Delta_{max} + O(\alpha_{max})$ colouring. We modify it further to create a data structure that adapts to the maximum degree and arboricity. We adapt the arboricity by making a structure that does not depend explicitly on α: instead, we give the level increasingly large out-degrees and show how the out-degree ends up being bounded by the current value of the arboricity within a constant factor. To adapt to the maximum degree, we locally adapt to the degree of the vertices: when the degree of a vertex decrease, we can recolour its problematic edges in polylogarithmic time.

4.1 Data Structure

In the following, $L = 1 + \lceil \log n \rceil$. We now consider a H-partitions with $k = L \cdot \lceil \log n \rceil$ levels. Let $k' = \max_{v \in V} l(v)$ denote the maximum level of any vertex, that is, the highest non-empty level.

We partition the levels into $\lceil \log n \rceil$ groups of size L (Figure 4).

Let $g(v)$ denote the group of a vertex and $d(v) = 2^{g(v)}$. We will maintain the following:

1. Each vertex v has at most $2\beta d(v)$ neighbours in $Z_{l(v)}$. In this section, we choose $\beta = 5$, so any vertex has at most $10d(v)$ out-neighbours.
2. Each vertex v has at least $d(v)$ neighbours in $Z_{l(v)-1}$
3. The colour of an edge uv is chosen from the first $\Delta(uv) + 2\beta d(v)$ colours of the palette. This condition enables the data structure to adapt to changes in the arboricity. In the following, we prove that for any vertex, $g(v) \leq \lceil \log(4\alpha) \rceil$, which will result in a $\Delta(uv) + O(\alpha)$ colouring.

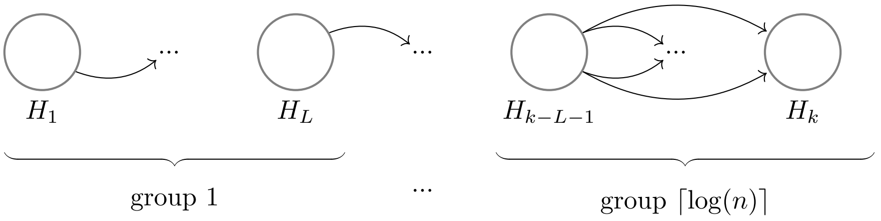


Figure 4 Hierarchical partition with two levels. The in degree of a vertex v is still only bounded by Δ, when the bound on the out degree depends on its group $g(v)$.

We store the neighbours and palettes of the vertices as described in the previous section.

▶ **Lemma 10** (Maximum level). *The index of the highest non empty level, k', is at most $O(\log \alpha \log n)$.*

Proof. Consider the group $\lceil \log(4\alpha) \rceil$ and a level i in this group. We can repeat the arguments from the proof of lemma 4 and show that the last set of the group $Z_{L\lceil \log(4\alpha) \rceil}$, with $L = 1 + \lceil \log n \rceil = O(\log n)$, has at most one element, therefore, all the higher groups are empty. ◀

▶ **Lemma 11.** *An uncoloured edge uv in an otherwise valid data structure can be coloured in time $O(\log \alpha \log n \log \Delta)$.*

Proof. Following the same reasoning as in the previous section, we have to recolour $O(k')$ edges, which we can do in $O(k' \log \Delta)$ time. ◀

4.2 Updating the hierarchical partition and full algorithm

When edges are deleted, the arboricity α or the maximum degree Δ may decrease. To adapt to the degree, we recolour the edges from in-neighbours of v when uv is deleted. To maintain the arboricity, we will recolour the edges to the out-neighbours of v when its level decrease. As a result, we maintain that an edge uv, $l(u) \leq l(v)$, has a colour from the palette $[\deg(v) + \beta d(u) - 1]$

▶ **Lemma 12** (Adapting to the maximum degree). *We can recolour the edges from the in-neighbours of v in $O(\log n \log^2 \alpha \log \Delta)$ time.*

Proof. In each group $i \le g(v)$, v might have at most one neighbour u such that uv has colour $\deg(v) + 2\beta d(u) - 1$. For each such group, it takes constant time to find this edge using the pointer $C(v)[\deg(v) + 10 \cdot 2^i - 1]$. There are therefore at most $g(v)$ edges that we may have to recolour, each in $O(l(v) \log \Delta)$ time, when the degree of v decreases. When we delete an edge, we do this for each of its two endpoints, which takes $O(g(v)l(v) \log \Delta)$ time. ◀

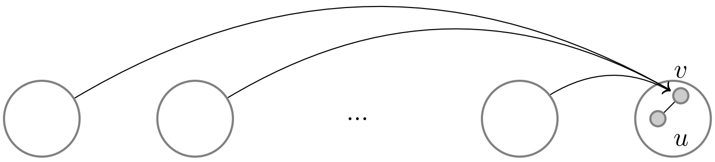


Figure 5 When the degree of a vertex v decrease, we may need to recolour at most $l(v)$ edges.

▶ **Lemma 13** (Adapting to the arboricity). *Increasing the level of a vertex v takes $O(\deg^+(v) \log \Delta)$ time. Decreasing the level of a vertex takes $O(d \cdot k' \log \Delta)$ time.*

Proof. The difference with the previous section is the following: when we decrement the level of a vertex v, we may need to recolour any edge uv such that $\min(l(u), l(v))$ decrease, i.e. u was on the same level as v or higher. There are at most $\deg^+(v) = O(d)$ such edges, which can be recoloured in $O(k' \log \Delta)$ time. ◀

Algorithm 6 Decrementing the level of a vertex with adaptative arboricity.

```
procedure ADAPTATIVE DECREMENT($v$)
    $i = l(v)$.
    Merge $N_{Z_i}(v)$ and $N_{H_{i-1}}(v)$ into $N_{Z_{i-1}}(v)$.
    Create the palette of $v$ for $Z_{i-1}$.
    Discard the palette of $v$ for $Z_i$.
    for $u \in N_{Z_i}(v)$ do
        Move $v$ from $N_{H_i}(u)$ or $N_{Z_i}(u)$ to $N_{H_{i-1}}(u)$.
        Update the pointers accordingly.
        if $l(u) = i$ then
            Update the palette of $u$ for the sets $Z_i$: remove colour $c(uv)$.
        RECOLOUR($uv$)
    Decrement $l(v)$
```

▶ **Theorem 14.** *We can maintain a dynamic $\Delta(uv) + O(\alpha)$ edge colouring of a graph in amortized $O(\log n \log \alpha_{max} \log \Delta_{max})$ time for insertions, $O(\log^2 n \log \alpha_{max} \log \alpha \log \Delta_{max})$ for deletions.*

Proof. We define the following potential:

$$\begin{aligned} B &= k'_{max} \log \Delta_{max} \sum_{v \in V} \phi(v) + \log \Delta_{max} \sum_{e \in E} \psi(e) \\ \phi(v) &= \sum_{j=1}^{l(v)-1} max\left(0, \beta d(v) - \deg_{Z_j}(v)\right) \\ \psi(u,v) &= 2(k'_{max} - \min(l(u), l(v))) + \mathbf{1}_{l(u)=l(v)} \\ k'_{max} &= L\lceil \log(4\alpha_{max}) \rceil \in O(\log n \log \alpha_{max}) \end{aligned}$$

When we insert an edge, we create a potential $\psi(u,v) \leq 2k'_{max}$. The potential of the other edges do not change and the potential of a vertex can only decrease, therefore the potential B increase by at most $\log \Delta_{max} \cdot 2k'_{max}$. When we delete an edge, we need $O(\log n \log^2 \alpha \log \Delta)$ time to update the overflowing colours of the in-neighbours of v (lemma 12). Then for the potentials: $\phi(u)$ and $\phi(v)$ increase by at most k' each, $\psi(u,v)$ is deleted, and the other potentials are not affected, therefore the potential increase by at most $k'_{max} \log \Delta_{max} \cdot 2k'$, therefore the costs.

Incrementing the level of a dirty vertex. Let d denote the value of $d(v)$ before the change of level and d' the value after the modification.

- If v changes group, we have $d' = 2d$, otherwise $d' = d$. Either way, we have $\deg^+(v) > 10d \Rightarrow \max(0, \beta d' - \deg^+(v)) = 0$. It follows that $\phi(v)$ is unchanged. The potential of the other vertices can not increase (though it might decrease for a neighbour of v).
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v)$ edges whose potential drop by one or two.

The total drop in potential is at least:

$$\deg^+(v) \log \Delta_{max} = \Theta(\deg^+(v) \log \Delta)$$

Decrementing the level of a dirty vertex.

- We must have $\deg_{Z_{i-1}}(v) < d \Rightarrow max\left(\beta d - \deg_{Z_{i-1}}(v)\right) \geq 4d$. Therefore, the $\phi(v)$ drops by at least $4d$. If the level of v decreases, the potential only decreases further.
- If u is a neighbour of v, $\deg_{Z_j}(v)$ is decremented if $j = i$ and is unchanged otherwise. This affects the potential of u if $l(u) > i$. Therefore, $\sum_{u \in \Gamma(v)} \phi(u)$ increase by at most $6 \deg_{Z_{i+1}}(v) < \deg_{Z_{i-1}}(v) < d$.
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \leq i$. Therefore, there are exactly $\deg^+(v) < \deg_{Z_{i-1}}(v) < d$ edges whose potential increase by one or two.

The total drop of potential is at least:

$$k'_{max} \log \Delta_{max} (4d - d) - \log \Delta_{max} \cdot 2d \geq k'_{max} \log \Delta_{max} \cdot d \qquad \blacktriangleleft$$

5 Conclusion

In this paper, we show how to maintain a $\Delta(uv) + O(\alpha)$ edge colouring in polylogarithmic time through the use of dynamic hierarchical partition. We also propose a simpler data structure to maintain a $\Delta_{max} + O(\alpha_{max})$ edge colouring, which can be done faster than the aforementioned algorithm.

We give an amortized analysis of the running time of our dynamic algorithms. This raises the question of what can be done in worst case time. In our case, we are only limited by the updates of our hierarchical partitions, so it motivates the search for hierarchical partitions with efficient worst-case update times.

The question that motivated our research is still open for graphs that have a large arboricity compared to their maximum degrees: is it possible to maintain a $\Delta + O(\Delta^{1-\epsilon})$ edge colouring, with ϵ a positive constant, in polylogarithmic time?

In the static setting, we showed that we can make a $\Delta(uv) + 2\alpha - 2$ edge colouring in $O(m \log \Delta)$ time, which is as fast as the greedy $2\Delta(uv) - 1$ algorithm. Thus, we get a $\Delta(uv) + O(1)$ edge colouring for graphs of constant arboricity, such as planar graphs, in near-linear time: more precisely, a planar graph has arboricity at most 3 [27], so by our result, it can in near-linear time be edge-coloured with $\Delta(uv) + 2\alpha - 2 = \Delta(uv) + 4$ colours.

Recently, it was shown by Bhattacharya, Costa, Panski and Solomon [8] that one can compute a $(\Delta + 1)$ edge colouring in $\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$-time, which gives a near linear time algorithm for graphs of polylogarithmic arboricity. It emphasises the question whether a near-linear time $\Delta + O(1)$ edge-colouring algorithm could be obtained for a wider class of graphs.

References

1 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 25–34, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1400751.1400757.

2 Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed $(\Delta + o(\Delta))$-edge-coloring, and vertex-coloring of graphs with bounded diversity. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 175–184. ACM, 2017. doi:10.1145/3087801.3087812.

3 Anton Bernshteyn. A Fast Distributed Algorithm for $((\Delta + 1))$-Edge-Coloring. *Journal of Combinatorial Theory, Series B*, 152:319–352, 2022. doi:10.1016/j.jctb.2021.10.004.

4 Anton Bernshteyn and Abhishek Dhawan. Fast algorithms for vizing's theorem on bounded degree graphs. *CoRR*, abs/2303.05408, 2023. doi:10.48550/arXiv.2303.05408.

5 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^{2} n)$ replacements. *J. ACM*, 66(5):37:1–37:23, 2019. doi:10.1145/3344999.

6 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

7 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Arboricity-dependent algorithms for edge coloring. *CoRR*, abs/2311.08367, 2023. doi:10.48550/arXiv.2311.08367.

8 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Density-sensitive algorithms for $(\Delta+1)$-edge coloring. *CoRR*, abs/2307.02415, 2023. doi:10.48550/arXiv.2307.02415.

9 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM, 2015. doi:10.1145/2746539.2746592.

10 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In Frank Dehne, Jörg-Rüdiger Sack, Arvind Gupta, and Roberto Tamassia, editors, *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 773–782, Netherlands, 1999. Springer. 6th International Workshop on Algorithms and Data Structures. WADS 1999 ; Conference date: 11-08-1999 Through 14-08-1999. doi:10.1007/3-540-48447-7_34.

11 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2633–2652. SIAM, 2018. doi:10.1137/1.9781611975031.168.

12 Aleksander B. G. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. *CoRR*, abs/2311.10616, 2023. doi:10.48550/arXiv.2311.10616.

13 Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1013–1026. ACM, 2023. doi:10.1145/3564246.3585105.

14 Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D$ time. *Combinatorica*, 21(1):5–12, January 2001. doi:10.1007/s004930170002.

15 Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4273–4295. SIAM, 2023. doi:10.1137/1.9781611977554.ch163.

16 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1937–1945, USA, 2019. Society for Industrial and Applied Mathematics.

17 Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1):33–50, 2001. doi:10.1016/S0304-3975(99)00152-8.

18 Harold N. Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-colouring graphs. *Technical Report*, 1985.

19 Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic Distributed Edge-Coloring with Fewer Colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 418–430, 2018.

20 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity, 2020. doi:10.48550/arXiv.2002.10142.

21 Ian Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.

22 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.

23 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983. doi:10.1145/2402.322385.

24 C. St.J. A. Nash-Williams. Decomposition of Finite Graphs Into Forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, January 1964. doi:10.1112/jlms/s1-39.1.12.

25 Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic MIS in uniformly sparse graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 92:1–92:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.92.

26 David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 712–729. SIAM, 2016. `doi:10.1137/1.9781611974331.ch51`.

27 K. S. Poh. On the linear vertex-arboricity of a planar graph. *J. Graph Theory*, 14(1):73–75, 1990. `doi:10.1002/jgt.3190140108`.

28 Corwin Sinnamon. A randomized algorithm for edge-colouring graphs in $O(m\sqrt{n})$ time. *CoRR*, abs/1907.03201, 2019. `arXiv:1907.03201`.

29 Hsin-Hao Su and Hoa T Vu. Towards the Locality of Vizing's Theorem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 355–364, 2019.

30 V. G. Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, May 1965. `doi:10.1007/BF01885700`.

Approximating Minimum Sum Coloring with Bundles

Seyed Parsa Darbouy

Department of Computing Science, University of Alberta, Canada

Zachary Friggstad[1]

Department of Computing Science, University of Alberta, Canada

Abstract

In the Minimum Sum Coloring with Bundles problem, we are given an undirected graph $G = (V, E)$ and (not necessarily disjoint) bundles $V_1, V_2, \ldots, V_p \subseteq V$ with associated weights $w_1, \ldots, w_p \geq 0$. The goal is to give a proper coloring of G using positive integers to minimize the weighted average/total completion time of all bundles, where the completion time of a bundle is the maximum integer assigned to one of its nodes. This is a common generalization of the classic Minimum Sum Coloring problem, i.e. when all bundles are singleton nodes, and the classic Chromatic Number problem, i.e. the only bundle is all of V.

Despite its generality as an extension of Minimum Sum Coloring, only very special cases have been studied with the most common being the line graph $L(H)$ of a graph H (also known as Coflow Scheduling). We provide the first constant-factor approximation in perfect graphs and, more generally, graphs whose chromatic number is within a constant factor of the maximum clique size in any induced subgraph. For example, we obtain constant-factor approximations for graphs that are well-studied in minimum sum coloring such as interval graphs and unit disk graphs.

Next, we extend our results to get constant-factor approximations for a general model where the bundles are disjoint (i.e. can be thought of as jobs brought by the corresponding client) and we are only permitted to color/schedule a bounded number of jobs from each bundle at any given time. Specifically, we get constant-factor approximations for this model if the nodes of graph G have an ordering $v_1, v_2, \ldots, v_n$ such that the *left-neighborhood* $N_\ell(v_i) := \{v_j : j < i, v_i v_j \in E\}$ can be covered by $O(1)$ cliques. For example, this applies to chordal graphs, unit disc graphs, and circular arc graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Scheduling algorithms

Keywords and phrases Approximation Algorithms, Scheduling, Coloring

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.21

Funding *Zachary Friggstad*: Supported by an NSERC Discovery Grant and an Accelerator Supplement.

1 Introduction

The Minimum Sum Coloring (**MSC**) problem is a well-studied problem lying at the intersection of scheduling theory and graph coloring. In it, we are given an undirected graph $G = (V, E)$ on n nodes. The goal is to find a **proper coloring** $\chi : V \to \{1, 2, \ldots\}$ of nodes of V, i.e. $\chi(u) \neq \chi(v)$ for all $uv \in E$, with minimum total color $\sum_{v \in V} \chi(v)$. Viewed as a scheduling problem, this models settings where unit-length jobs may be completed in parallel but resource conflicts prevent certain pairs of jobs from being completed at the same time.

While different than standard graph coloring where the goal is to simply minimize the number of distinct colors of the coloring, it is essentially just as hard to approximate. That is, unless $\mathbf{P} = \mathbf{NP}$, there is no $n^{1-\delta}$-approximation for any constant $\delta > 0$ [2]. However, in

[1] Corresponding Author

certain cases, it is possible to get improved approximations. For example, if it is possible to efficiently compute a maximum independent set in G and any of its induced subgraphs then greedily coloring by computing a maximum independent set I, coloring I with the next unused integer, and then removing I from G yields a 4-approximation [2]. There is a large body of work on getting improved constant-factor approximation in more structured special cases or obtaining constant-factor approximations in other graph classes, see the related works section below for further discussion.

In this work, we extend this model in two ways. The first extension is given as follows.

▶ **Definition 1.** *In the* Minimum Sum Coloring with Bundles *problem (***MSCB***), we are given an undirected graph $G = (V, E)$ and collection of bundles $V_1, \ldots, V_p \subseteq V$ with associated weights $w_1, \ldots, w_p \geq 0$. The goal is to find a proper coloring $\chi : V \to \{1, 2, \ldots, |V|\}$ of V in a way that minimizes the total weighted makespan of all bundles. That is, the coloring should minimize $\sum_{i=1}^{p} w_i \cdot \max_{v \in V_i} \chi(v)$.*

If one views bundles as being associated with clients, then each client is interested in the time it takes for their bundle to be complete. The objective of **MSCB** is then equivalent to minimizing the average weighted completion time of all clients' bundles. If all bundles are singletons, then **MSCB** is the same as **MSC**. If the partition has only a single bundle, i.e. $p = 1$, then **MSCB** is the same as computing the chromatic number of G. Thus, **MSCB** is a common generalization of both problems.

It should be noted that **MSCB** has been studied previously in certain special cases. For example, when G is the line graph of an undirected graph the problem is known as Coflow Scheduling. See Section 1.1 for a discussion of this and other related problems.

Many techniques that are successful for **MSC** fail in **MSCB**. For example, while iteratively computing a maximum independent set in G to color the nodes is a 4-approximation for **MSC** [2], it could be as bad as an $\Omega(\log n)$-approximation for computing the chromatic number even in trees (see Appendix A). Further difficulties in designing **MSCB** approximations will be discussed in Section 1.3.

Our first main result is a constant-factor approximation for **MSCB** in perfect graphs. Recall a graph G is **perfect** if the maximum clique size in $G[U]$ equals the chromatic number of $G[U]$ for any $U \subseteq V$ where $G[U] = (U, \{uv \in E : u, v \in U\})$ denotes the subgraph of G induced by U. Examples of perfect graphs include bipartite graphs, line graphs of bipartite graphs, interval graphs, comparability graphs, split graphs, permutation graphs, and chordal graphs as well as the edge-complements of these graphs. For a broader discussion of perfect graphs, we refer the reader to Golumbic's excellent book [8].

▶ **Theorem 2.** **MSCB** *on perfect graphs admits a polynomial-time* 10.874*-approximation.*

Our presentation will first focus on proving Theorem 2 in the case of interval graphs. These are graphs $G = (V, E)$ where each $v \in V$ is associated with an interval $[s_v, t_v] \subseteq \mathbb{R}$ and we have an edge $uv \in E$ whenever the corresponding intervals intersect, i.e. $[s_u, t_u] \cap [s_v, t_v] \neq \emptyset$. The linear programming (LP) relaxation we use in this case is simpler than in the general case of perfect graphs. After presenting the algorithm for interval graphs, we discuss the few necessary changes to extend it to perfect graphs in general.

Our techniques extend further to classes of graphs for which the chromatic number is approximately equal to the maximum clique number and these quantities can be approximated in polynomial time. Namely, we get the following extension. Recall a graph class $\mathcal{G}$ is **hereditary** if for any $G \in \mathcal{G}$ we have $G[U] \in \mathcal{G}$ as well for all induced subgraphs of G.

▶ **Corollary 3.** *Let $\mathcal{G}$ be any hereditary graph class with the following properties: (a) the maximum clique size is within a constant factor of the chromatic number of any $G \in \mathcal{G}$, (b) given vertex weights $w_v, v \in V$ for a graph $G \in \mathcal{G}$, there is a constant-factor approximation algorithm for the maximum-weight clique of G, and (c) there is a constant-factor approximation for coloring any $G \in \mathcal{G}$ with the minimum number of colors. Then there is a constant-factor approximation for* **MSCB** *for graphs in $\mathcal{G}$.*

For example, we get $O(1)$-approximations for the following graph classes:

- Unit disc graphs: when vertices are associated with discs of radius 1 in the plane and edges indicate when two discs intersect. The classic algorithm in [4] for computing a maximum-size clique easily generalizes to compute a maximum-weight clique in polynomial time. It is possible to color any unit disc graph with maximum clique size K using fewer than $3 \cdot K$ colors in polynomial time [13].
- Circular-arc graphs: when vertices are associated with arcs of a circle and edges indicate when two such arcs intersect. A maximum weight clique can be found in polynomial time, e.g. [3]. One can efficiently $2 \cdot K$-color a circular arc graph with maximum clique size K by first coloring the arcs spanning one particular point with at most K colors. After removing these arcs, we are left with an interval graph which can also be colored with K additional colors since interval graphs are perfect.

Our general definition of **MSCB** allows bundles to overlap. A natural special case is where the bundles constitute a partition $V_1, \ldots, V_p$ of V, i.e. each client brings their own jobs V_i to be processed. In this case, it is natural to imagine that clients themselves are somewhat limited on how quickly they can interact with the service provider that is processing the jobs. For example, perhaps clients can only deliver or take away a bounded number of jobs from the processing center at any given time. We consider the following generalization of **MSCB**

▶ **Definition 4.** *In the* Minimum Sum Coloring with Bundles and Task Concurrency *problem (***MSCB-TC***), we are given a graph $G = (V, E)$ and a* **partition** *$V_1, \ldots, V_p$ of V with associated weights $w_1, \ldots, w_p \geq 0$. Further, for each $1 \leq i \leq p$ we are given a bound $d_i \geq 1$ on the number of jobs from bundle i that may be processed at any moment. The goal is still to find a proper coloring χ that minimizes the total weighted completion time of all bundles, but we further require $|\{v \in V_i | \chi(v) = t\}| \leq d_i$ for each $1 \leq i \leq p$ and each time step/color t.*

To the best of our knowledge, this extension of **MSCB** has not been previously studied even in special cases. We obtain constant-factor approximations for this problem, though in more restricted settings. Recall that a graph G is chordal if every cycle of length ≥ 4 has a chord. That is, if $v_1, v_2, \ldots, v_\ell, v_1$ is a cycle of length $\ell \geq 4$ then we also have $v_i v_j \in E$ for some $1 \leq i, j \leq \ell$ where i, j are not consecutive indices around the cycle, i.e. $1 \leq i \leq j - 1$ and if $i = 1$ then $j \neq \ell$. Interval graphs, an important class of graphs in scheduling, are chordal.

▶ **Theorem 5.** **MSCB-TC** *in chordal graphs admits a polynomial-time 16.31-approximation.*

The key property of chordal graphs that drives our algorithm is that they admit **perfect elimination orderings**. That is, it is possible to compute an ordering $v_1, v_2, \ldots, v_n$ of all nodes V such that the left-neighborhood $N_\ell(v_i) := \{v_j : j < i \text{ and } v_i v_j \in E\}$ is a clique for all $1 \leq i \leq n$. In fact, a graph is chordal if and only if it admits such an ordering and this ordering can be computed in linear time [8]. Our techniques extend more generally to other classes of graphs.

▶ **Corollary 6.** *Let $\mathcal{G}$ be a hereditary graph class with the same properties as in Corollary 3. Further, for any $G \in \mathcal{G}$ suppose we can compute an ordering $v_1, \ldots, v_n$ of its nodes in polynomial time such that the left-neighborhood $N_\ell(v_i)$ can be covered by a constant number of cliques in polynomial time. Then* **MSCB-TC** *has a constant-factor when restricted to inputs whose graphs lie in $\mathcal{G}$.*

For example, such an ordering exists for unit disk graphs (with each left-neighborhood being covered by ≤ 3 cliques) [13]. Such an ordering can be also found for circular arc graphs with each left-neighborhood being covered by ≤ 2 cliques, i.e. consider the coloring algorithm mentioned above: if one orders the arcs spanning the selected point and then orders the remaining arcs according to a perfect elimination ordering in the resulting interval graph.

The algorithm from Theorem 5 requires one additional structural result about coloring than the algorithm from Theorem 2, namely Lemma 10 in Section 3. Unfortunately this structural result fails to hold in perfect graphs, which is why Theorem 5 is only for chordal graphs. Still, we are able to recover the following.

▶ **Theorem 7.** *There is an $O(\sqrt{n})$-approximation for* **MSCB-TC** *in perfect graphs.*

While the ratio is quite large, it is at least better than the lower bound in general graphs of $n^{1-\delta}$ for any constant $\delta > 0$, which is inherited from the same hardness for **MSC** [2]. Perhaps it is possible to design a constant-factor approximation for **MSCB-TC** in perfect graphs, we leave this as an open problem.

1.1 Related Work

MSCB has been studied in certain special cases. The most notable example is Coflow Scheduling, which is equivalent to **MSCB** when the input graph is the line graph[2] of a bipartite graph (i.e. at any given time step a matching of the edges/jobs is scheduled). This problem was first introduced in [14] where a 22.34-approximation was given. Later, 4-approximations followed for Coflow Scheduling and generalizations with release times for the jobs [1, 16, 6].

In a matroid setting, the jobs are given as items in the ground set X of a matroid rather than as vertices in a graph and bundles are subsets of items in X. The set of jobs scheduled at any given time must form an independent set of the matroid. A 2-approximation for the problem of minimizing the total weighted completion time of all bundles was given in [12].

MSC itself is much more well studied. As mentioned earlier, a 4-approximation is known in settings where the maximum independent set of the graph (and any induced subgraph) can be computed in polynomial time [2]. Special attention has been given to particular graph classes, in particular a 1.796-approximation is known in interval graphs [10] which was recently extended to an algorithm with the same approximation guarantee for chordal graphs [5]. In the more general setting of perfect graphs, a 3.591-approximation is known [7]. A broader summary of approximation algorithms for **MSC** in special graph classes can be found in the survey article by Halldórsson and Kortsarz [9].

1.2 Organization

After discussing some challenges in adapting previous **MSC** and Coflow Scheduling algorithms to **MSCB**, Section 2 presents our algorithm for **MSCB** and the proofs of Theorem 2 and Corollary 3. Section 3 extends these techniques to **MSCB-TC** and proves Theorem 5

[2] Recall the line graph of $G = (V, E)$ is the graph $L(G) = (E, F)$ where two edges $e, f \in E$ are considered adjacent in $L(G)$ if they share a common endpoint.

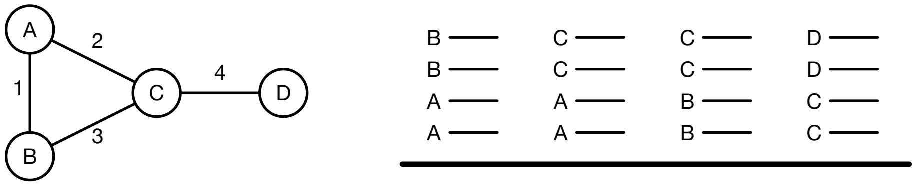


Figure 1 An example of the reduction from an instance of the Maximum Independent Set problem to a collection of intervals using $t = 2$. It is straightforward to verify a subset of nodes is independent if and only if each point on the underlying line is spanned by at most t intervals in the union of their bundles.

and Corollary 6. We also discuss why our **MSCB-TC** algorithm does not extend to perfect graphs in general, point out that we can at least get an $O(\sqrt{n})$-approximation in perfect graphs for **MSCB-TC**, and leave further improvements for future work.

1.3 Challenges

While **MSCB** is a common generalization of **MSC** and Coflow Scheduling, it turns out most techniques used to address these problems fail. For example, a 4-approximation for **MSC** follows by iteratively finding a maximum independent set in the graph of unscheduled tasks in each time step. But for the classic problem of computing the chromatic number of a graph, i.e. the special case of **MSCB** where all nodes are in a single bundle, this can be as bad as an $\Omega(\log n)$-approximation even if the underlying graph is a tree, see Appendix A for a simple example.

Another strategy that is used to get refined approximations for **MSC** is to compute maximum t-colorable subgraphs of G for a geometric sequence of values for t as in [10, 5]. For **MSCB**, one could consider an algorithm that for a geometric sequence of values t will compute a maximum-size subset of bundles $\mathcal{P} \subseteq \{1, 2, \ldots, p\}$ such that all nodes in these bundles, i.e. $\cup_{j \in \mathcal{P}} V_j$ can be scheduled without conflict in t steps. That is, we do not just compute a maximum-size t-colorable induced subgraph of G itself, rather we are concerned with how many clients can have their bundles completed within t steps.

Unfortunately, even in the special case where G is an interval graph, this seems impossible to approximate well.

▶ **Lemma 8.** *Let $G = (V, E)$ be an interval graph and $V_1, V_2, \ldots, V_p$ a partition of V. For any $t \leq |V|$ and any constant $\delta > 0$, there is no $O(|V|^{1/3-\delta})$-approximation algorithm for computing the maximum size $\mathcal{P} \subseteq \{1, 2, \ldots, p\}$ such that $\cup_{j \in \mathcal{P}} V_j$ can be scheduled within t steps unless* **P** $=$ **NP**.

Proof. We reduce from the Maximum Independent Set problem. Let $H = (U, F)$ be an undirected graph. Order F arbitrarily as $e_1, e_2, \ldots, e_{|F|}$. We build an interval graph over the line $[1, 2 \cdot |F|]$. Namely, for each $v \in U$ and each e_i having v as an endpoint, add t copies of the interval $[2 \cdot i - 1, 2 \cdot i]$. Let U_v denote all intervals created from $v \in U$ this way. Note, there are exactly $2 \cdot t$ intervals of the form $[2 \cdot i - 1, 2 \cdot i]$ for each $1 \leq i \leq |F|$, one for each endpoint of e_i. Let G be the resulting interval graph whose nodes/intervals are partitioned as $\{U_v : v \in U\}$. Figure 1 illustrates this reduction.

Let $\mathcal{I}$ be a subset of bundles. It is straightforward to verify the intervals in $\cup_{U_v \in \mathcal{I}} U_v$ can be t-colored (i.e. can be scheduled within t time steps) if and only if $\{v \in U : U_v \in \mathcal{I}\}$ is an independent set in H.

Finally, recall for any constant $\delta > 0$, there is no $|U|^{1-\delta}$-approximation for the maximum independent set problem unless $\mathbf{P} = \mathbf{NP}$ [17]. Since $t \leq n$, then G has $2 \cdot |F| \cdot t \leq O(|U|^3)$ vertices, as required. Thus, an $\alpha = o(|V|^{1/3-\delta/3})$-approximation for the largest number of parts that can be scheduled in t time steps would yield a $o(|U|^{1-\delta})$-approximation for the maximum independent set problem in H. Replacing δ by 3δ (again a constant) yields the result. ◀

Since **MSC** techniques seem to fail for **MSCB**, one could look to techniques successfully used for approximating Coflow Scheduling. Recall Coflow Scheduling is the special case of **MSCB** when the graph G is the line graph $L(H)$ of an undirected graph. A property of Coflow Scheduling that is commonly leveraged to design approximation algorithms is that for each edge e in H (i.e. a job), all other edges that conflict with e share one of two endpoints with e.

For example, the algorithm in [1] solves a time-indexed LP relaxation and uses the familiar trick of greedy scheduling jobs according to their fractional completion time. Their analysis relies on the fact that there are only 2 “reasons” an edge may not be scheduled at a time step (i.e. one of the two endpoints already has an edge at that time step). Also, in [6] a hypergraph matching result by Haxell [11] is used to demonstrate that a good schedule of jobs exists. However, the way this matching result is used in [6] crucially relies on the fact that the formed hyperedges only have 3 nodes, which comes from the fact that each edge of H has only two endpoints.

2 Approximating MSCB in Perfect Graphs

Despite the challenges pointed out in Section 1.3, we are still able to design a constant-factor approximation for **MSCB** in perfect graphs. Our techniques can be seen as a workaround to the problem highlighted in Lemma 8, though we also need to use LPs to do so. That is, we use an LP-relaxation to fractionally schedule the jobs. For a geometrically-increasing sequence of values t, we consider the jobs that are completed to an extent of at least $1/2$ by time t in the fractional solution. The LP constraints will witness that the size of the largest clique among these jobs is $O(t)$. The fact the graph is perfect then allows us to color these jobs with $O(t)$ colors, i.e. to schedule them all within $O(t)$ time steps.

For simplicity of presentation, we suppose $G = (V, E)$ is an interval graph with $n = |V|$ nodes and that $V = \{V_1, V_2, \ldots, V_p\}$. This allows us to write a polynomial-size LP relaxation. The straightforward extension to perfect graphs (albeit with an exponential-size LP) will be discussed at the end of this section. Thus, we suppose each vertex $v \in V$ is associated with an interval $[s_v, t_v] \subseteq \mathbb{R}$. It is a folklore result that we may further assume, without loss of generality, that each endpoint of each interval lies in the set $\{1, 2, \ldots, 2 \cdot n\}$. For each $1 \leq i \leq 2 \cdot n$ the set $C_i := \{v \in V : i \in [s_v, t_v]\}$ is easily seen to be a clique in G. It is also known [8] that every clique of G is a subset of C_i for some $1 \leq i \leq 2 \cdot n$.

Like some previous work in **MSC** [5] and Coflow Scheduling [1, 6], we consider a time-indexed LP relaxation for **MSCB**. For each $v \in V$ and $1 \leq t \leq n$ we let $x_{v,t}$ be a variable indicating v should be colored t and for each $1 \leq k \leq p$ we let f_k be a variable that is intended to be the largest color used to color nodes in V_k (i.e. when all jobs for bundle k are completed). Throughout this section, we adhere to the following indexing conventions: k will be used for bundles, t for time steps/colors, i for points on the underlying line $\{1, 2, \ldots, 2 \cdot n\}$ in the interval graph, and j for indexing geometric groupings of jobs defined in the algorithm.

$$\begin{array}{rrcll}
\textbf{minimize}: & \sum_{k=1}^{p} w_k \cdot f_k & & & \\
\textbf{subject to}: & f_k & \geq & \sum_{t=1}^{n} t \cdot x_{v,t} & \forall\ 1 \leq k \leq p, v \in V_k \\
& \sum_{v \in C_i} x_{v,t} & \leq & 1 & \forall\ 1 \leq t \leq n, \forall\ 1 \leq i \leq 2 \cdot n \\
& \sum_{t=1}^{n} x_{v,t} & = & 1 & \forall\ v \in V \\
& x, f & \geq & 0 &
\end{array} \qquad (\textbf{LP-MSCB})$$

The first constraint says that bundle k is considered finished only after each $v \in V_k$ is completed, and the second constraint ensures that at most one vertex from any clique in the interval graph can be processed at a time. The third ensures each job is completed at some point. Clearly, the optimum value of (**LP-MSCB**) is at most the optimum value of the **MSCB** instance since the natural integer solution corresponding to the optimal **MSCB** solution is feasible for this LP.

2.1 Rounding Algorithm

After computing an optimal solution to **LP-MSCB**, for each job $v \in V$ we let τ_v denote the smallest time t such that $\sum_{t' \leq t} x_{v,t'} \geq 1/2$, i.e. when the LP solution has completed v to an extent of at least $1/2$. Notice by minimality of τ_v we also have $\sum_{t' \geq t} x_{v,t'} = 1 - \sum_{t' < t} x_{v,t'} > 1 - 1/2 = 1/2$.

For a bundle k, we then let $\hat{f}_k := \max_{v \in V_k} \tau_v$ be the minimum time when all jobs in V_k are completed to an extent of at least $1/2$ by the LP solution. As we show below, it is not hard to see $\sum_k \hat{f}_k$ is within a constant factor of the optimal LP solution.

We then employ geometric grouping of the jobs $v \in V$. That is, for each time t in a geometric sequence we form a group with all jobs v having $\tau_v \leq t$. Using properties of the LP solution and interval graphs, we show we can properly color all jobs in each such group with $2 \cdot t$ colors. Concatenating these schedules for the various groups in this geometric sequence completes the algorithm.

To optimize our final ratio, we carefully choose the geometric growth rate and also pick an initial random geometric offset, as has been done in many previous works in minimum-latency problems, e.g. [10]. We could also try a different parameter than $1/2$ as the choice of threshold for the defining values τ_v, but it turns out that $1/2$ is the optimal value for our approach. Also, readers with experience in **MSC** algorithms may wonder about another optimization. Namely, with **MSC** once one has colored a geometric group one may get a slight improvement in the approximation guarantee by optimally ordering the colors so that larger color classes are finished earlier. However, this optimization does not work in our setting since we are concerned with the completion times of bundles and reordering color classes within a group's coloring may not affect the completion time of a bundle.

We let $q > 1$ be a constant. It turns out the optimal setting for q in our algorithm is just e, the base of the natural logarithm. We leave q as an unspecified constant for now and only set it at the point in the analysis where it is apparent that this was the best choice of constant. The precise description of our rounding technique is presented in Algorithm 1.

2.2 Analysis

Recall the sets U_j described in Algorithm 1.

▷ Claim 9. For each j, the jobs in U_j can be scheduled without any conflicts using at most $2 \cdot q^{j+\alpha}$ time steps.

Algorithm 1 **MSCB** Scheduling.

```
Compute an optimal solution $(x, f)$ to (LP-MSCB).
Set $\tau_v$ to the smallest integer such that $\sum_{t \le \tau_v} x_{v,t} \ge 1/2$ for each $v \in V$.
Sample $\alpha \sim [0, 1)$ uniformly.
Let $U_j = \{v \in V : \lfloor q^{j-1+\alpha} \rfloor < \tau_v \le \lfloor q^{j+\alpha} \rfloor\}$ for $0 \le j \le \log_q n$.
for each $U_j$ in increasing order of $j$ do
  Schedule all jobs in $U_j$ within the next $2 \cdot \lfloor q^{j+\alpha} \rfloor$ unused time steps. {Claim 9}
```

Proof. We claim the size of the largest clique contained in U_j is at most $2 \cdot q^{j+\alpha}$. If so, then we can properly color all of U_j using at most $2 \cdot q^{j+\alpha}$ colors because interval graphs are perfect.

First, consider any $v \in U_j$ and say $v \in V_k$. Then $\tau(v) \le \hat{f}_k \le q^{j+\alpha}$, so

$$\sum_{t \le q^{j+\alpha}} x_{v,t} \ge \sum_{t \le \tau(v)} x_{v,t} \ge \frac{1}{2}.$$

Now consider any point $i \in \{1, 2, \ldots, 2 \cdot n\}$ on the interval, our goal is to show $|U_j \cap C_i| \le 2 \cdot q^{j+\alpha}$. Letting $X_{i,j} := \sum_{v \in U_j \cap C_i} \sum_{t \le q^{j+\alpha}} x_{v,t}$, summing the above bound over $v \in |U_j \cap C_i|$ shows $X_{i,j} \ge |U_j \cap C_i|/2$.

On the other hand, by the LP constraints we also have

$$X_{i,j} = \sum_{t \le q^{j+\alpha}} \sum_{v \in U_j \cap C_i} x_{v,t} \le \sum_{t \le q^{j+\alpha}} \sum_{v \in C_i} x_{v,t} \le \sum_{t \le q^{j+\alpha}} 1 = q^{j+\alpha}.$$

From these two bounds on $X_{i,j}$ we have $|U_j \cap C_i|/2 \le X_{i,j} \le q^{j+\alpha}$ so $|U_j \cap C_i| \le 2 \cdot q^{j+\alpha}$.

Finally, consider any clique $C \subseteq U_j$. Any clique of G is contained in a clique of the form C_i so $C \subseteq U_j \cap C_i$. Thus, we have $|C| \le |U_j \cap C_i| \le 2 \cdot q^{j+\alpha}$, that is the bound holds for all cliques $C \subseteq U_j$. ◁

Next, we show each bundle k finishes within time $O(\hat{f}_k)$. Fix any such bundle k and pick any $v_k \in V_k$ with $\tau_{v_k} = \hat{f}_k$. For any value α sampled by the algorithm, the completion time of bundle k is upper bounded by the completion time of all jobs in the bundle U_j that contains v_k. This is because no job of V_k will be placed in a bundle $U_{j'}$ having $j' > j$ and because we concatenated the schedules for the various buckets in increasing order of j.

Since $0 \le \alpha \le 1$ then there is some integer j_k such that $v_k \in U_{j_k - 1}$ or $k \in U_{j_k}$, depending on the value of α. The breaking point between these two events occurs at $\alpha = \log_q \hat{f}_k - (j_k - 1)$. Letting T_α be the quantity $2 \cdot q^{j+\alpha}$ for the group $j \in \{j_k - 1, j_k\}$ that v_k is assigned to for a given α, we have:

$$T_\alpha \le \begin{cases} 2 \cdot q^{j_k - 1 + \alpha} & k \in U_{j_k - 1} \\ 2 \cdot q^{j_k + \alpha} & k \in U_{j_k} \end{cases}$$

Since we concatenate the schedules for the groups U_j in increasing order of index j and each group U_j is completed by time $2 \cdot q^{j+\alpha}$ (Claim 9), then for any j each job in U_j will be completed by time $\sum_{j'=1}^{j} 2 \cdot q^{j'+\alpha} \leq \frac{2 \cdot q}{q-1} \cdot q^{j+\alpha}$. Therefore we have

$$\begin{aligned}
\mathbf{E}_{\alpha \sim [0,1)}[T_\alpha] &= \int_0^1 T_\alpha \; d\alpha \\
&= \frac{2 \cdot q}{q-1} \cdot \left(\int_0^{\log_q \hat{f}_k - (j_k - 1)} q^{j_k + \alpha} \; d\alpha + \int_{\log_q \hat{f}_k - (j_k-1)}^{1} q^{j_k - 1 + \alpha} \; d\alpha \right) \\
&= \frac{2 \cdot q}{q-1} \cdot \left(\frac{q^{j_k+\alpha}}{\ln q} \bigg|_0^{\log_q \hat{f}_k - (j_k-1)} + \frac{q^{j_k-1+\alpha}}{\ln q} \bigg|_{\log_q \hat{f}_k - (j_k-1)}^{1} \right) \\
&= \frac{2 \cdot q}{\ln q} \cdot \tau_{v_k} = \frac{2 \cdot q}{\ln q} \cdot \hat{f}_k
\end{aligned}$$

At this point, we see the optimal choice of q is $e \approx 2.717$, the base of the natural logarithm. Setting q to e yields

$$\mathbf{E}_{\alpha \sim [0,1)}[T_\alpha] = \frac{2 \cdot q}{\ln q} \cdot \hat{f}_k = 2 \cdot e \cdot \hat{f}_k$$

We complete the proof by bounding f_k by $O(\hat{f}_k)$. Recalling $\sum_{t \geq \tau_{v_k}} x_{v_k,t} \geq 1/2$, we see

$$f_k \geq \sum_t t \cdot x_{v_k,t} \geq \sum_{t \geq \tau_{v_k}} t \cdot x_{v_k,t} \geq \tau_v \cdot \sum_{t \geq \tau_{v_k}} x_{v_k,t} \geq \frac{\tau_{v_k}}{2} = \frac{\hat{f}_k}{2}$$

To put this all together, by Claim 9 the completion time of a bundle k is at most T_α. In expectation over the random choice of α, this is at most $2 \cdot e \cdot \hat{f}_k$. Finally, from the bound directly above we see the expected completion time of a bundle is then at most $4 \cdot e \cdot f_k$. Thus, the expected total completion time of all bundles is at most $4 \cdot e \leq 10.874$ times the optimum value of (**LP-MSCB**).

2.3 Extensions

Perfect Graphs. The only change to the LP is that the second collection of constraints is replaced by the following more general constraints:

$$\sum_{v \in C} x_{v,t} \leq 1 \quad \forall\, t, \forall \text{ cliques } C \text{ of } G \tag{1}$$

In general there are exponentially many cliques (and even exponentially-many maximal cliques) in a perfect graph. Still, these constraints can be separated in polynomial time for perfect graphs (Theorem 67.6 in [15]) meaning the LP can still be solved optimally in polynomial time.

The rest of the proof carries through essentially without modification: the size of a maximum clique in U_j is still bounded to be at most $2 \cdot q^{j+\alpha}$. That is, let $C \subseteq U_j$ be a clique. Since each $v \in U_j$ has $\tau_v \leq \lfloor q^{j+\alpha} \rfloor$ then

$$\sum_{v \in C} \sum_{t \leq q^{j+\alpha}} x_{v,t} \geq \sum_{v \in C} \frac{1}{2} = |C|/2.$$

On the other hand, by the more general clique constraints (1) we have

$$\sum_{t \leq q^{j+\alpha}} \sum_{v \in C} x_{v,t} \leq \sum_{t \leq q^{j+\alpha}} 1 \leq q^{j+\alpha}.$$

Since G is perfect, then by definition U_j can be colored using at most $2 \cdot q^{j+\alpha}$ colors and such a coloring can be done in polynomial time (Corollary 67.2c[15]). The rest of the analysis is unchanged, thus the full form of Theorem 2 is proven.

Derandomizing. It is simple to efficiently derandomize our approach. We simply list all break points α of the form $\log_q \hat{f}_k - (j_k - 1)$ over all bundles k and try all α between these break points. Our algorithm is deterministic once α is given and these break points are the only values of α where the behavior of the algorithm changes. Taking the best solution found over all such α is surely no worse than the expected cost of the returned solution when choosing α randomly

Extensions to Other Graph Classes. For Corollary 3, the assumptions mean we can approximately separate the clique constraints $\sum_{v \in C} x_{v,t} \leq 1$ in polynomial time ultimately leading to an efficient algorithm that finds an LP solution with cost at most OPT where all constraints hold except perhaps these new clique constraints. Instead, we would have $\sum_{v \in C} x_{v,t} \leq c$ where c is the approximation factor of computing a maximum-weight clique in G.

The approximate relationship between maximum cliques and the chromatic number of graphs satisfying the assumptions of Corollary 3 allow us to conclude U_j can be colored with at most $c' \cdot q^{i+\alpha}$ colors where c' is also a constant. Carrying this term through the rest of the analysis shows the algorithm is an $O(1)$-approximation.

3 MSCB with Task Concurrencies

Recall in **MSCB-TC**, the bundles form a *partition* $V_1, \ldots, V_p$ of P and for each bundle k we have a bound d_k on the number of jobs in V_k that can be scheduled at any single time. This models settings where clients can only deliver/retrieve a bounded number of their jobs at any single time. Also, recall that we assume G is a chordal graph.

The new algorithm starts with (**LP-MSCB**) except the cliques C_i used to define the constraints are the polynomially-many maximal cliques of G [8] (which can be enumerated in polynomial time) and two additional classes of constraints are added. First, for any bundle $1 \leq k \leq p$ and any time t we add the constraints

$$\sum_{v \in V_k} x_{v,t} \leq d_k.$$

That is, at any given time a maximum of d_k jobs for bundle k can be processed. We call these *concurrency constraints*. Next, For any bundle $1 \leq k \leq p$ we add the constraints

$$f_k \geq \lceil |V_k|/d_k \rceil$$

which enforces the trivial lower bound that $\lceil |V_k|/d_k \rceil$ time steps are required to finish bundle k even if we processed d_k of its jobs per step. Note, without this bound the LP could cheat in the following way: if $d_k = 1$ and $V_k = \{v_1, \ldots, v_m\}$ we could set $x_{v_i,t} = 1/m$ for all $1 \leq i \leq m$ and $1 \leq t \leq m$ which would permit us to set $f_k = (m+1)/2$ whereas an integer solution would clearly require $f_k \geq m$.

For the rest of this section, by "schedule" we mean a proper coloring of G with the additional constraint that for any bundle k and any time t we have at most d_k jobs in V_k being colored with t.

We need to make some minor modifications to the algorithm. First, we now define $\hat{f}_k := \max\{\lceil |V_k|/d_k \rceil, \max_{v \in V_k} \tau_v\}$. Next, we change U_j to be

$$U_j = \{k : \lfloor q^{j-1+\alpha} \rfloor < \hat{f}_k \leq \lfloor q^{j+\alpha} \rfloor\}.$$

Finally, when we color U_j, we will ensure that the new concurrency constraints are satisfied with this coloring. The following structural result enables us to do this while limiting the loss in the final approximation guarantee. Here, we are letting $\chi(G)$ denote the chromatic number of G.

▶ **Lemma 10.** *Let $G = (V, E)$ be a chordal graph whose vertices are partitioned as $V_1, \ldots, V_p$. Further, for each $1 \leq k \leq p$ let $d_k \geq 1$ be an integer. In polynomial time, we can compute a proper coloring of G using at most $\chi(G) + \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil - 1$ colors such that for each $1 \leq k \leq p$, no color appears more than d_k times among nodes in V_k.*

Proof. Recall that a graph is a chordal graph if and only if it has a perfect elimination ordering, i.e. an ordering $v_1, v_2, \ldots, v_n$ such that for each $1 \leq j \leq n$, the *left-neighborhood* $N_\ell(v_j) = \{i < j : v_i v_j \in E\}$ of each node is a clique and that this ordering can be computed in linear time [8].

To compute the coloring we need, process the nodes v_i in this order. When coloring v_i, we simply avoid using a color already assigned to a node in $N_\ell(v_i)$ or already assigned to d_k nodes in the same part V_k as v_i. This can be done if we allow $\chi(G) + \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil - 1$ colors. ◀

We briefly remark that Lemma 10 is tight even for interval graphs where $d_k = 1$ for all V_k. Consider the case where V_1 consists of m jobs whose corresponding intervals are $[1, 2], [3, 4], [5, 6], \ldots, [2m - 1, 2m]$ and $V_2, \ldots, V_p$ each consists of a single job whose corresponding interval is $[1, 2m]$. The chromatic number is exactly p but no two jobs can receive the same color since the only non-intersecting pairs of intervals have their corresponding jobs in the same bundle V_1. Therefore, $|V_1| + |V_2| + \ldots + |V_p| = p + m - 1$ colors are required.

Towards coloring U_j, we define $\mathcal{V}_j = \{k : V_k \cap U_j\} \neq \emptyset$ to be all bundles having some job in U_j and then we let $S_j = \max_{k \in \mathcal{V}_j} \lceil |V_k \cap U_j|/d_k \rceil$. Since $\lceil |V_k \cap U_j|/d_k \rceil$ is a lower bound on the time required to finish all jobs $V_k \cap U_j$ due to the task concurrency constraint for bundle k, we have that S_j is another lower bound for the time needed to complete all jobs in the set U_j. The new LP constraints help assert this lower bound as well.

▶ **Lemma 11.** *For each group j, $S_j \leq q^{j+\alpha}$.*

Proof. This is demonstrated by leveraging the additional constraint incorporated into our LP. For every $k \in \mathcal{V}_j$, we know that $\frac{|V_k|}{d_k} \leq \hat{f}_k$. Furthermore, based on the new definition of U_j it is clear that $\hat{f}_k \leq q^{j+\alpha}$. Consequently, $\frac{|V_k|}{d_k}$ is less than equal to $q^{j+\alpha}$, implying $|V_k \cap U_j| \leq q^{j+\alpha} \cdot d_k$. Therefore, it follows that $S_j \leq q^{j+\alpha}$. ◀

As with the **MSCB** approximation, the maximum clique size in U_j is at most $2 \cdot q^{j+\alpha}$. Further, we have just shown $|V_k \cap U_j| \leq q^{j+\alpha} \cdot d_k$ for any $k \in \mathcal{V}_j$. So Lemma 10 means there is a proper coloring of U_j using at most $3 \cdot q^{j+\alpha}$ colors such that no bundle in $\mathcal{V}_k$ has more than d_k jobs colored with the same color.

The rest of the analysis is similar to the analysis of the algorithm for **MSCB** except the approximation ratio has changed since we used $3 \cdot q^{\alpha+j}$ colors instead of $2 \cdot q^{\alpha+j}$ colors to color each U_j. Thus, it is a $6 \cdot e \leq 16.31$-approximation.

Corollary 6 essentially follows by how Corollary 3 followed from Theorem 2 but using a more general form of Lemma 10. Namely, if there is an ordering of the nodes $v_1, v_2, \ldots, v_n$ such that the left-neighborhood $N_\ell(v_i) = \{v_j : v_i v_j \in E, j < i\}$ of any node v_i can be covered with $R = O(1)$ cliques then we can find a proper coloring of G using at most $R \cdot \chi(G) + \max_k \lfloor |V_k|/d_k \rfloor$ colors by picking the lowest available color not appearing in the left-neighborhood of v_i that is also not used d_k times in the part V_k with v_i.

4 MSCB-TC in Perfect Graphs – A Barrier

Lemma 10 fails to hold in perfect graphs even within any constant factor. That is, it may require $\Theta(\sqrt{n}) \cdot \max\{\chi(G), \max_k |V_k|/d_k\}$ colors to even if $d_k = 1$ for all k. Consider the following simple example on $n = N^2$ nodes for some integer N. The graph $G_N = (V, E)$ is partitioned into sets $V_1, \ldots, V_N$ and each V_k has exactly N nodes. We have an edge between any pair of nodes in different parts, but each part is an independent set.

It is easy to see such graphs are perfect. More generally, a graph that is partitioned into b nonempty independent sets and has all possible edges between these parts has chromatic number b and maximum clique size b (pick one node from each part). Since any induced subgraph of our graph G_N is of this form, then G_N is also perfect.

But any coloring satisfying task concurrency limits of $d_k = 1$ for all parts must in fact use n colors. Two nodes in different parts cannot receive the same color because they are connected by an edge and two nodes in the same part cannot receive the same color because the task concurrency limit is 1. Yet, $\chi(G) = N = \sqrt{n}$ and the maximum size of a part is also $N = \sqrt{n}$.

Still, this is the worst case. The following variation of Lemma 10 leads to an $O(\sqrt{n})$-approximation for **MSCB-TC** in perfect graphs.

▶ **Lemma 12.** *Let $G = (V, E)$ be a graph whose vertices are partitioned as $V_1, \ldots, V_p$. Further, for each $1 \leq k \leq p$ let $d_k \geq 1$ be an integer. There is a proper coloring of G using at most $\sqrt{n} \cdot \max\left\{\chi(G), \max_k \left\lceil \frac{|V_k|}{d_k} \right\rceil\right\}$ colors such that for each $1 \leq k \leq p$, no color appears more than d_k times among nodes in V_k. Such a coloring can be computed in polynomial time if G can be optimally colored in polynomial time.*

Proof. If $\max_k \lceil |V_k|/d_k \rceil \geq \sqrt{n}$, then the trivial n-coloring (i.e. all nodes get different colors) suffices. Otherwise, consider a proper coloring $\sigma : V \to \{1, 2, \ldots, \chi(G)\}$ of G. Order the nodes $v^k_1, v^k_2, \ldots, v^k_{|V_k|}$ arbitrarily in each part V_k.

Recolor a vertex v^k_i with the pair $(\chi(v^k_i), \lfloor i/d_k \rfloor)$. Clearly, this is a proper coloring since the first components of the new colors of nodes will differ on any edge of G. Further, at most d_k nodes in V_k will have the same second part of the pair describing their color. The number of colors used is $\chi(G) \cdot \max_k \lceil |V_k|/d_k \rceil \leq \chi(G) \cdot \sqrt{n}$, as required.

Finally, this can be done in polynomial time if we can compute a coloring of G with $\chi(G)$ colors in polynomial time. ◀

Using this in place of Lemma 10 yields an $O(\sqrt{n})$-approximation for **MSCB-TC** in perfect graphs. This proves Theorem 7.

5 Conclusion

We have given the first constant-factor approximations for **MSCB** in a large variety of graph classes including perfect graphs and unit-disc graphs. Our techniques extend to give the first constant-factor approximations for **MSCB-TC** in certain graphs like chordal graphs, interval graphs, and unit-disc graphs.

It would be interesting to see what other graph classes admit constant-factor approximations for **MSCB** and, perhaps, also for **MSCB-TC**. Another interesting direction would be to get a purely combinatorial constant-factor approximation for **MSCB** in certain graph classes, i.e. one that avoids solving a linear program. Such algorithms exist for **MSC** in many cases, e.g. [2, 10]. One barrier is that it seems hard to approximate the maximum number of bundles that can be completed in a given number of time steps even in simple graph classes like interval graphs (Lemma 8). Perhaps a bicriteria approximation could be designed to circumvent this hardness, it would immediately lead to an $O(1)$-approximation through standard minimum latency arguments.

References

1 S. Ahmadi, S. Khuller, M. Purohit, and S. Yang. On scheduling coflows – (extended abstract). In *Proceedings of 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 13–24, 2017.

2 A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

3 B. K. Bhattacharya and D. Kaller. An $o(m + n \log n)$ algorithm for the maximum-clique problem in circular-arc graphs. *Journal of Algorithms*, 25(2):336–358, 1997.

4 B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

5 I. DeHaan and Z. Friggstad. Approximate minimum sum colorings and maximum k-colorable subgraphs of chordal graphs. In *Algorithms and Data Structures Symposium (WADS)*, pages 326–339, 2023.

6 T. Fukunaga. Integrality gap of time-indexed linear programming relaxation for coflow scheduling. In *In Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques(APPROX)*, volume 245, pages 36:1–36:13, 2022.

7 R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved bounds for sum multicoloring and scheduling dependent jobs with minsum criteria. In *Approximation and Online Algorithms*, pages 68–82, 2005.

8 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

9 M. M. Halldórsson and G. Kortsarz. Algorithms for chromatic sums, multicoloring, and scheduling dependent jobs. In *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 671–684. Chapman and Hall/CRC, 2018.

10 M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37:187–209, 2003.

11 P. E. Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11:245–248, 1995.

12 S. Im, B. Moseley, K. Pruhs, and M. Purohit. Matroid Coflow Scheduling. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 145:1–145:13, 2019.

13 M. J. P. Peeters. On coloring j-unit sphere graphs. Research Memorandum FEW 512, Tilburg University, School of Economics and Management, 1991.

14 Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 294–303, 2015.

15 A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency.* Springer, 2003.

16 M. Shafiee and J. Ghaderi. An improved bound for minimizing the total weighted completion time of coflows in datacenters. *IEEE/ACM Transactions on Networking*, 26(4):1674–1687, 2018.

17 D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. of ACM Symposium on Theory of Computing (STOC 2006)*, pages 681–690, 2006.

A Greedy Coloring in Trees

We point out the greedy algorithm that iteratively picks a maximum independent set to color a graph may be as bad as an $\Omega(\log n)$-approximation. We point this out to show that the greedy 4-approximation for unweighted **MSC** does not extend to our setting which includes, as a special case, computing the chromatic number of a graph. While this seems to be well known in the community, we are unaware of a particular reference for such an example so we provide a simple construction here for completeness.

Let T_0 be the trivial tree with a single node. Inductively for $i \geq 1$ let T_i be constructed by attaching 2 leaf nodes to each node of T_{i-1}. So the number of nodes in T_i is 3^i.

The only maximum independent set in T_i is the set of all leaves in T_i (which clearly forms an independent set). To see this, let I be an independent set that includes an internal vertex of T_i (i.e. a node of T_{i-1}). Neither leaf that was attached to v to form T_i is in I because I is an independent set. But then we get a larger independent set by removing v from I and adding in the two associated leaves.

The greedy algorithm to compute a maximum independent set in T_i will first pick all of its leaves. Removing them leaves tree T_{i-1}. So by induction, with the base case $i = 0$ clearly requiring a single iteration to color, the number of iterations will be $i + 1$. Since $i + 1$ is logarithmic in the size of T_i (i.e. $n := 3^i$) and since the chromatic number of $T_i, i \geq 1$ is 2 (as is true for any tree with at least one edge), this is an $\Omega(\log n)$-approximation for the chromatic number of a tree.

Stability in Graphs with Matroid Constraints

Fedor V. Fomin
University of Bergen, Norway

Petr A. Golovach
University of Bergen, Norway

Tuukka Korhonen
University of Bergen, Norway

Saket Saurabh
University of Bergen, Norway
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

We study the following Independent Stable Set problem. Let G be an undirected graph and $\mathcal{M} = (V(G), \mathcal{I})$ be a matroid whose elements are the vertices of G. For an integer $k \geq 1$, the task is to decide whether G contains a set $S \subseteq V(G)$ of size at least k which is independent (stable) in G and independent in $\mathcal{M}$. This problem generalizes several well-studied algorithmic problems, including Rainbow Independent Set, Rainbow Matching, and Bipartite Matching with Separation. We show that

- When the matroid $\mathcal{M}$ is represented by the independence oracle, then for any computable function f, no algorithm can solve Independent Stable Set using $f(k) \cdot n^{o(k)}$ calls to the oracle.
- On the other hand, when the graph G is of degeneracy d, then the problem is solvable in time $\mathcal{O}((d+1)^k \cdot n)$, and hence is FPT parameterized by $d+k$. Moreover, when the degeneracy d is a constant (which is not a part of the input), the problem admits a kernel polynomial in k. More precisely, we prove that for every integer $d \geq 0$, the problem admits a kernelization algorithm that in time $n^{\mathcal{O}(d)}$ outputs an equivalent framework with a graph on $dk^{\mathcal{O}(d)}$ vertices. A lower bound complements this when d is part of the input: Independent Stable Set does not admit a polynomial kernel when parameterized by $k+d$ unless NP $\subseteq$ coNP /poly. This lower bound holds even when $\mathcal{M}$ is a partition matroid.
- Another set of results concerns the scenario when the graph G is chordal. In this case, our computational lower bound excludes an FPT algorithm when the input matroid is given by its independence oracle. However, we demonstrate that Independent Stable Set can be solved in $2^{\mathcal{O}(k)} \cdot \|\mathcal{M}\|^{\mathcal{O}(1)}$ time when $\mathcal{M}$ is a linear matroid given by its representation. In the same setting, Independent Stable Set does not have a polynomial kernel when parameterized by k unless NP $\subseteq$ coNP /poly.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Matroids and greedoids; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases frameworks, independent stable sets, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.22

Related Version *Full Version*: `https://arxiv.org/abs/2404.03979`

Funding The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528) and the European Research Council (ERC) via grant LOPPRE, reference 819416.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 22; pp. 22:1–22:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We initiate the algorithmic study of computing stable (independent) sets in frameworks. The term *framework*, also known as *pregeometric graph* [28, 29], refers to a pair $(G, \mathcal{M})$, where G is a graph and $\mathcal{M} = (V(G), \mathcal{I})$ is a matroid on the vertex set of G. We remind the reader that pairwise nonadjacent vertices of a graph form a *stable* or *independent* set. To avoid confusion with independence in matroids, we consistently use the term “stable set” throughout the paper. Whenever we mention independence, it is in reference to independence with respect to a matroid. We consider the following problem.

Independent Stable Set

Input: A framework $(G, \mathcal{M})$ and an integer $k \geq 0$.

Task: Decide whether G has vertex set $S \subseteq V(G)$ of size at least k that is stable in G and independent in $\mathcal{M}$.

The Independent Stable Set problem encompasses several well-studied problems related to stable sets.

When $\mathcal{M}$ is a uniform matroid with every k-element subset of $V(G)$ forming a basis, the Independent Stable Set problem seeks to determine whether a graph G contains a stable set of size at least k. This is the classic Stable Set problem.

For a partition matroid $\mathcal{M}$ whose elements are partitioned into k blocks and independent sets containing at most one element from each block, Independent Stable Set transforms into the rainbow-independence (or Rainbow-Stable Set) problem. To express this problem in graph terminology, consider a graph G with a vertex set $V(G)$ colored in k colors. A set of vertices S is termed *rainbow-independent* if it is stable in G and no color occurs in S more than once [3, 25]. This concept is also known in the literature as an *independent transversal* [19, 18, 23] and an independent system of representatives [2].

Rainbow-independence generalizes the well-studied combinatorial concept of rainbow matchings [1, 13]. (Note that a matching in a graph is a stable set in its line graph.) It also has a long history of algorithmic studies. In the Rainbow Matching problem, we are given a graph G, whose edges are colored in q colors, and a positive integer k. The task is to decide whether a matching of size at least k exists whose edges are colored in distinct colors. Itai, Rodeh, and Tanimoto in [24] established that Rainbow Matching is NP-complete on bipartite graphs. Le and Pfender [26] strongly enhanced this result by showing that Rainbow Matching is NP-complete even on paths and complete graphs. Gupta et al. [21] considered the parameterized complexity of Rainbow Matching. They gave an FPT algorithm of running time $2^k \cdot n^{\mathcal{O}(1)}$. They also provided a kernel with $\mathcal{O}(k^2\Delta)$ vertices, where Δ is the maximum degree of a graph. Later, in [22], the same set of authors obtained a kernel with $\mathcal{O}(k^2)$ vertices for Rainbow Matching on general graphs.

When $\mathcal{M}$ is a transversal matroid, Independent Stable Set transforms into the Bipartite Matching with Separation problem [30]. In this variant of the maximum matching problem, the goal is to determine whether a bipartite graph H contains a matching of size k with a separation constraint: the vertices on one side lie on a path (or a grid), and two adjacent vertices on a path (or a grid) are not allowed to be matched simultaneously. This problem corresponds to Independent Stable Set on a framework $(G, \mathcal{M})$, where G is a path (or a grid) on vertices U, and $\mathcal{M}$ is a transversal matroid of the bipartite graph $H = (U, W, E_H)$ whose elements are U, and the independent subsets are sets of endpoints of matchings of H. Manurangsi, Segal-Halevi, and Suksompong in [30] proved that Bipartite Matching with Separation is NP-complete and provided approximation algorithms.

Stable Set is a notoriously difficult computational problem. It is well-known to be NP-complete and W[1]-complete when parameterized by k [9]. On the other hand, Stable Set is solvable in polynomial time on perfect graphs [20]. When it comes to parameterized algorithms and kernelization, Stable Set is known to be FPT and to admit polynomial (in k) kernel on classes of sparse graphs, like graphs of bounded degree or degeneracy [10]. The natural question is which algorithmic results about the stable set problem could be extended to Independent Stable Set.

- We commence with a lower bound on Independent Stable Set. Theorem 2 establishes that when the matroid in a framework is represented by the independence oracle, for any computable function f, no algorithm can solve Independent Stable Set using $f(k) \cdot n^{o(k)}$ calls to the oracle. Moreover, we show that the lower bound holds for frameworks with bipartite, chordal, claw-free graphs, and AT-free graphs for which the classical Stable Set problem can be solved in polynomial time. While the usual bounds in parameterized complexity are based on the assumption FPT $\neq$ W[1], Theorem 2 rules out the existence of an FPT algorithm for Independent Stable Set parameterized by k unconditionally.
- In Section 4, we delve into the parameterized complexity of Independent Stable Set when dealing with frameworks on d-degenerate graphs. The first result of this section, Theorem 3, demonstrates that the problem is FPT when parameterized by $d + k$, by providing an algorithm of running time $\mathcal{O}((d + 1)^k \cdot n)$. Addressing the kernelization aspect, Theorem 5 reveals that when d is a constant, Independent Stable Set on frameworks with graphs of degeneracy at most d, admits a kernel polynomial in k. More precisely, we prove that for every integer $d \geq 0$, the problem admits a kernelization algorithm that in time $n^{\mathcal{O}(d)}$ outputs an equivalent framework with a graph on $dk^{\mathcal{O}(d)}$ vertices. This is complemented by Theorem 6, establishing that Independent Stable Set on frameworks with d-degenerate graphs and partition matroids lacks a polynomial kernel when parameterized by $k + d$ unless NP $\subseteq$ coNP /poly.

  Shifting the focus to the stronger maximum vertex degree Δ parameterization, Theorem 4 establishes improved kernelization bounds. Specifically, Independent Stable Set admits a polynomial kernel on frameworks that outputs an equivalent framework with a graph on at most $k^2\Delta$ vertices.
- When it comes to perfect graphs, there is no hope of polynomial or even parameterized algorithms with parameter k: Rainbow-Stable Set is already known to be NP-complete and W[1]-complete when parameterized by k on bipartite graphs by the straightforward reduction from the dual Mulitcolored Biclique problem [9]. Also, the unconditional lower bound from Theorem 2 holds for bipartite and chordal graphs if the input matroids are given by the independence oracles.

  Interestingly, it is still possible to design FPT algorithms for frameworks with chordal graphs when the input matroids are given by their representations. In Theorem 8, we show that Independent Stable Set can be solved in $2^{\mathcal{O}(k)} \cdot \|A\|^{\mathcal{O}(1)}$ time by a one-sided error Monte Carlo algorithm with false negatives on frameworks with chordal graphs and linear matroids given by their representations A. When it concerns kernelization, Theorem 9 shows that Independent Stable Set on frameworks with chordal graphs and partition matroids does not admit a polynomial kernel when parameterized by k unless NP $\subseteq$ coNP /poly.

2 Preliminaries

In this section, we introduce the basic notation used throughout the paper and provide some auxiliary results.

Graphs. We use standard graph-theoretic terminology and refer to the textbook of Diestel [11] for missing notions. We consider only finite undirected graphs. For a graph G, $V(G)$ and $E(G)$ are used to denote its vertex and edge sets, respectively. Throughout the paper, we use n to denote the number of vertices if it does not create confusion. For a graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X. We denote by $G - X$ the graph obtained from G by the deletion of every vertex of X (together with incident edges). For $v \in V(G)$, we use $N_G(v)$ to denote the *(open) neighborhood* of v, that is, the set of vertices of G that are adjacent to v; $N_G[v] = N_G(v) \cup \{v\}$ is the *closed neighborhood* of v. For a set of vertices X, $N_G(X) = \big(\bigcup_{v\in X} N_G(v)\big) \setminus X$ and $N_G[X] = \bigcup_{v\in X} N_G[v]$. We use $d_G(v) = |N_G(v)|$ to denote the *degree* of v; $\delta(G)$ and $\Delta(G)$ denote the minimum and maximum degree of a vertex in G, respectively. For a nonnegative integer d, G is *d-degenerate* if for every subgraph H of G, $\delta(H) \leq d$. Equivalently, a graph G is d-degenerate if there is an ordering $v_1, \ldots, v_n$ of the vertices of G, called *elimination ordering*, such that $d_{G_i}(v_i) \leq d$ for every $i \in \{1, \ldots, n\}$ where $G_i = G[\{v_i, \ldots, v_n\}]$. Given a d-degenerate graph G, the elimination ordering can be computed in linear time [32]. The *degeneracy* of G is the minimum d such that G is d-degenerate. We remind that a graph G is *bipartite* if its vertex set can be partitioned into two sets V_1 and V_2 in such a way that each edge has one endpoint in V_1 and one endpoint in V_2. A graph G is *chordal* if it has no induced cycles on at least four vertices. A graph G is said to be *claw-free* if it does not contain the *claw* graph $K_{1,3}$ as an induced subgraph. An independent set of three vertices such that each pair can be joined by a path avoiding the neighborhood of the third is called an *asteroidal triple* (AT). A graph is *AT-free* if it does not contain asteroidal triples.

Matroids. We refer to the textbook of Oxley [34] for the introduction to Matroid Theory. Here we only briefly introduce the most important notions.

▶ **Definition 1.** *A pair $\mathcal{M} = (V, \mathcal{I})$, where V is a* ground set *and $\mathcal{I}$ is a family of subsets, called* independent sets *of $\mathcal{M}$, is a* matroid *if it satisfies the following conditions, called* independence axioms*:*

(I1) $\emptyset \in \mathcal{I}$,
(I2) *if $X \subseteq Y$ and $Y \in \mathcal{I}$ then $X \in \mathcal{I}$,*
(I3) *if $X, Y \in \mathcal{I}$ and $|X| < |Y|$, then there is $v \in Y \setminus X$ such that $X \cup \{v\} \in \mathcal{I}$.*

We use $V(\mathcal{M})$ and $\mathcal{I}(\mathcal{M})$ to denote the ground set and the family of independent sets of $\mathcal{M}$, respectively, unless $\mathcal{M}$ is clear from the context. An inclusion-maximal set of $\mathcal{I}$ is called a *base*; it is well-known that all bases of $\mathcal{M}$ have the same cardinality. A function $r\colon 2^V \to \mathbb{Z}_{\geq 0}$ such that for every $X \subseteq V$,

$$r(X) = \max\{|Y|\colon Y \subseteq X \text{ and } Y \in \mathcal{I}\}$$

is called the *rank function* of $\mathcal{M}$. The *rank of* $\mathcal{M}$, denoted $r(\mathcal{M})$, is $r(V)$; equivalently, the rank of $\mathcal{M}$ is the size of any base of $\mathcal{M}$. Let us remind that a set $X \subseteq V$ is independent if and only if $r(X) = |X|$. The *closure* of a set X is the set $\mathsf{cl}(X) = \{v \in V\colon r(X \cup \{v\}) = r(X)\}$. The matroid $\mathcal{M}' = (V \setminus X, \mathcal{I}')$, where $\mathcal{I}' = \{Y \in \mathcal{I}\colon Y \subseteq V \setminus X\}$, is said to be obtained from $\mathcal{M}$ by the *deletion* of X. The *restriction* of $\mathcal{M}$ to $X \subseteq V$ is the matroid obtained from $\mathcal{M}$ by the deletion of $V \setminus X$. If X is an independent set then the matroid $\mathcal{M}'' = (V \setminus X, \mathcal{I}'')$, where $\mathcal{I}'' = \{Y \subseteq V \setminus X\colon Y \cup X \in \mathcal{I}\}$, is the *contraction* of $\mathcal{M}$ by X. For a positive integer k, the *k-truncation* of $\mathcal{M} = (V, \mathcal{I})$ is the matroid $\mathcal{M}'$ with the same ground set V such that $X \subseteq V$ is independent in $\mathcal{M}'$ if and only if $X \in \mathcal{I}$ and $|X| \leq k$. Because in Independent Stable Set, we are interested only in independent sets of size at most k, we assume throughout our paper that the rank of the input matroids is upper bounded by k. Otherwise, we replace $\mathcal{M}$ by its k-truncation.

In our paper, we assume in the majority of our algorithmic results that the input matroids in instances of Independent Stable Set are given by independence oracles. An *independence oracle* for $\mathcal{M}$ takes as its input a set $X \subseteq V$ and correctly returns either yes or no in unit time depending on whether X is independent or not. We assume that the memory used to store oracles does not contribute to the input size; this is important for our kernelization results. Notice that given an independence oracle, we can greedily construct an inclusion-maximal independent subset of X and this can be done in $\mathcal{O}(|X|)$ time. Note also that the oracle for $\mathcal{M}$ can be trivially transformed to an oracle for the k-truncation of $\mathcal{M}$.

Our computational lower bounds, except the unconditional bound in Theorem 2, are established for partition matroids. The *partition* matroid for a given partition $\{V_1, \ldots, V_\ell\}$ of V is the matroid with the ground set V such that a set $X \subseteq V$ is independent if and only if $|X \cap V_i| \leq 1$ for each $i \in \{1, \ldots, \ell\}$ (in a more general setting, it is required that $|V \cap X_i| \leq d_i$ where $d_1, \ldots, d_\ell$ are some constant but we only consider the case $d_1 = \cdots = d_\ell = 1$).

Matroids also could be given by their representations. Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $\mathbb{F}$ be a field. An $r \times n$-matrix A is a *representation of* $\mathcal{M}$ *over* $\mathbb{F}$ if there is a bijective correspondence f between V and the set of columns of A such that for every $X \subseteq V$, $X \in \mathcal{I}$ if and only if the set of columns $f(X)$ consists of linearly independent vectors of $\mathbb{F}^r$. Equivalently, A is a representation of M if M is isomorphic to the *column* matroid of A, that is, the matroid whose ground set is the set of columns of the matrix and the independence of a set of columns is defined as the linear independence. If $\mathcal{M}$ has a such a representation, then $\mathcal{M}$ is *representable* over $\mathbb{F}$ and it is also said M is a *linear* (or $\mathbb{F}$-*linear*) matroid.

Parameterized Complexity. We refer to the books of Cygan et al. [9] and Fomin et al. [16] for an introduction to the area. Here we only briefly mention the notions that are most important to state our results. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$ where Σ^* is a set of strings over a finite alphabet Σ. An input of a parameterized problem is a pair (x, k) where x is a string over Σ and $k \in \mathbb{N}$ is a *parameter.* A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f. The complexity class FPT contains all fixed-parameter tractable parameterized problems. A *kernelization algorithm* or *kernel* for a parameterized problem L is a polynomial-time algorithm that takes as its input an instance (x, k) of L and returns an instance (x', k') of the same problem such that (i) $(x, k) \in L$ if and only if $(x', k') \in L$ and (ii) $|x'| + k' \leq f(k)$ for some computable function $f \colon \mathbb{N} \to \mathbb{N}$. The function f is the *size* of the kernel; a kernel is *polynomial* if f is a polynomial. While every decidable parameterized problem is FPT if and only if the problem admits a kernel, it is unlikely that all FPT problems have polynomial kernels. In particular, the *cross-composition* technique proposed by Bodlaender, Jansen, and Kratsch [5] could be used to prove that a certain parameterized problem does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly.

We conclude the section by defining Rainbow-Stable Set.

Rainbow-Stable Set

Input: A graph G and a partition $\{V_1, \ldots, V_k\}$ of $V(G)$ into k sets, called color classes.

Task: Decide whether G has a stable set S of size k such that $|S \cap V_i| = 1$ for each $i \in \{1, \ldots, k\}$.

As mentioned, Rainbow-Stable Set is a special case of Independent Stable Set for partition matroids where k is the number of subsets in the partition defining the input matroid.

3 Unconditional computational lower bound

Because Independent Stable Set generalizes the classical Stable Set problem, Independent Stable Set is NP-complete [17] and W[1]-complete [12]. However, when the input matroids are given by their independence oracles, we obtain an unconditional computational lower bound. Moreover, we show that the lower bound holds for several graph classes for which the classical Stable Set problem can be solved in polynomial time. For this, we remind that Stable Set is polynomial on claw-free and AT-free graphs by the results of Minty [33] and Broersma et al. [8], respectively.

▶ **Theorem 2.** *There is no algorithm solving Independent Stable Set for frameworks with matroids represented by the independence oracles using $f(k) \cdot n^{o(k)}$ oracle calls for any computable function f. Furthermore, the bound holds for bipartite, chordal, claw-free, and AT-free graphs.*

Proof. First, we show the bound for claw-free and AT-free and then explain how to modify the proof for other graph classes.

Let p and q be positive integers. We define the graph $G_{p,q}$ as the disjoint union of G_i constructed as follows for each $i \in \{1,\dots,p\}$.

- For each $j \in \{1,\dots,q\}$, construct two vertices $a_{i,j}$ and $b_{i,j}$; set $A_i = \{a_{i,1},\dots,a_{i,q}\}$ and $B_i = \{b_{i,1},\dots,b_{i,q}\}$.
- Make A_i and B_i cliques.
- For each $j \in \{1,\dots,q\}$ and for all distinct $h, j \in \{1,\dots,q\}$, make $a_{i,h}$ and $b_{i,j}$ adjacent.

Equivalently, each G_i is obtained by deleting a perfect matching from the complete graph K_{2q}. By the construction, $G_{p,q}$ is both claw-free and AT-free and has $2pq$ vertices. Consider a family of indices $j_1,\dots,j_p \in \{1,\dots,q\}$ and set $W = \bigcup_{i=1}^{p}\{a_{i,j_i}, b_{i,j_i}\}$. We define the matroid $\mathcal{M}_W$ with the ground set $V(G_{p,q})$ as follows for $k = 2p$:

- Each set $X \subseteq V(G_{p,q})$ of size at most $k-1$ is independent and any set of size at least $k+1$ is not independent.
- A set $X \subseteq V(G_{p,q})$ of size k is independent if and only if either $X = W$ or there is $i \in \{1,\dots,p\}$ such that $|A_i \cap X| \geq 2$ or $|B_i \cap X| \geq 2$ or there are distinct $h, j \in \{1,\dots,q\}$ such that $a_{i,h}, b_{i,j} \in X$.

Denote by $\mathcal{I}_W$ the constructed family of independent sets. We will now show that $\mathcal{M}_W$ is indeed a matroid.

▶ **Claim 2.1.** *$\mathcal{M}_W = (V(G_{p,q}), \mathcal{I}_W)$ is a matroid.*

Proof of Claim 2.1. We have to verify that $\mathcal{I}_W$ satisfies the independence axioms (I1)–(I3). The axioms (I1) and (I2) for $\mathcal{I}_W$ follow directly from the definition of $\mathcal{I}_W$. To establish (I3), consider arbitrary $X, Y \in \mathcal{I}_W$ such that $|X| < |Y|$. If $|X| < k-1$ then for any $v \in Y \setminus X$, $Z = X \cup \{v\} \in \mathcal{I}_W$ because $|Z| \leq k-1$.

Suppose $|X| = k-1$ and $|Y| = k$. If there is $i \in \{1,\dots,p\}$ such that $|A_i \cap X| \geq 2$ or $|B_i \cap X| \geq 2$ or there are distinct $h, j \in \{1,\dots,q\}$ such that $a_{i,h}, b_{i,j} \in X$ then for any $v \in Y \setminus X$, the set $Z = X \cup \{v\}$ has the same property and, therefore, $Z \in \mathcal{I}_W$. Assume that this is not the case. By the construction of $G_{p,q}$, we have that for each $i \in \{1,\dots,p\}$, $|X \cap A_i| \leq 1$ and $|X \cap B_i| \leq 1$, and, furthermore, there is $j \in \{1,\dots,q\}$ such that $X \cap (A_i \cup B_i) \subseteq \{a_{i,j}, b_{i,j}\}$. Because $|X| = k-1$, we can assume without loss of generality that there are indices $h_1,\dots,h_p \in \{1,\dots,q\}$ such that $X \cap (A_i \cup B_i) = \{a_{i,h_i}, b_{i,h_i}\}$ for $i \in \{1,\dots,p-1\}$ and $X \cap (A_p \cup B_p) = \{a_{p,h_p}\}$. Recall that $W = \bigcup_{i=1}^{p}\{a_{i,j_i}, b_{i,j_i}\}$ for $j_1,\dots,j_p \in \{1,\dots,q\}$. If there is $v \in Y \setminus X$ such that $v \neq b_{p,j_p}$ then consider $Z = X \cup \{v\}$.

We have that there is $i \in \{1,\dots,p\}$ such that $|A_i \cap Z| \geq 2$ or $|B_i \cap Z| \geq 2$ or there are distinct $h, j \in \{1,\dots,q\}$ such that $a_{i,h}, b_{i,j} \in Z$, that is, $Z \in \mathcal{I}_W$. Now we assume that $Y \setminus X = \{b_{p,j_p}\}$. Then $Y = W$ and we can take $v = b_{p,j_p}$. We obtain that $X \cup \{v\} = Y \in \mathcal{I}_W$. This concludes the proof. ◁

We show the following lower bound for the number of oracle queries for frameworks $(G_{p,q}, \mathcal{M}_W)$.

▶ **Claim 2.2.** *Solving* Independent Stable Set *for instances $(G_{p,q}, \mathcal{M}_W, k)$ with the matroids $\mathcal{M}_W$ defined by the independence oracle for an (unknown) stable set W of $G_{p,q}$ of size k demands at least $q^p - 1$ oracle queries.*

Proof of Claim 2.2. Notice that every stable set of X of size k contains exactly two vertices of each G_i and, moreover, there is $j \in \{1,\dots,q\}$ such that $X \cap V(G_i) = \{a_{i,j}, b_{i,j}\}$. Because the only stable set of this structure that is independent with respect to $\mathcal{M}_W$ is W, the task of Independent Stable Set boils down to finding an unknown stable set W of $G_{p,q}$ of size k using oracle queries. Querying the oracle for sets X of size at most $k-1$ or at least $k+1$ does not provide any information about W. Also, querying the oracle for X of size k with the property that there is $i \in \{1,\dots,p\}$ such that $|A_i \cap X| \geq 2$ or $|B_i \cap X| \geq 2$ or there are distinct $h, j \in \{1,\dots,q\}$ such that $a_{i,h}, b_{i,j} \in X$ also does not give any information because all these are independent. Hence, we can assume that the oracle is queried only for sets X of size k with the property that for each $i \in \{1,\dots,p\}$, there is $j \in \{1,\dots,q\}$ such that $X \cap V(G_i) = \{a_{i,j}, b_{i,j}\}$, that, is the oracle is queried for stable sets of size k. The graph $G_{p,q}$ has q^p such sets. Suppose that the oracle is queried for at most $q^p - 2$ stable sets of size k with the answer no. Then there are two distinct stable sets W and W' of size k such that the oracle was queried neither for W nor W'. The previous queries do not help to distinguish between W and W'. Hence, at least one more query is needed. This proves the claim. ◁

Now, we are ready to prove the claim of the theorem. Suppose that there is an algorithm $\mathcal{A}$ solving Independent Stable Set with at most $f(k) \cdot n^{g(k)}$ oracle calls for computable functions f and g such that $g(k) = o(k)$. Without loss of generality, we assume that f and g are monotone non-decreasing functions. Because $g(k) = o(k)$, there is a positive integer K such that $g(k) < k/2$ for all $k \geq K$. Then for each $k \geq K$, there is a positive integer N_k such that for every $n \geq N_k$, $(f(k) \cdot n^{g(k)} + 1)k^{k/2} < n^{k/2}$.

Consider instances $(G_{p,q}, \mathcal{M}_W, k)$ for even $k \geq K$ where $p = k/2$ and $q \geq N_k/k$. We have that $k = 2p$ and $n = 2pq$. Then $\mathcal{A}$ applied to such instances would use at most $f(k) \cdot n^{g(k)} < \left(\frac{n}{k}\right)^{k/2} - 1 = q^p - 1$ oracle queries contradicting Claim 2.2. This completes the proof for claw and AT-free graphs.

Now we sketch the proof of Theorem 2 for bipartite graphs. For positive integers p and q, we define $H_{p,q}$ as the disjoint union of the graphs H_i for $i \in \{1,\dots,p\}$ constructed as follows.

- For each $j \in \{1,\dots,q\}$, construct three vertices $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$; set $A_i = \{a_{i,1},\dots,a_{i,q}\}$, $B_i = \{b_{i,1},\dots,b_{i,q}\}$, and $C_i = \{c_{i,1},\dots,c_{i,q}\}$.
- For each $j \in \{1,\dots,q\}$, make $a_{i,j}$ and $b_{i,j}$ adjacent to every $c_{i,h}$ for $h \in \{1,\dots,q\}$ such that $h \neq j$.

Notice that $H_{p,q}$ is a bipartite graph with $3pq$ vertices. We define $R = \bigcup_{i=1}^{p}(A_i \cup B_i)$. Consider a family of indices $j_1,\dots,j_p \in \{1,\dots,q\}$ and set $W = \bigcup_{i=1}^{p}\{a_{i,j_i}, b_{i,j_i}\}$. Note that W is a stable set of $H_{p,q}$ of size $2p$. We define the matroid $\mathcal{M}_W$ with the ground set $V(H_{p,q})$ by setting a set $X \subseteq V(H_{p,q})$ to be independent if and only if

- for each $i \in \{1,\dots,p\}$, $|C_i \cap X| \leq 1$ and
- it holds that

- either $X \cap R = W$,
- or $|X \cap R| < 2p$,
- or $|X \cap R| = 2p$ and there is $i \in \{1, \ldots, p\}$ such that $|A_i \cap X| \geq 2$ or $|B_i \cap X| \geq 2$ or there are distinct $h, j \in \{1, \ldots, q\}$ such that $a_{i,h}, b_{i,j} \in X$.

We denote by $\mathcal{I}_W$ the constructed family of independent sets and prove that $\mathcal{M}_W$ is a matroid.

▶ **Claim 2.3.** *$\mathcal{M}_W = (V(H_{p,q}), \mathcal{I}_W)$ is a matroid.*

Proof of Claim 2.3. Let $S = \bigcup_{i=1}^{p} C_i$ and consider $\mathcal{M}_1 = (S, \mathcal{I}_1)$ where $\mathcal{I}_1$ is the set of all $X \subseteq S$ such that $|X \cap C_i| \leq 1$ for $i \in \{1, \ldots, p\}$. Clearly, $\mathcal{M}_1$ is a partition matroid. Now consider $\mathcal{M}_2 = (R, \mathcal{I}_2)$ where $\mathcal{I}_2$ consists of sets $X \subseteq D$ such that either $X = W$, or $|X| < 2p$, or $|X| = 2p$ and there is $i \in \{1, \ldots, p\}$ such that $|A_i \cap X| \geq 2$ or $|B_i \cap X| \geq 2$ or there are distinct $h, j \in \{1, \ldots, q\}$ such that $a_{i,h}, b_{i,j} \in X$. We observe that $\mathcal{M}_2$ is a matroid and the proof of this claim is identical to the proof of Claim 2.1. To complete the proof, it remains to note that $\mathcal{M}_W = \mathcal{M}_1 \cup \mathcal{M}_2$, that is, a set $X \in \mathcal{I}_W$ if and only if $X = Y \cup Z$ for $Y \in \mathcal{I}_1$ and $Z \in \mathcal{I}_2$. This implies that $\mathcal{M}_W$ is a matroid [34]. ◁

We consider instances $(H_{p,q}, \mathcal{M}_W, k)$ of Independent Stable Set with the matroid $\mathcal{M}_W$ defined by the independence oracle for an (unknown) W and $k = 3p$. By the definition of $\mathcal{M}_W$, any stable set X of $H_{p,q}$ of size k that is independent with respect to $\mathcal{M}_W$ has the property that $|X \cap C_i| = 1$ for every $i \in \{1, \ldots, p\}$. The construction of $H_{p,q}$ implies that if $c_{i,j} \in X \cap C_i$ then $X \cap (A_i \cup B_i) \subseteq \{a_{i,j}, b_{i,j}\}$. Because $|X| = k = 3p$, we obtain that $X \cap (A_i \cup B_i) = \{a_{i,j}, b_{i,j}\}$. Then by the construction of $\mathcal{M}_W$, we obtain that $X = \bigcup_{i=1}^{p} \{a_{i,j_i}, b_{i,j_i}, c_{i,j_i}\}$ where $W = \bigcup_{i=1}^{p} \{a_{i,j_i}, b_{i,j_i}\}$, that is, X is uniquely defined by W. In the same way as in Claim 2.2 we obtain that solving Independent Stable Set for instances $(H_{p,q}, \mathcal{M}_W, k)$ with the matroids $\mathcal{M}_W$ defined by the independence oracle for an (unknown) W demands at least $q^p - 1$ oracle queries. Similarly to the case of claw and AT-free graphs, we conclude that the existence of an algorithm for Independent Stable Set using $f(k) \cdot n^{o(k)}$ oracle calls would lead to a contradiction. This finishes the proof for bipartite graphs.

For chordal graphs, we modify the construction of $H_{p,q}$ by making each C_i a clique. Then $H_{p,q}$ becomes chordal but we can apply the same arguments to show that solving Independent Stable Set for instances $(H_{p,q}, \mathcal{M}_W, k)$ with the matroids $\mathcal{M}_W$ defined by the independence oracle for an (unknown) W demands at least $q^p - 1$ oracle queries. This completes the proof. ◀

4 Independent Stable Set on sparse frameworks

In this section, we consider Independent Stable Set for graphs of bounded maximum degree and graphs of bounded degeneracy. First, we observe that the problem is FPT when parameterized by the solution size and the degeneracy by giving a recursive branching algorithm.

▶ **Theorem 3.** *Independent Stable Set can be solved in $\mathcal{O}((d+1)^k \cdot n)$ time on frameworks with d-degenerate input graphs.*

Proof. The algorithm is based on the following observation. Let $(G, \mathcal{M})$ be a framework such that for every $v \in V(G)$, $\{v\} \in \mathcal{I}$. Then there is a stable set X of G that is independent with respect to $\mathcal{M}$ whose size is maximum such that $X \cap N_G[v] \neq \emptyset$. To see this, let X be a stable set that is also independent in $\mathcal{M}$ and such that $X \cap N_G[v] = \emptyset$. Because $\{v\}$ and

X are independent, there is $Y \subseteq X$ of size $|X|-1$ such that $Z = Y \cup \{v\}$ is independent. Because $N_G(v) \cap Z = \emptyset$ and Y is a stable set, Z is a stable set. Thus, set Z of size $|X|$ is stable in G and is independent in $\mathcal{M}$. This proves the observation.

Consider an instance $(G, \mathcal{M}, k)$ of Independent Stable Set. Because G is a d-degenerate graph, there is an elimination ordering $v_1, \ldots, v_n$ of the vertices of G, that is, $d_{G_i}(v_i) \leq d$ for every $i \in \{1, \ldots, n\}$ where $G_i = G[\{v_i, \ldots, v_n\}]$. Recall that such an ordering can be computed in linear time [32].

If there is $v \in V(G)$ such that $\{v\} \notin \mathcal{I}$, then we delete v from the framework as such vertices are trivially irrelevant. From now on, we assume that $\{v\} \in \mathcal{I}$ for any $v \in V(G)$. If $k = 0$, then $\emptyset$ is a solution, and we return yes and stop. If $k \geq 1$ but $V(G) = \emptyset$, then we conclude that the answer is no and stop. We can assume that $V(G) \neq \emptyset$ and $k \geq 1$.

Let u be the first vertex in the elimination ordering. Clearly, $d_G(u) \leq d$. We branch on at most $d+1$ instances $(G - v, \mathcal{M}/v, k-1)$ for $v \in N_G[u]$, where $\mathcal{M}/v$ is the contraction of $\mathcal{M}$ by $\{v\}$. By our observation, $(G, \mathcal{M}, k)$ is a yes-instance of Independent Stable Set if and only if at least one of the instances $(G - v, \mathcal{M}/v, k-1)$ is a yes-instance.

In each step, we have at most $d+1$ branches and the depth of the search tree is at most k. Note that we do not need to recompute the elimination ordering when a vertex is deleted; instead, we just delete the vertex from the already constructed ordering. This means we can use the ordering constructed for the original input instance. Thus, the total running time is $\mathcal{O}((d+1)^k \cdot n)$. This concludes the proof. ◀

For bounded degree graphs, we prove that Independent Stable Set has a polynomial kernel when parameterized by k and the maximum degree.

▶ **Theorem 4.** *Independent Stable Set admits a polynomial kernel on frameworks with graphs of maximum degree at most Δ such that the output instance contains a graph with at most $k^2\Delta$ vertices.*

Proof. Let $(G, \mathcal{M}, k)$ be an instance of Independent Stable Set with $\Delta(G) \leq \Delta$. Recall that by our assumption, $r(\mathcal{M}) \leq k$. If $r(\mathcal{M}) < k$ then $(G, \mathcal{M}, k)$ is a no-instance. In this case, our kernelization algorithm returns a trivial no-instance of constant size and stops. Now we can assume that $r(\mathcal{M}) = k$. If $k = 0$ then we return a trivial yes-instance as $\emptyset$ is a solution. If $\Delta = 0$, then any base of $\mathcal{M}$ is a solution, and we return a trivial yes-instance. Now we can assume that $k \geq 1$ and $\Delta \geq 1$.

We set $W_0 = \emptyset$. Then for $i = 1, \ldots, \ell$ where $\ell = k\Delta$, we greedily select a maximum-size independent set $W_i \subseteq V(G) \setminus \left(\bigcup_{j=0}^{i-1} W_j\right)$. Our kernelization algorithms returns the instance $(G', \mathcal{M}', k)$ where $G' = G[\bigcup_{i=1}^{\ell} W_i]$ and $\mathcal{M}'$ is the restriction of $\mathcal{M}$ to $V(G')$. It is straightforward to see that $|V(G')| \leq k^2\Delta$ as $|W_i| \leq r(\mathcal{M}) = k$ and the new instance can be constructed in polynomial time. We claim that $(G, \mathcal{M}, k)$ is a yes-instance of Independent Stable Set if and only if $(G', \mathcal{M}', k)$ is a yes-instance.

Because G' is an induced subgraph of G, any stable set of G' is a stable set of G. This immediately implies that if $(G', \mathcal{M}', k)$ is a yes-instance then any solution to $(G', \mathcal{M}', k)$ is a solution to $(G, \mathcal{M}, k)$ and, thus, $(G, \mathcal{M}, k)$ is a yes-instance. Suppose that $(G, \mathcal{M}, k)$ is a yes-instance. It means that G contains a stable set of size k independent in $\mathcal{M}$. We show that there is a stable set $X \subseteq V(G')$ of G of size k that is independent with respect to $\mathcal{M}$.

To show this, let X be a stable set of size k that is independent in $\mathcal{M}$ with the maximum number of vertices in $V(G')$. For the sake of contradiction, assume that there is $u \in X \setminus V(G')$. We define $Y = X \setminus \{u\}$. Consider the set W_i for some $i \in \{1, \ldots, \ell\}$. By the construction of the set, we have that $u \in \mathsf{cl}(W_i)$. Then it holds that $r(Y \cup W_i) \geq r(X)$. This implies that there is $w_i \in W_i$ such that $r(Y \cup \{w_i\}) = r(X) = k$. Because this property holds for arbitrary $i \in \{1, \dots, \ell\}$, we obtains that there are $\ell = k\Delta$ vertices $w_1, \dots, w_\ell \in V(G')$ such that for any $i \in \{1, \dots, \ell\}$, $r(Y \cup \{w_i\}) = k$. Notice that $w_i \notin Y$ for $i \in \{1, \dots, \ell\}$ and $|N_G(Y)| \leq (k-1)\Delta$. Therefore, there is $i \in \{1, \dots, \ell\}$ such that w_i is not adjacent to any vertex of Y. Then $Z = Y \cup \{w_i\}$ is a stable set of G. However, $|Z \cap V(G')| > |X \cup V(G')|$ contradicting the choice of X. This proves that there is a stable set $X \subseteq V(G')$ of G of size k that is independent in $\mathcal{M}$. Then X is a solution to $(G', \mathcal{M}', k)$, that is, $(G', \mathcal{M}', k)$ is a yes-instance. This concludes the proof. ◀

Theorem 4 is handy for kernelization with parameter k when the degeneracy of the graph in a framework is a constant.

▶ **Theorem 5.** *For every integer $d \geq 0$, Independent Stable Set admits a polynomial kernel with running time $n^{\mathcal{O}(d)}$ on frameworks with graphs of degeneracy at most d such that the output instance contains a graph with $dk^{\mathcal{O}(d)}$ vertices.*

Proof. Let $(G, \mathcal{M}, k)$ be an instance of Independent Stable Set where the degeneracy of G is at most d. We assume without loss of generality that $r(\mathcal{M}) = k$. Otherwise, if $r(\mathcal{M}) < k$, then $(G, \mathcal{M}, k)$ is a no-instance, and we can return a trivial no-instance of constant size and stops. If $d = 0$, then G is an edgeless graph, and any set of vertices forming a base of $\mathcal{M}$ is a stable set of size k that is independent with respect to $\mathcal{M}$, that is, $(G, \mathcal{M}, k)$ is a yes-instance. Then we return a trivial yes-instance and stop. From now on, we assume that $d \geq 1$. Also, we assume that $k \geq 2$. Otherwise, if $k = 0$, the empty set is a trivial solution. If $k = 1$ then because $r(\mathcal{M}) = k \geq 1$, there is a vertex v such that $\{v\} \in \mathcal{I}(\mathcal{M})$ and $\{v\}$ is an independent set of size k. In both cases, we return a trivial yes-instance and stop.

Since G is a d-degenerate graph, it admits an elimination ordering $v_1, \dots, v_n$ of the vertices of G, that is, $d_{G_i}(v_i) \leq d$ for every $i \in \{1, \dots, n\}$ where $G_i = G[\{v_i, \dots, v_n\}]$. Recall that such an ordering can be computed in linear time [32]. For a set of vertices $X \subseteq V(G)$, we use $F(X)$ to denote the set of common neighbors of the vertices of X that occur before the vertices of X in the elimination ordering. Note that because G is a d-degenerate graph, $F(X) = \emptyset$ if $|X| > d$. For an integer $i \geq 1$, $f_i(G) = \max\{|F(X)| \colon X \subseteq V(G) \text{ and } |X| = i\}$. Clearly, $f_i(G) = 0$ if $i > d$.

For each $h = d, \dots, 1$, we apply the following reduction rule starting with $h = d$. Whenever the rule deletes some vertices, we do not recompute the elimination ordering; instead, we use the induced ordering obtained from the original one by vertex deletions.

▶ **Reduction Rule 5.1.** *Set $d_h = d + f_{h+1}(G)$. For each $X \subseteq V(G)$ such that $|X| = h$, do the following:*

(i) *set $W_0 = \emptyset$,*

(ii) *for $i = 1, \dots, \ell$ where $\ell = kd_h$, greedily select a maximum-size independent set $W_i \subseteq F(X) \setminus \left(\bigcup_{j=0}^{i-1} W_j\right)$,*

(iii) *delete the vertices of $D = F(X) \setminus \left(\bigcup_{i=1}^{\ell} W_i\right)$ and restrict $\mathcal{M}$ to $V(G) \setminus D$.*

It is easy to see that the rule can be applied in $n^{\mathcal{O}(d)}$ time. We show that the rule is *safe*, that is, it returns an equivalent instance of the problem.

▶ **Claim 5.1.** *Reduction Rule 5.1 is safe.*

Proof of Claim 5.1. Let $X \subseteq V(G)$ be of size h. Denote by G' the graph obtained from G by applying steps (i)–(iii) for X and let $\mathcal{M}$ be the restriction of $\mathcal{M}$ to $V(G) \setminus D$. We prove that $(G, \mathcal{M}, k)$ is a yes-instance of Independent Stable Set if and only if $(G', \mathcal{M}', k)$ is a yes-instance. Clearly, this is sufficient for the proof of the claim. Since G' is an induced subgraph of G, any solution to $(G', \mathcal{M}', k)$ is a solution to $(G, \mathcal{M}, k)$. Thus, if $(G', \mathcal{M}', k)$ is a yes-instance then the same holds for $(G, \mathcal{M}, k)$. Hence, it remains to show that if $(G, \mathcal{M}, k)$ is a yes-instance then $(G', \mathcal{M}', k)$ is a yes-instance as well.

We use the following axillary observation: for every $v \in V(G) \setminus X$, $|N_G(v) \cap F(X)| \leq d_h$. To see this, consider $v \in V(G) \setminus X$, and denote by L and R the sets of vertices of $F(X)$ that are prior and after v, respectively, in the elimination ordering. By the definition of an elimination ordering, $|N_G(v) \cap R| \leq d$. For $N_G(v) \cap L$, we have that $N_G(v) \cap L \subseteq F(X \cup \{v\})$. Then $|N_G(v) \cap L| \leq |F(X \cup \{v\})| \leq f_{h+1}$. We conclude that $|N_G(v) \cap F(X)| = |N_G(v) \cap L| + |N_G(v) \cap R| \leq d + f_{h+1} = d_h$. This proves the observation.

Suppose that G has a stable set Y of size k that is independent with respect to $\mathcal{M}$. Among all these sets, we select Y such that $Y \cap D$ has the minimum size. We claim that $Y \cap D = \emptyset$. The proof is by contradiction and is similar to the proof of Theorem 4. Assume that there is $u \in Y \cap D$ and let $Z = Y \setminus \{u\}$. For each $i \in \{1, \ldots, \ell\}$, $u \in \mathsf{cl}(W_i)$ by the construction of W_i. Thus, $r(Z \cup W_i) \geq r(Y)$ and for each $i \in \{1, \ldots, \ell\}$, there is $w_i \in W_i$ such that $r(Z \cup \{w_i\}) = r(Y) = k$. Therefore, there are ℓ vertices $w_1, \ldots, w_\ell \in F(X) \setminus D$ such that for any $i \in \{1, \ldots, \ell\}$, $r(Z \cup \{w_i\}) = k$. Notice that $w_i \notin Z$ for all $i \in \{1, \ldots, \ell\}$ and $Y \cap X = \emptyset$ because u is adjacent to every vertex of X. By the above observation, we have that $|N_G(Z) \cap F(X)| \leq (k-1)d_h$. Since $\ell = kd_h > (k-1)d_h$, there is $i \in \{1, \ldots, \ell\}$ such that $w_i \notin N_G(Z)$. Then $Y' = Z \cup \{w_i\}$ is a stable set of G. Because Y' is independent with respect to $\mathcal{M}$ and $u \notin D$, this leads to a contradiction with the choice of Y. We conclude that there is a stable set Y of G of size k that is independent with respect to $\mathcal{M}$ such that $Y \cap D = \emptyset$. This means that Y is a solution to $(G', \mathcal{M}', k)$, that is, $(G', \mathcal{M}', k)$ is a yes-instance of Independent Stable Set. This concludes the proof. ◁

Denote by $(G', \mathcal{M}', k)$ the instance of Independent Stable Set obtained after applying Reduction Rule 5.1. We prove that the maximum degree of G' is bounded.

▶ **Claim 5.2.** $\Delta(G') \leq dk^{2d+1}$.

Proof of Claim 5.2. For $i \in \{1, \ldots, d\}$, denote by G_i the graph obtained from G by applying Reduction Rule 5.1 for $h = d, \ldots, i$. Note that $G' = G_1$. Because $r(\mathcal{M}) = k$, for each set W_j selected in step (ii) of Reduction Rule 5.1, $|W_j| \leq k$. Therefore, $|\bigcup_{j=1}^{\ell} W_j| \leq k\ell = k^2 d_h$. Notice that for $h = d$, $f_{h+1}(G) = 0$ and, therefore, $d_h = d$. This implies that $f_d(G_d) \leq k^2 d$. For $i < d$, we have that $f_i(G_i) \leq k^2 d_i = k^2(d + f_{i+1}(G_{i+1}))$. Therefore, $f_i(G_i) \leq d \sum_{j=i}^{d} k^{2(j-i+1)}$ and, as $k \geq 2$,

$$f_1(G') \leq f_1(G_1) \leq d \sum_{j=1}^{d} k^{2j} = d \sum_{j=0}^{d} k^{2j} - d = d \frac{k^{2(d+1)} - 1}{k^2 - 1} - d \leq dk^{2d+1} - d.$$

Therefore, each vertex v of G' has at most $dk^{2d+1} - d$ neighbors in G' that are prior v in the elimination ordering. Because v has at most d neighbors that are after v in the ordering, $d_{G'}(v) \leq dk^{2d+1}$. This concludes the proof. ◁

Because the maximum degree of G' is bounded, we can apply Theorem 4. Applying the kernelization algorithm from this theorem to $(G', \mathcal{M}', k)$, we obtain a kernel with at most dk^{2d+3} vertices. This concludes the proof of the theorem. ◀

In Theorem 5, we proved that Independent Stable Set admits a polynomial kernel on d-degenerate graphs when d is a fixed constant. We complement this result by showing that it is unlikely that the problem has a polynomial kernel when parameterized by both k and d.

▶ **Theorem 6.** *Independent Stable Set on frameworks with d-degenerate graphs and partition matroids does not admit a polynomial kernel when parameterized by $k + d$ unless* $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$.

Proof. We use the fact that Rainbow-Stable Set is a special case of Independent Stable Set and show that Rainbow-Stable Set does not admit a polynomial kernel when parameterized by $k + d$ unless $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$ where k is the number of color classes.

We use cross-composition from Rainbow-Stable Set. We say that two instances $(G, \{V_1, \ldots, V_k\})$ and $(G', \{V'_1, \ldots, V'_{k'}\})$ are *equivalent* if $|V(G)| = |V(G')|$ and $k = k'$. Consider t equivalent instances $(G_i, \{V^i_1, \ldots, V^i_k\})$ of Rainbow-Stable Set for $i \in \{1, \ldots, t\}$ where each graph has n vertices. We assume that $t = 2^p$ for some $p \geq 1$. Otherwise, we add $2^{\lceil \log t \rceil} - t$ copies of $(G_1, \{V^1_1, \ldots, V^1_k\})$ to achieve the property for $p = \lceil \log t \rceil$; note that by this operation, we may add at most t instances. Then we construct the instance $(G, \{V_1, \ldots, V_{k+p}\})$ of Rainbow-Stable Set as follows.

- Construct the disjoint union of copies of $G_1, \ldots, G_t$.
- For each $i \in \{1, \ldots, p\}$,
  - construct two adjacent vertices u_i and v_i,
  - for each $j \in \{1, \ldots, t\}$, consider the binary encoding of $j - 1$ as a string s with p symbols and make all the vertices of G_j adjacent to u_i if $s[i] = 0$ and make them adjacent to v_i, otherwise, for $i \in \{1, \ldots, p\}$.
- Define $k + p$ color classes $V_i = \bigcup_{j=1}^{t} V^j_i$ for $i \in \{1, \ldots, k\}$ and $V_{k+i} = \{u_i, v_i\}$ for $i \in \{1, \ldots, p\}$.

It is straightforward to see that the instance $(G, \{V_1, \ldots, V_{k+p}\})$ of Rainbow-Stable Set can be constructed in polynomial time. We claim that $(G, \{V_1, \ldots, V_{k+p}\})$ is a yes-instance of Rainbow-Stable Set if and only if there is $j \in \{1, \ldots, t\}$ such that $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance of Rainbow-Stable Set.

Suppose that $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance for some $j \in \{1, \ldots, t\}$. Then there is a stable set $X \subseteq V(G_j)$ of size k such that $|X \cap V^j_i| = 1$ for $i \in \{1, \ldots, k\}$. Let s be the string with p symbols that is the binary encoding of $j - 1$. Consider the set $Y \subseteq \bigcup_{i=1}^{p} \{u_i, v_i\}$ such that for each $i \in \{1, \ldots, p\}$, Y contains either u_i or v_i, and u_i is in Y whenever $s[i] = 1$. Observe that $Z = X \cup Y$ is a stable set of G and it holds that $|Z \cap V_h| = 1$ for each $h \in \{1, \ldots, p+k\}$. This means that $(G, \{V_1, \ldots, V_{k+p}\})$ is a yes-instance of Rainbow-Stable Set.

For the opposite direction, assume that $(G, \{V_1, \ldots, V_{k+p}\})$ is a yes-instance of Rainbow-Stable Set. Then there is a stable set Z of G of size $k' = k + p$ such that $|Z \cap V_h| = 1$ for each $h \in \{1, \ldots, p + k\}$. Let $Y = Z \cap \left(\bigcup_{i=1}^{p} \{u_i, v_i\}\right)$ and $X = Z \setminus Y$. By the construction of color classes and because Y is a stable set, for each $i \in \{1, \ldots, p\}$, Y contains either u_i or v_i. Also, we have that $X \subseteq \bigcup_{j=1}^{t} V(G_j)$. Consider the binary string s of length p such that $s[i] = 1$ if $u_i \in Y$ and $s[i] = 0$, otherwise, for all $i \in \{1, \ldots, p\}$. Notice that the vertices of G_j such that s is the binary encoding of $j - 1$ are not adjacent to the vertices of Y and for every $j' \in \{1, \ldots, t\}$ distinct from j, all the vertices of $G_{j'}$ are adjacent to at least one vertex of Y. This implies that $X \subseteq V(G_j)$. Therefore, X is a stable set of G_j of size k and $|X \cap V^j_i| = 1$ for $i \in \{1, \ldots, k\}$, that is, $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance of Rainbow-Stable Set.

Notice that each vertex $v \in V(G_j)$ for $j \in \{1, \ldots, t\}$ is adjacent in G to at most $n - 1$ vertices of G_j and p vertices of $\bigcup_{i=1}^{p} \{u_i, v_i\}$. Therefore, the degeneracy of G is at most $n + \log t$. Also, we have the number of color classes $k' = k + p \leq n + \log t$. Then because Rainbow-Stable Set is NP-complete and $(G, \{V_1, \ldots, V_{k+p}\})$ is a yes-instance of Rainbow-Stable Set if and only if there is $j \in \{1, \ldots, t\}$ such that $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance of

Rainbow-Stable Set, the result of Bodlaender, Jansen, and Kratsch [5] implies that Rainbow-Stable Set does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly when parameterized by the number of color classes k and the degeneracy of the input graph. This concludes the proof. ◀

5 Independent Stable Set on chordal graphs

For chordal graphs, we show that Independent Stable Set is FPT in the case of linear matroids when parameterized by k by demonstrating a dynamic programming algorithm over tree decompositions exploiting representative sets [28, 31, 15, 27].

Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $\mathcal{S}$ be a family of subsets of V. For a positive integer q, a subfamily $\widehat{\mathcal{S}}$ is *q-representative for* $\mathcal{S}$ if the following holds: for every set $Y \subseteq V$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$ then there is $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$. We write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ to denote that $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for S. We use the results of of Fomin et al. [15] to compute representative families for linear matroids. A family of sets $\mathcal{S}$ is said to be a *p-family* for an integer $p \geq 0$ if $|S| = p$ for every $S \in \mathcal{S}$, and we use $\|A\|$ to denote the bit-length of the encoding of a matrix A.

▶ **Proposition 7** ([15, Theorem 3.8]). *Let $M = (V, \mathcal{I})$ be a linear matroid and let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a p-family of independent sets. Then there exists $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size at most $\binom{p+q}{p}$. Furthermore, given a representation A of M over a field $\mathbb{F}$, there is a randomized Monte Carlo algorithm computing $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size at most $\binom{p+q}{p}$ in $\mathcal{O}(\binom{p+q}{p} tp^\omega + t\binom{p+q}{q}^{\omega-1}) + \|A\|^{\mathcal{O}(1)}$ operations over $\mathbb{F}$, where ω is the exponent of matrix multiplication.*[1]

The following theorem is proved by the bottom-up dynamic programming over a nice tree decomposition where representative sets are used to store partial solutions. Due to space constraints, the proof is omitted in this extended abstract and can be found in the full version [14].

▶ **Theorem 8.** *Independent Stable Set can be solved in $2^{\mathcal{O}(k)} \cdot \|A\|^{\mathcal{O}(1)}$ time by a one-sided error Monte Carlo algorithm with false negatives on frameworks with chordal graphs and linear matroids given by their representations A.*

The algorithm in Theorem 8 is randomized because it uses the algorithm from Proposition 7 to compute representative sets. For some linear matroids, the algorithm can be derandomized using the deterministic construction of representative sets given by Lokshtanov et al. [27]. In particular, this can be done for linear matroids over any finite field and the field of rational numbers.

We complement Theorem 8 by proving that it is unlikely that Independent Stable Set admits a polynomial kernel when parameterized by k in the case of chordal graphs.

▶ **Theorem 9.** *Independent Stable Set on frameworks with chordal graphs and partition matroids does not admit a polynomial kernel when parameterized by k unless* NP $\subseteq$ coNP /poly.

Proof. In the same way as in the proof of Theorem 6, we prove that Rainbow-Stable Set does not admit a polynomial kernel when parameterized by k on chordal graphs unless NP $\subseteq$ coNP /poly where k is the number of color classes.

[1] The currently best value is $\omega \approx 2.3728596$ [4].

We construct a cross-composition from Rainbow-Stable Set. Again, we say that two instances $(G, \{V_1, \ldots, V_k\})$ and $(G', \{V'_1, \ldots, V'_{k'}\})$ are *equivalent* if $|V(G)| = |V(G')|$ and $k = k'$. Consider t equivalent instances $(G_i, \{V^i_1, \ldots, V^i_k\})$ of Rainbow-Stable Set for $i \in \{1, \ldots, t\}$ where each graph is chordal and has n vertices. Then we construct the instance $(G, \{V_0, V_1, \ldots, V_k\})$ of Rainbow-Stable Set as follows.

- Construct the disjoint union of copies of $G_1, \ldots, G_t$.
- Construct a clique K with t vertices $v_1, \ldots, v_t$.
- For each $j \in \{1, \ldots, t\}$, make v_j adjacent to all the vertices of every G_i for $i \in \{1, \ldots, t\}$ that is distinct from j.
- Define $k + 1$ color classes $V_0 = K$ and $V_i = \bigcup_{j=1}^{t} V^j_i$ for $i \in \{1, \ldots, k\}$.

It is straightforward to see that G is chordal and the instance $(G, \{V_0, V_1, \ldots, V_k\})$ of Rainbow-Stable Set can be constructed in polynomial time. We claim that $(G, \{V_0, V_1, \ldots, V_k\})$ is a yes-instance of Rainbow-Stable Set if and only if there is $j \in \{1, \ldots, t\}$ such that $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance of Rainbow-Stable Set.

Suppose that $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance for some $j \in \{1, \ldots, t\}$. Then there is a stable set $X \subseteq V(G_j)$ of size k such that $|X \cap V^j_i| = 1$ for $i \in \{1, \ldots, k\}$. By the construction of G, the vertex $v_j \in K$ is not adjacent to any vertex of G_j. Thus, $Y = X \cup \{v_j\}$ is stable set of G such that $|Y \cap V_i| = 1$ for each $i \in \{0, \ldots, k\}$. Therefore, $(G, \{V_0, V_1, \ldots, V_k\})$ is a yes-instance of Rainbow-Stable Set.

For the opposite direction, assume that $(G, \{V_0, V_1, \ldots, V_k\})$ is a yes-instance of Rainbow-Stable Set. Then there is a stable set Y of G of size $k + 1$ such that $|Y \cap V_i| = 1$ for each $i \in \{0, \ldots, k\}$. In particular, $|Y \cap V_0| = 1$. Then there is $j \in \{1, \ldots, t\}$ such that $v_j \in Y$. By the construction of G, we have that $X = Y \setminus \{v_j\} \subseteq V(G_j)$. Then $|X \cap V^j_i| = 1$ for each $i \in \{1, \ldots, k\}$, that is, $(G_j, \{V^j_1, \ldots, V^j_k\})$ is a yes-instance of Rainbow-Stable Set.

Le and Pfender in [26] proved that Rainbow Matching remains NP-complete on paths. This implies that Rainbow-Stable Set is also NP-complete on paths, and hence on chordal graphs. Because the number of color classes is $k + 1 \leq n + 1$ and Rainbow-Stable Set is NP-complete on chordal graphs, we can apply the result of Bodlaender, Jansen, and Kratsch [5]. This concludes the proof. ◀

6 Conclusion

In this paper, we investigated the parameterized complexity of the Independent Stable Set problem for various classes of graphs where the classical Stable Set problem is tractable. We derived kernelization results and FPT algorithms, complemented by complexity lower bounds. We believe exploring Independent Stable Set on other natural graph classes with similar properties would be interesting. For instance, Stable Set is solvable in polynomial time on claw-free graphs [33] and AT-free graphs [8]. While our unconditional lower bound from Theorem 2 applies to these classes, it does not rule out the possibility of FPT algorithms for frameworks with *linear* matroids. A similar question arises regarding graphs with a polynomial number of minimal separators [6, 7].

References

1 Ron Aharoni, Eli Berger, Maria Chudnovsky, David M. Howard, and Paul D. Seymour. Large rainbow matchings in general graphs. *Eur. J. Comb.*, 79:222–227, 2019. doi:10.1016/J.EJC.2019.01.012.

2 Ron Aharoni, Eli Berger, and Ran Ziv. Independent systems of representatives in weighted graphs. *Combinatorica*, 27(3):253–267, 2007.

3 Ron Aharoni, Joseph Briggs, Jinha Kim, and Minki Kim. Rainbow independent sets in certain classes of graphs. *Journal of Graph Theory*, 104(3):557–584, 2023.

4 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discret. Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

6 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

7 Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002. doi:10.1016/S0304-3975(01)00007-X.

8 Hajo Broersma, Ton Kloks, Dieter Kratsch, and Haiko Müller. Independent sets in asteroidal triple-free graphs. *SIAM J. Discret. Math.*, 12(2):276–287, 1999. doi:10.1137/S0895480197326346.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Trans. Algorithms*, 13(3):43:1–43:22, 2017. doi:10.1145/3108239.

11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Arthur A. Drisko. Transversals in row-latin rectangles. *Journal of Combinatorial Theory, Series A*, 84(2):181–195, 1998. doi:10.1006/jcta.1998.2894.

14 Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, and Saket Saurabh. Stability in graphs with matroid constraints, 2024. arXiv:2404.03979.

15 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

16 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

17 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

18 Alessandra Graf, David G. Harris, and Penny Haxell. Algorithms for weighted independent transversals and strong colouring. *ACM Trans. Algorithms*, 18(1):1:1–1:16, 2022. doi:10.1145/3474057.

19 Alessandra Graf and Penny Haxell. Finding independent transversals efficiently. *Comb. Probab. Comput.*, 29(5):780–806, 2020. doi:10.1017/S0963548320000127.

20 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

21 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms and kernels for rainbow matching. *Algorithmica*, 81(4):1684–1698, 2019. doi:10.1007/S00453-018-0497-3.

22 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Quadratic vertex kernel for rainbow matching. *Algorithmica*, 82(4):881–897, 2020. doi:10.1007/S00453-019-00618-0.

23 Penny Haxell. On forming committees. *The American Mathematical Monthly*, 118(9):777–788, 2011.

24 Alon Itai, Michael Rodeh, and Steven L. Tanimoto. Some matching problems for bipartite graphs. *J. ACM*, 25(4):517–525, 1978. doi:10.1145/322092.322093.

25 Jinha Kim, Minki Kim, and O-joung Kwon. Rainbow independent sets on dense graph classes. *Discrete Applied Mathematics*, 312:45–51, 2022.

26 Van Bang Le and Florian Pfender. Complexity results for rainbow matchings. *Theor. Comput. Sci.*, 524:27–33, 2014. doi:10.1016/J.TCS.2013.12.013.

27 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018. doi:10.1145/3170444.

28 L. Lovász. Flats in matroids and geometric graphs. In *Combinatorial surveys (Proc. Sixth British Combinatorial Conf., Royal Holloway Coll., Egham, 1977)*, pages 45–86, 1977.

29 László Lovász. *Graphs and geometry*, volume 65 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2019. doi:10.1090/coll/065.

30 Pasin Manurangsi, Erel Segal-Halevi, and Warut Suksompong. On maximum bipartite matching with separation. *Inf. Process. Lett.*, 182:106388, 2023. doi:10.1016/J.IPL.2023.106388.

31 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

32 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.

33 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

34 James Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford University Press, Oxford, second edition, 2011. doi:10.1093/acprof:oso/9780198566946.001.0001.

A Logarithmic Integrality Gap for Generalizations of Quasi-Bipartite Instances of Directed Steiner Tree

Zachary Friggstad
University of Alberta, Canada

Hao Sun
University of Alberta, Canada

Abstract

In the classic Directed Steiner Tree problem (**DST**), we are given an edge-weighted directed graph $G = (V, E)$ with n nodes, a specified root node $r \in V$, and k terminals $X \subseteq V - \{r\}$. The goal is to find the cheapest $F \subseteq E$ such that r can reach any terminal using only edges in F.

Designing approximation algorithms for **DST** is quite challenging, to date the best approximation guarantee of a polynomial-time algorithm for **DST** is $O(k^\epsilon)$ for any constant $\epsilon > 0$ [Charikar et al., 1999]. For network design problems like **DST**, one often relies on natural cut-based linear programming (LP) relaxations to design approximation algorithms. In general, the integrality gap of such an LP for **DST** is known to have a polynomial integrality gap lower bound [Zosin and Khuller, 2002; Li and Laekhanukit, 2021]. So particular interest has been invested in special cases or in strengthenings of this LP.

In this work, we show the integrality gap is only $O(\log k)$ for instances of **DST** where no Steiner node has both an edge from another Steiner node and an edge to another Steiner node, i.e. the longest path using only Steiner nodes has length at most 1. This generalizes the well-studied case of quasi-bipartite **DST** where no edge has both endpoints being Steiner nodes. Our result is also optimal in the sense that the integrality gap can be as bad as poly(n) even if the longest path with only Steiner nodes has length 2.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Steiner Tree, Approximation Algorithms, Linear Programming

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.23

Funding *Zachary Friggstad*: Research supported by an NSERC Discovery Grant and Accelerator Supplement.

1 Introduction

The Directed Steiner Tree problem (**DST**) is one of the most foundational models in combinatorial optimization and network design. Given a directed graph $G = (V, E)$ with n nodes, a specified root node $r \in V$, and k terminals $X \subseteq V - \{r\}$, the goal is to buy the cheapest $F \subseteq E$ such that r can reach any terminal using only edges in F. Throughout, we say nodes in $V - (X \cup \{r\})$ are **Steiner nodes**.

Despite its central position in discrete optimization, there is a large gap in our understanding concerning its approximability. Namely, the best polynomial-time approximation is currently an $O(k^\epsilon)$-approximation for any constant $\epsilon > 0$ by Charikar et al. [4]. Grandoni, Laekhanukit, and Li show **DST** cannot be approximated within $o(\log^2 n/ \log\log n)$ unless $\mathbf{NP} \subseteq \cap_{0<\delta}\mathbf{ZTIME}(2^{n^\delta})$ [12], improving on a slightly weaker lower bound than the one inherited from Group Steiner Tree [13]. These bounds differ by an order of magnitude. On the other hand, Grandoni, Laekhanukit, and Li do obtain matching $O(\log^2 k/ \log\log k)$-approximation in quasi-polynomial time. Still, a polylogarithmic approximation in polynomial time remains elusive.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Linear Programming Relaxations and Previous Work

In this paper, we consider the following natural linear programming (LP) relaxation for **DST** in which we have a variable x_e for each edge $e \in E$ modelling whether we include edge e in the solution or not.

$$\begin{array}{rrcll} \textbf{minimize}: & \sum_{e \in E} c_e \cdot x_e & & & \\ \textbf{subject to}: & x(\delta^{in}(S)) & \geq & 1 & \forall\, S \subseteq V - \{r\}, S \cap X \neq \emptyset \\ & x(\delta^{in}(v)) & \leq & 1 & \forall\, v \in V \\ & x & \geq & 0 & \end{array} \tag{\textbf{DST-LP}}$$

Here, for any $S \subseteq V$ we let $\delta^{in}(S) = \{(u,v) \in E : u \notin S, v \in S\}$ and we use the shorthand $\delta^{in}(v) := \delta^{in}(\{v\})$ for any $v \in V$. The cut constraints capture the fact that every cut separating the root from some terminal must be crossed by at least one edge in a feasible **DST** solution. In any minimal **DST** solution (i.e. a feasible $F \subseteq E$ that can not be made smaller by dropping an edge), every node will have indegree at most one since the solution is a directed tree spanning all terminals and, perhaps, some Steiner nodes. This justifies the indegree constraints. So the optimum LP solution value, denoted OPT_{LP}, is at most the cost of an optimal Steiner tree solution. We remark that (**DST-LP**) admits a simple polynomial-time separation oracle by simply checking that we can send one unit of $r-t$ flow to each terminal when edges have capacity x_e.

The integrality gap of this relaxation is well studied. First, Zosin and Khuller demonstrated the gap can be $\Omega(\sqrt{k})$ [19] in some instances. The number of vertices in their construction is exponential in the number of terminals, so the possibility of an $O(\log^c n)$ integrality gap bound was open. More recently, this was refuted by Li and Laekhanukit [15] who gave an example with integrality gap $\Omega(n^{0.0418})$. We remark that both [19] and [15] considered a different flow-based relaxation and their relaxation did not include the indegree bound for non-root nodes, but their examples are valid for (**DST-LP**).

Special Cases

Perhaps the first polylogarithimic integrality gap bound recorded for **DST** in certain settings was an $O(\log k)$ upper bound in **quasi-bipartite** instances. These are instances of **DST** such that every edge has at most one of its endpoints being a Steiner node. Another way to say this is that the subgraph induced by Steiner nodes contains no edges. Hibi and Fujito first gave an $O(\log k)$-approximation for this setting [14] and Friggstad, Könemann, and Shadravan then gave a primal-dual algorithm that demonstrated the integrality gap of (**DST-LP**) (even without the indegree constraints) is bounded by $O(\log k)$ [7]. In quasi-bipartite instances of **DST** where the underlying undirected graph excludes a fixed minor (e.g. planar graphs), (**DST-LP**) is known to have an integrality gap of $O(1)$ [8].

Chan et al. [3] generalized the $O(\log k)$ integrality gap bound to higher connectivity settings. They demonstrate an appropriate generalization of (**DST-LP**) (without the indegree constraints) for the problem of finding the cheapest $F \subseteq E$ ensuring r is at least R-edge connected to each terminal has an integrality gap bound of $O(\log k \cdot \log R)$.

Nutov [16] extended this to more settings involving more general supermodular cut requirement functions in with relaxations to the quasi-bipartite property. Namely, [16] considers a cut requirement function $f : 2^{V-\{r\}} \to \mathbb{Z}_{\geq 0}$ satisfies $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$ whenever $f(A) > 0, f(B) > 0$ and $A \cap B \cap T = \emptyset$. If one further has the property that every edge has an endpoint v such that $v \in X$ or $f(A) = 0$ for each $\{v\} \subseteq A \subseteq V - \{r\}$. In this case, [16] gives an $O(\log k \cdot \log R)$-approximation where R is the maximum value taken by f. Note, this does not capture our setting as our graphs can have edges (u, v) with both u, v being Steiner nodes yet any $A \subseteq V - \{r\}$ with $u, v \in A$ and $A \cap X \neq \emptyset$ requires an incoming edge.

Layered Graphs

An instance of **DST** is ℓ-layered if V is partitioned as $V_1 = \{r\}, V_2, V_3, \ldots, V_\ell = X$ and all edges $(u, v) \in E$ have $u \in V_i, v \in V_{i+1}$ for some $1 \leq i < \ell$. An α-approximation for **DST** in ℓ-layered graphs is known to yield an $O(\alpha \cdot \ell \cdot k^{1/\ell})$-approximation in general [18, 2]. This was the starting point for a k^ϵ-approximation by Charikar et al. [4].

The bad integrality gap examples in [19] and [15] are 5-layered instances of DST. It can easily be seen that 3-layered instances of DST (which are necessarily quasi-bipartite) have an integrality gap of $O(\log k)$ by adapting randomized SET COVER rounding techniques.

Friggstad et al. show the integrality gap of (**DST-LP**) remains $O(\log k)$ even in 4-layered instances [6]. They do this by mapping an LP solution to a natural relaxation for a related instance of GROUP STEINER TREE in a tree with constant height and using the known integrality gap bound for such instances [10]. Intuitively, this is possible since the first two layers of edges can only be reached in one way and each edge in the last layer is only used to connect to one terminal.

The behavior of LP relaxations for **DST** under hierarchies has also been considered. First, Rothvoss showed for ℓ-layered graphs that lifting a related flow-based LP relaxation through $O(\ell)$ layers of the Laserre hierarchy reduces the integrality gap to $O(\ell \cdot \log k)$ [17]. Later, [6] showed the result holds for a considerably weaker version of (**DST-LP**) that is valid only for layered graphs and using only the LP-based hierarchies of Lovasz-Schrijver and Sherali-Adams.

Undirected Graphs

Finally, it should be noted that in undirected graphs, the integrality gap of a related relaxation with undirected cut constraints $x(\delta(S)) \geq 1$ (and no vertex degree constraints) is well-known to be exactly 2. If one considers the bi-directed cut relaxation, i.e. the directed graph having both orientations of each undirected edge, then it is an open problem to determine if (**DST-LP**) has an integrality gap being some constant smaller than 2. It is at least known for quasi-bipartite graphs that the integrality gap of this bi-directed relaxation is better than 2 [9, 5]. Finally, a significant strengthening of the standard relaxation for general instances of undirected STEINER TREE, known as they hypergraphic relaxation, is known to have an integralty gap of $\ln(4)$ and can be efficiently solved to within any constant factor of the optimum solution cost in polynomial time [1, 11].

1.2 Our Results

We consider a generalization of **DST** in quasi-bipartite graphs and prove the following result.

▶ **Theorem 1.** *Suppose no Steiner node has both incoming and outgoing edges to other Steiner nodes. Then the integrality gap of* (**DST-LP**) *is* $O(\log k)$*.*

In other words, we consider instances where the subgraph induced by Steiner nodes may contain edges but not paths with more than one edge. Thus, this is a generalization of quasi-bipartite **DST**. This is also extends the integrality gap bound of $O(\log k)$ in 4-layered graphs [6] to a more general setting.

We emphasize that an $O(\log k)$-approximation for such graphs was already given by Hibi and Fujito [14]. The main purpose of our paper is to establish integrality gap bounds. The techniques in [14] seem unlikely to produce integrality gap bounds because they also produce $O(\log k)$-approximation for **DST** in 5-layered graphs, for which we know the integrality gap is not polylogarithmic (see Section 1.1).

Our algorithm can be seen as a common generalization of the rounding algorithm for quasi-bipartite instances from Chan et al. [3] and the analysis of Group Steiner Tree presented by Rothvoss [17]. At a high level, we round edges in phases: each phase will reduce the number of terminals we are required to connect by a constant while only paying $O(OPT_{LP})$ for the edges purchased each round.

In more detail, [3] identifies a maximal violated set around each such terminal (that excludes other required terminals) is identified and each iteration will "cover" the violated cuts in those sets. They show that no edge can be fully contained in more than one such maximal violated set around the required terminals. Unfortunately, that is not the case in our setting. Still, we can show the only edges shared between these maximal sets have at least one endpoint being a terminal, so the overlap in these sets is limited to edges between Steiner nodes. Then we use a variation of Group Steiner Tree rounding to ensure the edges e that might be used to connect to multiple nodes are only sampled with probability $O(x_e)$ in our algorithm.

2 Preliminaries

We call an edge $e = (u, v)$ a **Steiner edge** if both u and v are Steiner nodes. Call a Steiner node v a **source-Steiner** node if there is an edge (v, w) to another Steiner node w. Otherwise, call v a **sink-Steiner** node.

Recall for a subset of edges $F \subseteq E$ and a subset of nodes $S \subseteq V$ we let $\delta^{in}_F(S) = \{(u, v) \in F : u \notin S, v \in S\}$ be all edges of F entering S. Similarly, $\delta^{out}_F(S)$ are edges leaving S. If $F = E$, we may omit the subscript and simply write $\delta^{in}(S)$ and $\delta^{out}(S)$. For brevity, we also write $\delta^{in}_F(v)$ and $\delta^{out}_F(v)$ for a single node $v \in V$ to mean $\delta^{in}_F(\{v\})$ and $\delta^{out}_F(\{v\})$.

Without loss of generality, we assume there is no edge entering r (they can be deleted), no direct edge from $X \cup \{r\}$ to X (such an edge e can be subdivided with two Steiner nodes into a path of length 3 with each edge having cost $c_e/3$), and no Steiner node has no edge to any other Steiner node (such a Steiner node v can be split into two Steiner nodes v^+, v^- with a 0-cost edge from v^+ to v^-). It is straightforward to check these reductions do not change the optimal value of (**DST-LP**) and that we can map solutions between the original graph and the modified graph without increasing their costs. Again, throughout we will let OPT_{LP} denote the optimum solution value of (**DST-LP**).

2.1 Representative Terminals for Partial Solutions

Our algorithm will iteratively purchase subsets of edges over phases while making progress toward a feasible solution. So we need to understand the structure of a partial solution $F \subseteq E$ that does not necessarily connect r to each terminal. If some terminals can already reach other terminals in (V, F), we only need to focus on purchasing edges to ensure r is connected to a subset of terminals that can reach all other terminals.

For $F \subseteq E$, we consider the following **pruning** process. First, consider the strongly-connected components (SCCs) of (V, F). Since r has no incoming edges in G, then $\{r\}$ is an SCC of (V, F). Say an SCC C is a **terminal-source component** if $C \cap X \neq \emptyset$ and the only nodes in $X \cup \{r\}$ that can reach $C \cap X$ in the graph (V, F) are those already in C.

Let X_F consist of a single arbitrarily-chosen terminal in each terminal-source SCC. Note that in the graph (V, F) all terminals in X can be reached from some node in X_F but no node in X_F can be reached from any other node in X_F. To **prune** F means to iteratively remove edges from F arbitrarily as long as doing so preserves the property that every node in X can be reached from a node in $X_F \cup \{r\}$. After pruning, F looks like a directed forest where all non-singleton components have a node in $X_F \cup \{r\}$ as a root and only terminals as leaf nodes. We say F is **pruned** with respect to X_F after this process and we call X_F **representative terminals**.

▶ **Lemma 2.** *Let $F \subseteq E$ be pruned and $F' \subseteq E - F$. If $(V, F \cup F')$ contains an $r - t$ path for each $t \in X_F$, then in fact it contains an $r - t$ path for each $t' \in X$.*

Proof. Each $t \in X$ is reachable from some $t' \in X_F \cup \{r\}$ using edges in F. Since r can reach t' using edges in $F \cup F'$, it can also reach t using edges in $F \cup F'$ ◀

Additional useful properties of a pruned set of edges having roots $X_F \cup \{r\}$ are:

- Each terminal $t \in X$ can be reached from exactly one $t' \in X_F \cup \{r\}$.
- Each Steiner node u can be reached from at most one $t \in X_F \cup \{r\}$. If u can be reached this way, it is not a leaf node in its corresponding tree. If u cannot be reached from any $X_F \cup \{r\}$, it is isolated (has no incoming or outgoing edges in F).

2.2 Tracking Progress

We will find a set of edges $F' \subseteq E - F$ with cost bounded by the optimum solution value of (**DST-LP**) that, in some sense, improves overall connectivity when added to F. If we could also ensure the number of terminals not connected from r decreases by a constant factor when adding F' to F, we would be done since it would be sufficient to iterate the procedure $O(\log k)$ times.

This view too optimistic. Rather, we track progress a different way by showing $|X_F|$ decreases by a constant factor each iteration. First, we show it suffices to ensure a constant fraction of terminals in X_F can be reached by another node in $X_F \cup \{r\}$. This is essentially the same as Lemma 5 in [3], we include its proof for completeness in Appendix A.

▶ **Lemma 3.** *$0 < \alpha < 1$ and let $F' \subseteq E - F$ be such that for at least an α-fraction of $t \in X_F$, there is some other $t' \in X_F - \{t\}$ that can reach t in $(V, F \cup F')$. Then $|X_{F' \cup F}| \leq (1 - \alpha/2) \cdot |X_F|$.*

Thus, our main algorithm boils down to finding such a set F'.

▶ **Theorem 4.** *Suppose $X_F \neq \emptyset$. There is a universal constant $0 < \alpha < 1$ and a randomized algorithm with polynomial expected running time that is guaranteed to find a set $F' \subseteq E - F$ such that (a) at least an α-fraction of $t \in X_F$ are reachable from some $t' \in X_F - \{t\}$ in $(V, F \cup F')$, and (b) the cost of F' is $O(OPT_{LP})$.*

Proving Theorem 4 is the focus of Section 3.

Our final algorithm iterates the procedure from Theorem 4 and adds the resulting set F' to the current set of given edges F. Since $|X_F|$ starts at k and decreases geometrically, after $O(\log k)$ iterations the set of all edges F purchased satisfies $X_F = \emptyset$ (i.e. all terminals are reachable from r) and $cost(F) = O(\log k) \cdot OPT_{LP}$. This procedure is summarized in Algorithm 1.

Algorithm 1 **DST** Rounding.

```
Compute an optimal solution $x$ to (DST-LP).
$F \leftarrow \emptyset$
$X_F \leftarrow X \cup \{r\}$
while $X_F \neq \{r\}$ do
  Obtain $F' \subseteq E - F$ using the algorithm from Theorem 4.
  $F \leftarrow F \cup F'$
  Let $X_F$ be a set of terminals, one from each source SCC in $(V, F)$.
  Prune $F$ with respect to $X_F$.
return $F$
```

3 The Rounding Algorithm

This section is dedicated to the proof of Theorem 4. Let x be an optimal solution to (**DST-LP**). We further assume that we cannot decrease any x_e by any positive amount.

▶ **Lemma 5.** *For each edge $e = (u, v) \in E$, $x_e \leq 1$. Additionally, if $u \neq r$ then $x_e \leq x(\delta^{in}(u))$.*

In fact, these properties would hold for any optimal solution if G had no 0-cost edges, we are just making sure 0-cost edges are well-behaved under x for our algorithm.

Proof. That $x_e \leq 1$ is obvious because all cut constraints require 1 edge, so no edge would be chosen to an extent of more than 1 in a minimal solution.

For the sake of contradiction, suppose $u \neq r$ yet $x_e > x(\delta^{in}(u))$. We claim that x_e could be decreased, contradicting minimality of x again. To see the latter, suppose otherwise, i.e. $x(\delta^{in}(S)) = 1$ for some constraint S with $e \in \delta^{in}(S)$. One easily checks

$$\begin{aligned} x(\delta^{in}(S \cup \{u\})) &= x(\delta^{in}(S)) + x(\delta^{in}(u) \cap \delta^{out}(V - S)) - x(\delta^{out}(u) \cap \delta^{in}(S)) \\ &\leq x(\delta^{in}(S)) + x(\delta^{in}(u)) - x_e \\ &< x(\delta^{in}(S)) = 1. \end{aligned}$$

This contradicts feasibility of x. ◀

Now let $F \subseteq E$ be a set of given edges (i.e. purchased in previous iterations). Our rounding procedure helps extend paths outward from nodes reachable from a node in $X_F \cup \{r\}$ toward other nodes in X_F. It does this in three phases, with the first two being very simple.

Step 1 – Forming F_1

Consider an edge $e = (u, v)$ with $u \in X \cup \{r\}$ and v being a Steiner node. Let $F_1 \subseteq E - F$ be formed by including each $e \in E - F$ independently with probability x_e. Clearly the expected cost of F_1 at most the cost of x.

Step 2 – Forming F_2

Form $F_2 \subseteq E$ as follows. For each Steiner edge $e = (u, v)$, if $\delta^{in}_{F_1}(u) \neq \emptyset$ then add f to F_2 with probability $\frac{x_e}{x(\delta^{in}(u))}$. Note the denominator cannot be 0 if we had successfully added an edge of $\delta^{in}(u)$ to F_1. So by Lemma 5, this is a valid probability. Now,

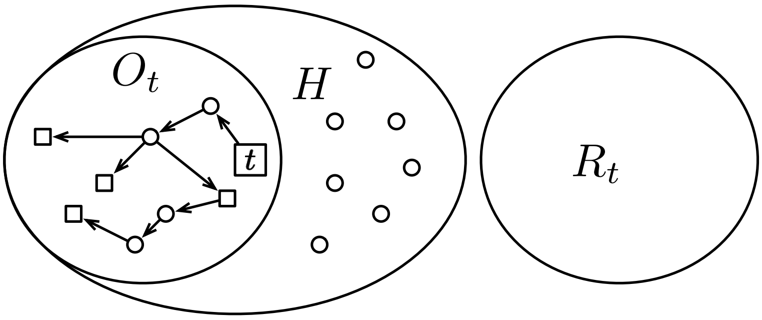


Figure 1 A depiction of the sets $O_t \cup H$ and R_t for some $t \in X_F$. Terminals are drawn as squares, Steiner nodes as circles. The edges shown are those in the pruned set F (though we do not show edges of F contained in R_t). The set R_t, which will be contracted to a single node we call r_t, consists of all nodes reachable from some other node in $X_F \cup \{r\}$ other than t. We just need to extend a path from R_t to t, the rounding algorithm we describe below will do this with constant probability.

$$\begin{aligned}
\mathbf{Pr}[e \in F_2] &= \mathbf{Pr}[e \in F_2 | \delta^{in}_{F_1}(u) \neq \emptyset] \cdot \mathbf{Pr}[\delta^{in}_{F_1}(u) \neq \emptyset] \\
&= x_e \cdot \frac{1 - \prod_{e \in \delta^{in}(u)}(1 - x_e)}{x(\delta^{in}(u))} \\
&\geq x_e \cdot \frac{1 - \exp(-x(\delta^{in}(u))}{x(\delta^{in}(u))} \\
&\geq (1 - \exp(-1)) \cdot x_e
\end{aligned}$$

The first inequality is a standard application of the arithmetic-geometric mean inequality and the bound $(1 - z/B)^B \leq \exp(-z)$ for $B \geq 1$ and $z \geq 0$. The second holds because $(1 - \exp(-1)) \cdot z \leq 1 - \exp(-z) \leq z$ holds for any $z \in [0, 1]$ and recalling the constraint $x(\delta^{in}(u)) \leq 1$ from (**DST-LP**)[1].

We also note a corresponding upper bound. The probability $\delta^{in}_{F_1}(u) \neq \emptyset$ is, by the union bound, at most $x(\delta^{in}(u))$. Using this upper bound above, we see $\mathbf{Pr}[e \in F_2] \leq x_e$.

Step 3 – Selecting the final set of edges

This step is considerably more involved, most of our new ideas are contained here. First, we discuss intuition.

Let H be all Steiner nodes v with $\delta^{in}_F(v) = \delta^{out}_F(v) = \emptyset$. For each terminal $t \in X_F$, let O_t be the set of all nodes (including t) that t can reach in (V, F). Since F is pruned, $O_t \cap O_{t'} = \emptyset$ for distinct $t, t' \in X_F$. Note that $R_t := V - (O_t \cup H)$ is the set of all nodes that can be reached by a node in $X_F - \{t\}$ using only edges in F, i.e. to reach t from $X_F - \{t\}$ it suffices to have any node in R_t reach t.

Finally, consider the graph G_t obtained by contracting R_t to a single vertex, keeping parallel edges that are created but discarding any loops. We let r_t denote this new node. Figure 1 illustrates these sets.

Now consider the following flow graph over G_t. For each edge e of G_t (i.e. an edge of G that was not contracted to a loop), install a capacity of x_e. Since $r \in R_t$, the LP constraints ensure we can send one unit of $r_t - t$ flow in G_t. We would like to sample a path from a

[1] This is the only point in our algorithm and analysis where we rely on this constraint.

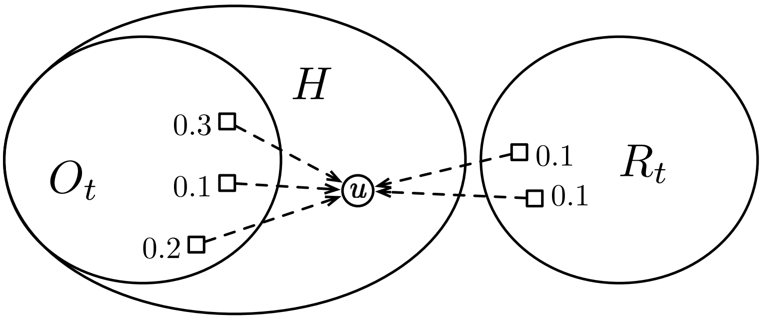


Figure 2 The Steiner node u is special for t as more than half of the value of $x(\delta^{in}(u))$ comes from nodes in O_t. Note u cannot be special for any other $t' \in X_F$ since their associated sets $O_{t'}$ are disjoint.

path decomposition of this flow, this would connect t from some other node in X_F and the expected cost of this path would be at most OPT_{LP} since no edge would be added with probability exceeding its x-value. The problem is that we cannot do this independently for different representative terminals in X_F since some edges are at risk of being considered multiple times.

We will show there is an $r_t - t$ flow of value $\geq 1/2$ that is safer to round. Intuitively, it will be that only the first two edges of any path in a path decomposition of this "safer" flow are at risk of supporting flows in G_t for too many terminals. The first two phases will have decided whether these edges will be included so we don't worry about oversampling them in this step.

Say a node $u \in H$ is **special** for terminal $t \in X_F$ if u is a source-Steiner node and the following holds:

$$\sum_{\substack{(w,u)\in\delta_G^{in}(u) \\ \text{s.t. } w\in O_t}} x_{(w,u)} > x(\delta_G^{in}(u))/2.$$

That is, u is special if more than half of the LP weight entering u comes from nodes only reachable from t. This is illustrated in Figure 2.

▷ Claim 6. Each node u is special for at most one terminal in X_F.

Proof. This is because $O_t \cap O_{t'} = \emptyset$ for distinct $t, t' \in X_F$, so at most one terminal $t \in X_F$ can have O_t claim more than half the LP weight of edges entering u. ◁

Finally, form a subgraph G'_t of G_t by including all vertices and edges except $\{(w,u) : w \in O_t$ and u is as source-steiner node that is **not** special for $t\}$. We can still push a constant amount of flow from r_t to t in G'_t, as the following shows.

▶ **Theorem 7.** *The maximum $r_t - t$ flow value in G'_t is at least $1/2$.*

Proof. For a graph G', we use notation $\delta_{G'}(S)$ to denote the set of edges of G' entering S to emphasize which graph we are discussing. Let $S \subseteq O_t \cup H$ be a subset of nodes in G'_t including t. Viewed as a subset of nodes in G, we have $x(\delta_G^{in}(S)) \geq 1$ by feasibility of the LP. Since G_t is obtained by contracting a subset of nodes lying outside of S, then $x(\delta_{G_t}(S)) \geq 1$ as well. Next we show in G'_t that this cut still has at least $1/2$ total x-weight in G'_t.

Consider any $(w,u) \in \delta^{in}(S)$. If u is not a source-Steiner or if u is special for t node then $(w,u) \in \delta^{in}_{S'}$. Otherwise, we know at least half of the weight of edges entering u comes from outside O_t, these would all be in $\delta^{in}_{G'_t}(S)$ as required. That is, $x(\delta^{in}(S)) \geq 1/2$. Since this holds for all $r_t - t$ cuts S, by the max-flow/min-cut theorem, G'_t supports at least 1/2 units of $r_t - t$ flow. ◀

Now consider any $r_t - t$ flow of value exactly 1/2 in G'_t and perform a path decomposition of this flow. That is, for various simple $r_t - t$ paths P we have a value $z_P \geq 0$ such that $\sum_P z_P = 1/2$ and $\sum_{P:e\in P} z_P \leq x_e$ for each edge e of G'_t. It is well known that such a decomposition exists with at most $|E|$ paths and can be computed in polynomial time.

Creating F_3^t

Finally we will create a set of edges F_3^t for each terminal $t \in X_F$ as follows. Consider an $r_t - t$ path P in the support of the path decomposition of G'_t. Let $E(P)$ denote the edges of G that correspond to edges of P. Let $e_1(P), e_2(P) \in E(P)$ be the first two edges of P (it may be that $|P| = 1$ in which case $e_2(P)$ is not defined). Write $e_1(P) = (v_1(P), v_2(P))$.

We consider the following random process to add some edges of $E(P)$, in doing so we also identify some **initial edges** $i(P)$ for the path P. Generally speaking, these are edges that we require to have been sampled in the formation of $F_1 \cup F_2$ in order for us to consider sampling the rest of the path P, though Case (iv) below differs slightly from this rule. Some of these cases are illustrated in Figure 3.

- **Case (i): $v_1(P)$ is a sink-Steiner node.**
  Set $i(P) := \emptyset$. With probability z_P, add all of $E(P)$ to F_3^t.
- **Case (ii): $v_1(P)$ is a source-Steiner node**
  Set $i(P) := \{e_1(P)\}$. If $e_1(P) \in F_2$, then with probability $z_P/x_{e_1(P)}$ add $E(P) - i(P)$ to F_3^t.
- **Case (iii): $v_1(P) \in X \cup \{r\}$ and $v_2(P)$ is special for t**
  Set $i(P) := \{e_1(P)\}$. If $e_1(P) \in F_1$, then with probability $z_P/x_{e_1(P)}$ add $E(P) - i(P)$ to F_3^t.
- **Case (iv): $v_1(P) \in X \cup \{r\}$ and $v_2(P)$ is not special for t**
  Then it must be that $e_2(P)$ is defined; set $i(P) := \{e_1(P), e_2(P)\}$. If $e_2(P) \in F_2$ and if **some** edge in $\delta^{in}(v_2(P)) \cap \delta^{out}(R_t)$ was added to F_1, then with probability $z_P/x_{e_2(P)}$ add $E(P) - i(P)$ to F_3^t with.

While case (ii) and (iii) are similar, there are important technical distinctions so we distinguish these cases for clarity in our analysis below. Note in all cases, if the random process adds edges of a path P to F_3^t it adds exactly the non-initial edges, i.e. $P - i(P)$.

3.1 Analysis of the Formation of the Sets F_3^t

We start by showing the probability any edge is added to a particular F_3^t is bounded by its x-value.

▶ **Lemma 8.** *For any $r_t - t$ path, the probability we added $E(P) - i(P)$ to F_3 due to processing P in its corresponding case is at most z_P. Consequently, for any $t \in X_F - \{r\}$ and any $e \in E, \mathbf{Pr}[e \in F_3^t] \leq x_e$.*

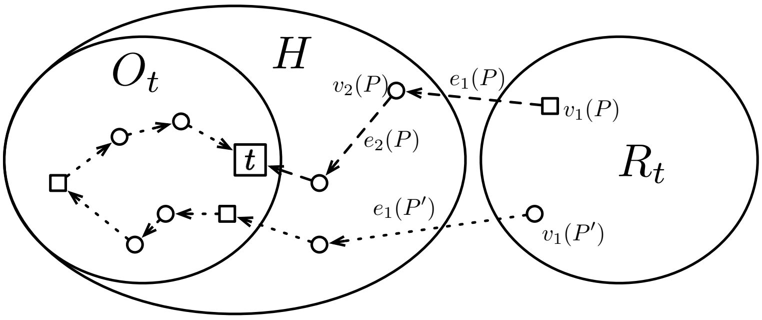


Figure 3 The graph G'_t except we have expanded node r_t to the full set R_t again. The top $r_t - t$ path (larger dashes) illustrates a path P that could either be from Case (iii) or Case (iv), depending on whether $v_2(P)$ is special for t or not. The lower path (with finer dots on the edges) illustrates a path P' from Case (ii). It might even be that some other path in the decomposition exits O_t after entering it before it eventually reaches t, but such a path could only use a Steiner edge (u, v) in H if after entering O_t if u was special for t since.

Proof. Focus on an $r_t - t$ path P and consider the corresponding case case for path P: (i) we simply added $E(P)$ with probability z_P, (ii) we added $E(P) - i(P)$ with probability $z_P/x_{e_1(P)}$ but only if $e_1(P) \in F_2$. As argued in **Step 2**, the latter happens with probability at most x_{e_1} so multiplying this against z_P/x_{e_1} finishes this case, (iii) $e_1(P)$ lies in F_1 with probability $x_{e_1(P)}$ so the total probability we added $E(P) - i(P)$ is exactly z_P.

For the final case (iv), P is sampled with probability $\frac{z_P}{x_{e_2(P)}}$ but only if the condition that includes $e_2(P) \in F_2$ is satisfied. Again, such a condition can only be satisfied with probability at most $x_{e_2}(P)$. Thus P is sampled with probability at most z_P overall.

The last statement in the lemma holds because the expected number of times an edge e is added to F_3^t is then at most $\sum_{P:e\in P} z_P \leq x_e$ because P is a path decomposition of a flow with capacity x_e on edge e. ◀

But this is not enough for a good overall cost bound, one should be concerned that an edge was added to multiple F_3^t sets for various t. The following effectively shows each edge that is a candidate to be added to some F_3^t can only support flow for at most one terminal $t \in X_F$.

▶ **Lemma 9.** *For each $e \in E$, there is at most one t such that $e \in E(P) - i(P)$ for some path P in the decomposition of the $r_t - t$ flow.*

Proof. Suppose $e = (u, v)$ has $v \in X \cup \{r\}$. The only such edges in G'_t have $v \in O_t$ since the only terminals not contracted into t_t are those in O_t. So e will only be an edge in G'_t for at most one t.

Next, suppose $u \in X \cup \{r\}$. If $u \notin O_t$ then $u \in R_t$ and $e = e_1(P)$ so we are in case (iii) or case (iv) for any path P containing e, but in either case $e \in i(P)$. Thus, we can only have $e \in E(P) - i(P)$ for the terminal t with $u \in O_t$.

Finally, suppose (u, v) is a Steiner edge. Suppose (u, v) lies on some path P in some G'_t. If $u \in R_t$ we are in case (ii) and $(u, v) \in i(P)$. If $u \notin R_t$, then either u is special for t or else the edge (w, u) prior to u is the first edge (i.e. $w \in R_t$) since we deleted all edges from O_t to u as u was not special for t. In the latter, we are in case (iv), so $(u, v) = e_2(P)$ means $(u, v) \in i(P)$. ◀

▶ **Theorem 10.** *The expected cost of $F_1 \cup F_2 \cup \bigcup_{t\in X_F - \{r\}} F_3^t$ is $O(OPT_{LP})$.*

Proof. We have already shown the expected costs of F_1 and F_2 are bounded by $O(OPT_{LP})$ since each edge is in F_1 or F_2 with probability at most x_e. We also know each e appears in any given F_3^t with probability at most x_3. Lemma 9 shows there is at most one F_3^t such that e has a nonzero probability of appearing in F_3^t, so e lies in $\bigcup_{t \in X_F - \{r\}} F_3^t$ with probability at most x_e. ◀

3.2 Success Probability

The last step is to show each terminal $t \in X_F$ can be reached from another node in $X_F - \{r\}$ with good probability. This is a bit subtle as there is shared randomness between the various $r_t - t$ paths P that reach t. Our analysis mirrors that in [17], which is providing an alternative analysis of the Group Steiner Tree rounding algorithm from [10].

We first require a general result about random variables. A proof was provided in [17] for the case $\mathbf{E}[X] = 1$. We need it in a slightly more general context so we include its proof in Appendix B for completeness.

▶ **Lemma 11.** *Let $\mu, \gamma \geq 0$ and let $X_1, X_2, \ldots, X_m$ be indicator random variables and $X = \sum_{i=1}^m X_i$ be their sum. Suppose $\mathbf{E}[X] \geq \mu$ and $\mathbf{E}[X|X_j = 1] \leq \gamma$ for any j. Then $\mathbf{Pr}[X \geq 1] \geq \mu/\gamma$.*

▶ **Theorem 12.** *There is a fixed constant $\alpha' > 0$ such that for each $t \in X_F$, with probability at least α' there is some $t' \in X_f \cup \{r\} - \{t\}$ such that t' can reach t in $(V, F \cup F')$.*

Proof. We show we added $E(P) - i(P)$ to F_3^t for at least one path P with constant probability, which suffices to prove the main result as then R_t could reach t along this path P. Consider the path decomposition and corresponding weights z_P. The subscripts in the sums on the right-hand side indicate which case the path corresponds to.

$$\frac{1}{2} = \sum_P z_P = \sum_{P:(i)} z_P + \sum_{P:(ii)} z_P + \sum_{P:(iii)} z_P + \sum_{P:(iv)} z_P$$

At least one of these sums is is at least 1/8.

Case: $\sum_{P:(i)} z_P \geq 1/8$. These paths were independently sampled with probability z_P each. The probability we did not pick one of them is then at most

$$\prod_{P:(i)} (1 - z_P) \leq \exp\left(-\sum_{P:(i)} z_P\right) \leq \exp(-1/8)$$

So at least one path was picked with probability $\geq 1 - \exp(-1/8)$.

Case: $\sum_{P:(ii)} z_P \geq 1/8$. All paths discussed here are those corresponding to case (ii) so we omit that qualifier throughout. We employ Lemma 11 where we have an indicator X_P for every path P and let $X = \sum_P X_P$. A path is added if both $e_1(P) \in F_1$ and then if P is sampled after that. This happens with probability $x_{e_1(P)} \cdot \frac{z_P}{x_{e_1(P)}} = z_P$. So $\mathbf{E}[X] \geq 1/8$.

Consider any particular path P', we want to bound $\mathbf{E}[X|X_{P'} = 1]$. We claim for any path P that

$$\mathbf{Pr}[X_P = 1 | X_{P'} = 1] = \begin{cases} 1 & \text{if } P = P' \\ z_P & \text{if } e_1(P) \neq e_1(P') \\ \frac{z_P}{x_{e_1(P')}} & \text{otherwise} \end{cases}$$

The first one is clear, the second is because $\mathbf{Pr}[X_P = 1] = \mathbf{Pr}[e_1(P) \in F_1] \cdot \frac{z_P}{x_e} = z_P$ and because the variables $X_P, X_{P'}$ are independent (since the random choice to add their initial edges F_1 were made independently). If $e_1(P) = e_1(P')$ then the only shared randomness between X_P and $X_{P'}$ was in the decision to add $e_1(P')$ to F_1. If we are given $X_{P'} = 1$, then we know $e_1(P') \in F_1$ but the choice to extend this to selecting P entirely was then made independently with probability $\frac{z_P}{x_{e_1(P')}}$.

So we have

$$\begin{aligned}
\mathbf{E}[X : X_{P'} \geq 1] &= \mathbf{Pr}[X_{P'} = 1 | X_{P'} = 1] + \sum_{P : e_1(P) \neq e_1(P')} \mathbf{Pr}[X_P = 1 | X_{P'} = 1] \\
&\quad + \sum_{P : P \neq P' \text{ and } e_1(P) = e_1(P')} \mathbf{Pr}[X_P = 1 | X_{P'} = 1] \\
&= 1 + \sum_{P : e_1(P) \neq e_1(P')} z_P + \sum_{P : P \neq P' \text{ and } e_1(P) = e_1(P')} \frac{z_P}{x_{e_1(P')}} \\
&\leq 1 + \frac{1}{2} + \frac{x_{e_1(P')}}{x_{e_1(P')}} \\
&= 5/2
\end{aligned}$$

That is, the total weight of all paths in the decomposition is at most the value of the flow, which is 1/2. Similarly, the total weight of all paths including the edge $e_1(P')$ is at most $x_{e_1(P')}$ since the flow respects capacities.

Using Lemma 11 with $\mu = 1/8$ and $\gamma = 5/2$ shows at least one path is sampled with probability at least 1/20.

Case: $\sum_{P:(iii)} z_P \geq 1/8$. The proof is essentially identical to the previous case and is omitted. We get the probability at least one path is sampled is at least 1/20.

Case: $\sum_{P:(iv)} z_P \geq 1/8$. Use similar indicator variables X_P and their sum X as in case (ii), but this time for the paths of form (iv). For any such path P, we have

$$\begin{aligned}
\mathbf{Pr}[X_P = 1] &= \frac{z_P}{x_e} \cdot \mathbf{Pr}[\delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset \wedge e_2(P) \in F_2] \\
&= \frac{z_P}{x_e} \cdot \mathbf{Pr}[\delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset] \\
&\quad \cdot \mathbf{Pr}[e_2(P) \in F_2 | \delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset] \\
&= \frac{z_P}{x_e} \cdot \mathbf{Pr}[\delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset] \cdot \frac{x_e}{x(\delta^{in}(u))} \\
&= \frac{z_P}{x(\delta^{in}(u))} \cdot \mathbf{Pr}[\delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset]
\end{aligned}$$

For brevity, let $B = \delta^{in}(v) \cap \delta^{out}(R_t)$. The last probability is

$$1 - \prod_{e \in B} (1 - x_e) \geq 1 - \exp\left(-\sum_{e \in B} x_e\right) \geq 1 - \exp(-x(\delta^{in}(v))/2)$$

The final inequality is because $v_2(P)$ is not special for t. For $z \in [0, 1]$, we have[2] $1 - \exp(-z/2) \geq (1 - \exp(-1/2)) \cdot z$, so the last expression is at least $(1 - \exp(-1/2)) \cdot x(\delta^{in}(u))$ and we finally see $\mathbf{Pr}[X_P = 1] \geq (1 - \exp(-1/2)) \cdot z_P$. Thus, $\mathbf{E}[X] \geq \frac{1 - \exp(-1/2)}{8}$.

[2] This holds since $1 - \exp(-z/2) = (1 - \exp(-1/2)) \cdot z$ for $z \in \{0, 1\}$ and since $1 - \exp(-z/2)$ is concave.

Finally, we upper bound $\mathbf{E}[X|X_{P'}=1]$ by a constant for any path P' considered in this case. Partition the set of paths from this case (iv) into four sets essentially based on how they interact with P' along their prefixes: $\{P'\}, \mathcal{P}_0 = \{P : v_2(P) \neq v_2(P')\}, \mathcal{P}_1 = \{P : v_2(P) = v_2(P')$ yet $e_2(P) \neq e_2(P')\}$, and $\mathcal{P}_2 = \{P : e_2(P) = e_2(P')\}$.

For $P \in \mathcal{P}_0$, simple inspection shows X_P and $X_{P'}$ are independent random variables so $\sum_{P\in\mathcal{P}_0} \mathbf{Pr}[X_P = 1|X_{P'} = 1] = \sum_{P\in\mathcal{P}_0} \mathbf{Pr}[X_P = 1] \leq \sum_{P\in\mathcal{P}_0} z_P \leq \frac{1}{2}$.

For $P \in \mathcal{P}_1$, we are given $\delta^{in}(v_2(P)) \cap \delta^{out}(R_t) \cap F_1 \neq \emptyset$ since $X_{P'} = 1$, so

$$\begin{aligned}\mathbf{Pr}[X_P = 1|X_{P'} = 1] &= \frac{z_P}{x_{e_2(P)}} \cdot \mathbf{Pr}[e_2(P) \in F_2|X_{P'} = 1] \\ &= \frac{z_P}{x_{e_2(P)}} \cdot \frac{x_{e_2(P)}}{x(\delta^{in}(v_2(P))} \\ &= \frac{z_P}{x(\delta^{in}(v_2(P))}.\end{aligned}$$

The total flow passing through $v_2(P)$ is at most its incoming edge capacity, so summing over all $P \in \mathcal{P}_1$ shows $\sum_{P\in\mathcal{P}_1} \mathbf{Pr}[X_P = 1|X_{P'} = 1] \leq 1$.

For $P \in \mathcal{P}_2$, we simply have $\mathbf{Pr}[X_P = 1|X_{P'} = 1] = \frac{z_P}{x_{e_2(P)}}$ since the condition to be met before sampling P is satisfied if we are given $X_{P'} = 1$. So $\sum_{P\in\mathcal{P}_2} \mathbf{Pr}[X_P = 1|X_{P'} = 1] = \sum_{P\in\mathcal{P}_2} \frac{z_P}{x_{e_2(P)}} \leq 1$. Thus,

$$\sum_P \mathbf{Pr}[X_P = 1|X_{P'} = 1] = 1 + \sum_{i\in\{0,1,2\}} \sum_{P\in\mathcal{P}_i} \mathbf{Pr}[X_P = 1|X_{P'} = 1] \leq 1 + \frac{1}{2} + 1 + 1 = 7/2.$$

Using Lemma 11 with $\mu = \frac{1-\exp(-1/2)}{8}$ and $\gamma = 7/2$ shows in the probability at least one path is sampled is at least some universal constant. Summarizing, no matter which case has at least 1/8 of the weight of paths we see there is a constant probability at least one path will be sampled. This completes the proof. ◀

We have shown the expected cost of the set $F' := F_1 \cup F_2 \cup \bigcup_{t\in X_F-\{r\}} F_3^t$ is at most $c \cdot OPT_{LP}$ for some universal constant c. We also showed each terminal $t \in X_F - \{r\}$ will be reachable from some other $t' \in X_t - \{t\}$ with probability at least some universal constant $\alpha' > 0$. So the expected number of terminals of this kind is at least $\alpha' \cdot |X_F|$.

Say this procedure failed if the cost of F' exceeds $\Delta \cdot c \cdot OT_{LP}$ for some constant Δ to be determined soon, or if the number of representative terminals that are now reachable from another representative is smaller than $\frac{\alpha'}{2} \cdot |X_F|$. Note we can check this condition in polynomial time.

The former happens with probability at most $1/\Delta$ by Markov's inequality. A standard variant of Markov's inequality for lower tails shows that if Y is a random variable with $\mathbf{E}[Y] \geq \alpha' \cdot M$ where M is the maximum possible value of Y, then $\mathbf{Pr}[Y < \frac{\alpha'}{2} \cdot M] \leq \frac{1-\alpha'}{1-\alpha'/2} < 1$. In our setting, we let Y be the number of representative terminals that become connected from another node in X_t after buying F', so the maximum value of Y is $|X_t|$ and the expected value is at least $\alpha \cdot |X_t|$.

Thus, by the union bound the procedure fails with probability at most $\frac{1}{\Delta} + \frac{1-\alpha}{1-\alpha/2}$. For sufficiently large constant Δ depending only on α, this is a constant less than one. That is, the procedure succeeds with constant probability. The final randomized algorithm then iterates this procedure until it does not fail, the expected number of iterations is constant. This proves Theorem 4.

References

1 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1), 2013.

2 Gruia Calinescu and Alexander Zelikovsky. The polymatroid steiner problems. *J. Combonatorial Optimization*, 33(3):281–294, 2005.

3 Chun-Hsiang Chan, Bundit Laekhanukit, Hao-Ting Wei, and Yuhao Zhang. Polylogarithmic approximation algorithm for k-connected directed steiner tree on quasi-bipartite graphs. *arXiv preprint*, 2019. `arXiv:1911.09150`.

4 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

5 Andreas Emil Feldmann, Jochen Könemann, Neil Olver, and Laura Sanità. On the equivalence of the bidirected and hypergraphic relaxations for steiner tree. *Mathematical Programming*, 160:379–406, 2014.

6 Zachary Friggstad, Jochen Könemann, Young Kun-Ko, Anand Louis, Mohammad Shadravan, and Madhur Tulsiani. Linear programming hierarchies suffice for directed steiner tree. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 285–296. Springer, 2014.

7 Zachary Friggstad, Jochen Könemann, and Mohammad Shadravan. A Logarithmic Integrality Gap Bound for Directed Steiner Tree in Quasi-bipartite Graphs . In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

8 Zachary Friggstad and Ramin Mousavi. A Constant-Factor Approximation for Quasi-Bipartite Directed Steiner Tree on Minor-Free Graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, pages 13:1–13:18, 2023.

9 Isaac Fung, Konstantinos Georgiou, Jochen Könemann, and Malcolm Sharpe. Efficient algorithms for solving hypergraphic steiner tree relaxations in quasi-bipartite instances. *CoRR*, abs/1202.5049, 2012. `arXiv:1202.5049`.

10 Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000.

11 Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, pages 1161–1176, 2012.

12 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. O (log2 k/log log k)-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–264, 2019.

13 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 585–594, 2003.

14 Tomoya Hibi and Toshihiro Fujito. Multi-rooted greedy approximation of directed steiner trees with applications. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 215–224. Springer, 2012.

15 Shi Li and Bundit Laekhanukit. Polynomial integrality gap of flow lp for directed steiner tree. *arXiv preprint*, 2021. `arXiv:2110.13350`.

16 Zeev Nutov. On rooted k-connectivity problems in quasi-bipartite digraphs. In *Computer Science – Theory and Applications: 16th International Computer Science Symposium*, pages 339–348. Springer-Verlag, 2021.

17 Thomas Rothvoß. Directed steiner tree and the lasserre hierarchy. *arXiv preprint*, 2011. `arXiv:1111.5473`.

18 Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

19 Leonid Zosin and Samir Khuller. On directed steiner trees. In *SODA*, volume 2, pages 59–63. Citeseer, 2002.

A Proof of Lemma 3

Proof. For each $t \in X_F$ that can be reached from some other $t' \in X_F \cup \{r\} - \{t\}$ in $(V, F \cup F')$, let $d(t) = t'$. If t can be reached from multiple such t', pick one arbitrarily to be $d(t)$. Finally, let F^* be all such edges $(d(t), t)$. We note that $(V, F \cup F')$ and $(V, F \cup F' \cup F^*)$ have the same SCCs because F^* provides direct connections between nodes that were already reachable in $(V, F \cup F')$.

We add the edges of F^* one at a time to (V, F) and track how the number of terminal-source SCCs decreases. Recall an SCC of (V, F) is a strongly connected component C containing a terminal that cannot be reached from any other terminal apart from those in C.

When adding $e_t = (d(t), t)$, let $S_{d(t)}$ and S_t be the SCCs containing $d(t)$ and t respectively at that time. We note S_t was a source SCC just before adding e_t because no edge entered the source component containing t before this addition.

If the number of terminal-source SCCs does not decrease after adding S_t, it must have been that S_t could already reach $S_{d(t)}$ by some path P. Let e' be the edge entering $S_{d(t)}$. Note $e' \in F^*$ since no vertex outside of $d(t)$'s SCC in (V, F) could reach $d(t)$ before (as it was a source SCC). Also note that e_t and e' are now drawn into the same SCC as S_t after e_t is added so e' will never enter another SCC again as we continue adding edges of F^*. That is, the number of iterations of adding an edge of the form e_t that do not cause the number of source SCCs to drop is at most $\alpha/2 \cdot |X_F|$, meaning the number of source SCCs in $(V, F \cup F^*)$ is at most $(1 - \alpha/2) \cdot |X_F|$. Thus, the number of terminal-source SCCs in $(V, F \cup F')$ is also bounded by $(1 - \alpha/2) \cdot |X_F|$ as required. ◀

B Proof of Lemma 11

Proof. This proof essentially just verifies the arguments in [17] generalize as required. Including the proof here also keeps our paper self-contained.

We do this in two steps. First, suppose we knew $\mathbf{E}[X|X \geq 1] \leq \gamma$. Then

$$\mu \leq \mathbf{E}[X] = \mathbf{E}[X|X = 0] \cdot \mathbf{Pr}[X = 0] + \mathbf{E}[X|X \geq 1] \cdot \mathbf{Pr}[X \geq 1] \leq \gamma \cdot \mathbf{Pr}[X \geq 1].$$

Rearranging shows $\mathbf{Pr}[X \geq 1] \geq \mu/\gamma$ which is what we wanted to show.

Now we show $\mathbf{E}[X|X \geq 1] \leq \gamma$ follows if $\mathbf{E}[X|X_j = 1] \leq \gamma$ for any j. By Jensen's inequality applied to the conditioned distribution, we have

$$\begin{aligned}
\mathbf{E}[X|X \geq 1]^2 &\leq \mathbf{E}[X^2|X \geq 1] \\
&= \sum_{(i,j)} \mathbf{Pr}[X_i = 1 \wedge X_j = 1|X \geq 1] \\
&= \sum_{(i,j)} \mathbf{Pr}[X_j = 1|X \geq 1 \wedge X_i = 1] \cdot \mathbf{Pr}[X_i = 1|X \geq 1] \\
&= \sum_{(i,j)} \mathbf{Pr}[X_j = 1|X_i = 1] \cdot \mathbf{Pr}[X_i = 1|X \geq 1] \\
&= \sum_{i} \mathbf{Pr}[X_i = 1|X \geq 1] \cdot \sum_{j} \mathbf{Pr}[X_j = 1|X_i = 1] \\
&= \sum_{i} \mathbf{Pr}[X_i = 1|X \geq 1] \cdot \mathbf{E}[X|X_i = 1] \\
&\leq \gamma \cdot \sum_{i} \mathbf{Pr}[X_i = 1|X \geq 1] \\
&= \gamma \cdot \mathbf{E}[X|X \geq 1]
\end{aligned}$$

All sums over (i, j) are over all m^2 ordered pairs of indices. To conclude, $\mathbf{E}[X|X \geq 1]^2 \leq \gamma \cdot \mathbf{E}[X|X \geq 1]$ and $\gamma \geq 0$ can only happen if $\mathbf{E}[X|X \geq 1] \leq \gamma$. ◀

Optimizing Symbol Visibility Through Displacement

Bernd Gärtner
Department of Computer Science, ETH Zürich, Switzerland

Vishwas Kalani
Department of Computer Science and Engineering, I.I.T. Delhi, India

Meghana M. Reddy[1]
Department of Computer Science, ETH Zürich, Switzerland

Wouter Meulemans
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Bettina Speckmann
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Miloš Stojaković
Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia

Abstract

In information visualization, the position of symbols often encodes associated data values. When visualizing data elements with both a numerical and a categorical dimension, positioning in the categorical axis admits some flexibility. This flexibility can be exploited to reduce symbol overlap, and thereby increase legibility. In this paper we initialize the algorithmic study of optimizing symbol legibility via a limited displacement of the symbols.

Specifically, we consider unit square symbols that need to be placed at specified y-coordinates. We optimize the drawing order of the symbols as well as their x-displacement, constrained within a rectangular container, to maximize the minimum visible perimeter over all squares. If the container has width and height at most 2, there is a point that stabs all squares. In this case, we prove that a staircase layout is arbitrarily close to optimality and can be computed in $O(n \log n)$ time. If the width is at most 2, there is a vertical line that stabs all squares, and in this case, we give a 2-approximation algorithm (assuming fixed container height) that runs in $O(n \log n)$ time. As a minimum visible perimeter of 2 is always trivially achievable, we measure this approximation with respect to the visible perimeter exceeding 2. We show that, despite its simplicity, the algorithm gives asymptotically optimal results for certain instances.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases symbol placement, visibility, jittering, stacking order

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.24

Funding *Meghana M. Reddy*: Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.
Wouter Meulemans: Partially supported by the Dutch Research Council (NWO) under project number VI.Vidi.223.137.
Miloš Stojaković: Partly supported by Ministry of Science, Technological Development and Innovation of Republic of Serbia (Grants 451-03-66/2024-03/200125 & 451-03-65/2024-03/200125). Partly supported by Provincial Secretariat for Higher Education and Scientific Research, Province of Vojvodina (Grant No. 142-451-2686/2021).

Acknowledgements This research was initiated at the 19th Gremo's Workshop on Open Problems (GWOP), Binn, Switzerland, June 13–17, 2022.

[1] The third author's full last name consists of two words and is *Mallik Reddy*. However, she consistently refers to herself with the first word of her last name being abbreviated.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 24; pp. 24:1–24:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

When communicating information visually, the position of symbols is an important visual channel to encode properties of the data. For example, in a scatter plot that visualizes age versus income of a given population, each data item (a person in the population) is visualized with a symbol (commonly a square, a cross, or a circle) which is placed at an x-coordinate that corresponds to their age and a y-coordinate that corresponds to their income. Hence persons with similar values are placed in close proximity, which allows the user to visually detect patterns. Another example from cartography are so-called proportional symbols maps which visualize numerical data associated with point locations by placing a scaled symbol (typically an opaque disc or square) at the corresponding point on the map. The size of the symbol is proportional to the data value of its location, such as the magnitude of an earthquake. The density and size of the symbols again supports visual pattern detection.

In both examples above, the position of the symbol is fixed and cannot be changed without severely distorting the information it encodes. In other settings, such as map labeling, the position of a symbol is not completely fixed, instead the symbol (the label) needs to be placed in contact with a particular point on the map. There are infinitely many potential placements for the symbol, but all must contain the same fixed point somewhere on its boundary. In this paper we consider a related symbol placement problem, which is motivated by the visualization of numerical data with associated categories: the age of employees within a certain division of a company or the page rank of tweets that exhibit a certain sentiment (positive, negative, neutral) on a topic such as vaccinations. Such data can be visualized in *categorical strips* of fixed width w, restricting the symbols to lie in the strip, and placing the symbols on a y-coordinate according to their numerical values (see Figure 1 for an example using twitter data). There are again infinitely many potential placements per symbol, but all placements are restricted to share the same y-coordinate.

If the positions of symbols are fixed or restricted, then close symbols will overlap, reducing the visible part of – or even fully obscuring – other symbols. Correspondingly, there is an ample body of work on optimizing the visibility of symbols under placement restrictions. The algorithmic literature considers a couple of variants. First of all, we either display all symbols or only a subset. For symbol maps and our categorical strips we always have to display all symbols, since otherwise not all data is visualized. For map labeling one usually chooses a subset of the labels which can be placed without any overlap; the corresponding optimization problems attempt to maximize the number of these labels while also taking priorities (such as city sizes) into account. If overlap between symbols might be unavoidable, visibility is optimized by either maximizing the minimum perimeter of the symbol that has least visibility or maximising the total visible perimeter. If the positions of the symbols are completely fixed, then the only choice we can make is the drawing order of the symbols.

In this paper we study the novel algorithmic question of how to optimize the visibility of a given set of symbols, all of which must be drawn, when we may choose their drawing order and their x-coordinate, given a set of fixed (and distinct) y-coordinates for each symbol. We measure the visibility of the result via the minimal visibility perimeter over all symbols [3]. Figure 1 shows that our algorithms do indeed greatly improve the visible perimeter and thereby give the viewer a more accurate impression of the data. Note that our theoretical results hold only for square symbols, but, as evidenced by Figure 1, the algorithms we propose readily extend to rectangular symbols. Proving similar bounds for more general symbol shapes is a challenging open problem.

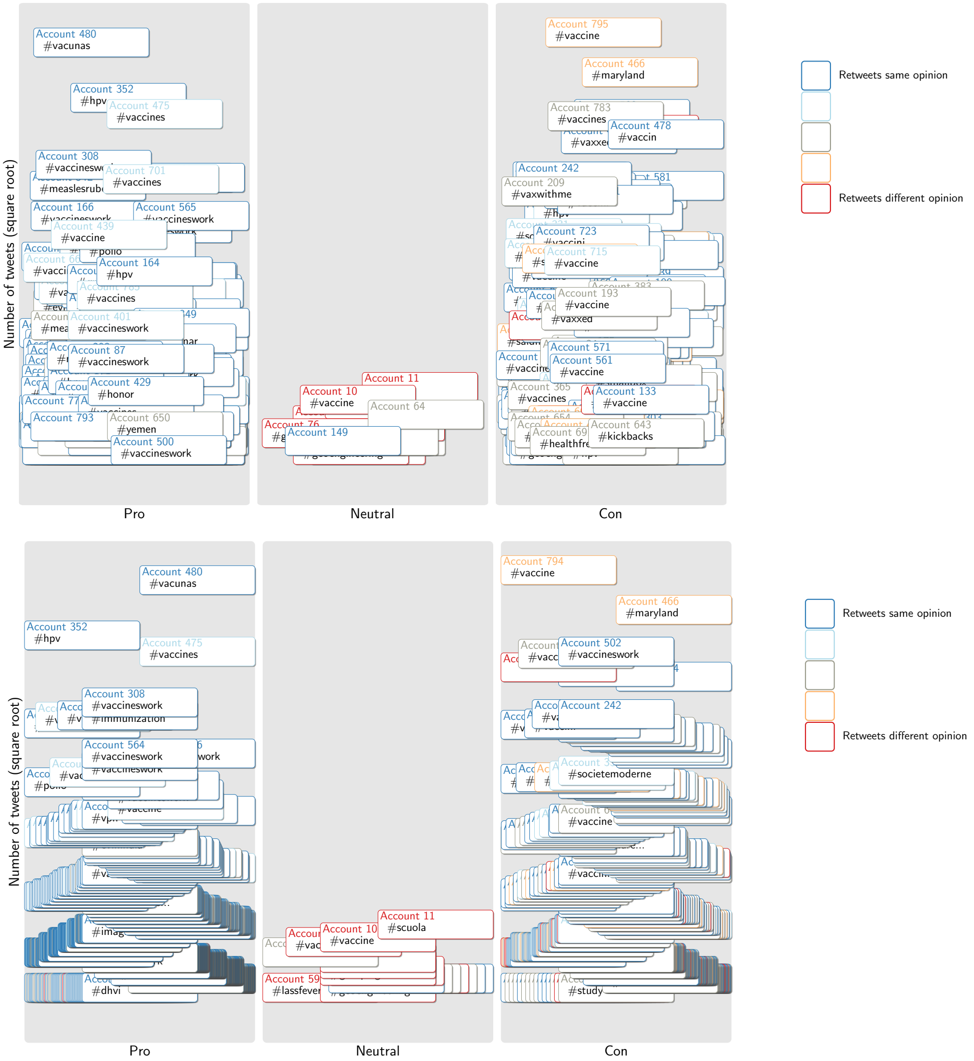


Figure 1 A pro and con vaccination twitter discussion from 2018, capturing 823 accounts. Accounts are placed in a categorical strip according to most frequent sentiment expressed. Color indicates if account mostly mentions accounts with a similar opinion (blue) or with a different opinion (red). Top: random jittering, bottom: our approximation algorithm described in Theorem 7. Random jittering hides many details, such as pro-vaccination accounts that mention predominantly anti-vaccination accounts (red boundaries). Many accounts at the bottom sent the same number of tweets and hence share a y-coordinate, which inevitably covers most of their horizontal edges.

Contributions and organization. In this paper we initiate the algorithmic study of optimizing symbol visibility through displacement. Specifically, we focus on unit square symbols, that may be shifted horizontally while remaining in a strip of width 2 (their categorical strip). In Section 2 we introduce our notation and make some initial observations. Most notably: the visible perimeter behaves non-continuously when squares are placed on the same x-coordinate. Hence the optimal visible perimeter is a supremum that cannot always be reached. In Section 3 we study the special case that the strip has height at most 2. In this scenario all squares are stabbed by a point. We first establish several useful geometric properties of so-called reasonable layouts – arrangements of the input squares which meet certain lower bound conditions – and then use these properties to prove that a simple $O(n \log n)$ algorithm suffices to compute a layout of the squares whose visible perimeter is arbitrarily close to the supremum.

In Section 4 we then study the general case of strips of arbitrary height (but still width 2). Here all squares are stabbed by a line. We leverage our previous result to obtain an $O(n \log n)$-time approximation algorithm. This approximation is with respect to the so-called gap – the visible perimeter minus two – since a minimal visible perimeter of 2 is trivially obtained for any instance. Furthermore, if the y-coordinates are uniformly distributed, then we can show that a specific layout – the zigzag layout – is asymptotically optimal. We close with a discussion of various avenues for future work, both towards the practical applicability of our results in visualization systems and towards more theoretical results in other settings, including different visibility definitions and other symbol shapes.

Related work. The questions we study in this paper combine various aspects and restrictions of well-known placement problems in the algorithmic (geo-)visualization literature in a novel way. Most closely related to our work are the two papers by Cabello et al. [3] and by Nivasch et al. [15] that consider perimeter problems for sets of differently sized symbols at fixed positions in the plane. Specifically, they consider the problems of maximizing the minimum perimeter of the symbol that has least visibility (as we do in this paper) or maximizing the total visible perimeter. Since the locations of the symbols are fixed, the algorithmic problem reduces to finding the optimal (not necessarily stacking) order of the symbols. This contrasts with our work where a limited form of displacement is allowed.

Labeling cartographic maps, where a subset of the labels are chosen such that they can be placed without any overlap, and the corresponding optimization problems are computationally complex in various settings [8], as they relate to maximum independent set. Constrained displacement of labels is often also allowed, and various of such placement models have been studied algorithmically [2, 4, 16, 17, 21]. The goal is always to place as many labels as possible without overlap. Displacing labels to the boundary of a map has attracted considerable algorithmic attention under different models as well, see [1] for a survey. However, such practice relies on leader lines to connect labels to the points being labeled; making it harder to identify data patterns and thus less suitable for visualizing data.

There is ample work on using symbol displacement to eliminate all symbol overlap, as it has various applications in visualization, including visualizing disjoint glyphs in geovisualization [11, 13, 20], removing overlap between vertices in graph drawing [6, 12], positioning nonspatial data [10, 18] and computing Dorling and Demer's cartograms [5, 14]. Such overlap-removal problems are NP-hard in many settings [7, 19], though efficiently solvable in some specific settings [13, 14]. However, eliminating all overlap may require considerable displacement, thereby distorting the view of the data. Our goal is not to eliminate all overlap, but to use both a limited displacement and a suitable drawing order to maximize visibility.

In the visualization literature, offsetting graphical symbols to improve visibility is known as jittering [22]. Often, jittering is done randomly, though in context of dense plots, arising, for example, from dimensionality reduction, more complex algorithms have been engineered for this task; see e.g. [9] for a recent method. However, such approaches are often heuristic in nature, without provable quality guarantees.

2 Preliminaries

Our input is a sequence of distinct y-coordinates $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. We want to find x-coordinates $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, determining unit squares $s_1, s_2, \ldots, s_n$, where s_i has centroid (x_i, y_i). We also want to find a stacking order, so that the minimum visible perimeter among all squares is maximized.

More specifically, we are given a *strip* T of width $w > 1$ and height $h > 1$, where we assume that $T = [0, w] \times [0, h]$, and that $\frac{1}{2} \leq y_1 < y_2 < \ldots < y_n \leq h - \frac{1}{2}$. Thus, in terms of the y-coordinates, we can speak about the *highest*, or the *lowest*, among any set of squares. We require $x_i \in [\frac{1}{2}, w - \frac{1}{2}]$ for all i, so that all squares are in the strip. We say that s_i is *left* (*right*) of s_j if $x_i < x_j$ ($x_i > x_j$). This determines a *leftmost* and a *rightmost* square among any set of squares (ties handled as needed).

A *stacking order* is a total order $\prec$ among the squares. If $i \prec j$, we say that s_j is *in front of* s_i, and s_i is *behind* s_j. The *bottom* and *top square* are the first and last squares in the order. The pair $(\mathbf{x}, \prec)$ is a *layout* for the instance $(w, h, \mathbf{y})$.

The *visible perimeter* of a square in a layout is the total length of all its *visible* boundary, where a point on the boundary is visible if any other square t containing it is behind s. The top square has visible perimeter 4. The visible boundary is made up of (horizontal or vertical) *visible edges.* Note that each side of s can host at most one visible edge, due to all squares having the same size.

The *gap* of a square in a layout is its visible perimeter minus 2. If this is non-positive, we say that the square has no gap. The gap of a layout is the minimum of the gaps of all squares. This definition is motivated by the fact that we can always achieve a positive gap by suitable "standard" layouts which we introduce below. Ideally, we want to find an optimal layout, one that has the maximum gap among all layouts. However, this may not exist: one can easily construct instances where the only candidates for optimal layouts have duplicate x-coordinates, but no such layouts can actually be optimal, refer to Figure 2 for one such instance. Therefore, we take the supremum gap over all layouts as the benchmark which we want to approximate as closely as possible.

The *bounding box* of a collection of squares is the inclusion-minimal axis-parallel box containing all the squares.

We call a layout a *staircase* if both $\mathbf{x}$ and $\prec$ are monotone:

- $x_1 \leq x_2 \leq \cdots \leq x_n$ ("facing right"), or $x_1 \geq x_2 \geq \cdots \geq x_n$ ("facing left"); and
- $s_1 \prec s_2 \prec \cdots \prec s_n$ ("facing up"), or $s_1 \succ s_2 \succ \cdots \succ s_n$ ("facing down").

Hence, there are 4 types of staircases; the one in Figure 3 (left) is facing right and up.

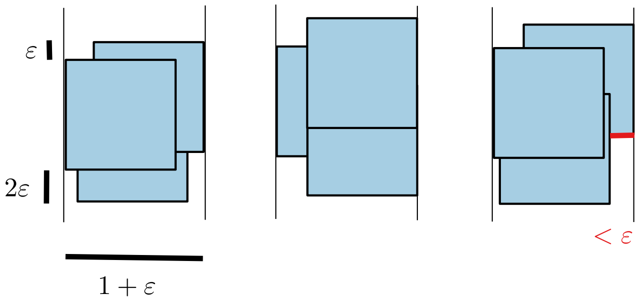


Figure 2 An instance with $y_2 - y_1 = 2\varepsilon, y_3 - y_2 = \epsilon$ and a strip of width $w = 1 + \varepsilon$, where $\varepsilon \leq 0.2$. From only looking at the two highest squares s_2, s_3, we see that the gap of every layout is bounded from above by 2ε, and to achieve this gap, s_2 and s_3 together need to span the full strip width. Given this, s_1 cannot be the bottom square, since then it has visible perimeter at most $1 + 5\varepsilon \leq 2$ and hence no gap *(left)*; s_2 cannot be the bottom square, either, as this would limit its visible perimeter to at most $1 + 2\varepsilon$ *(middle)*. So s_3 has to be the bottom square *(right)*. Since $x_1 = x_2$ would give the bottom square among s_1, s_2 a visible perimeter of at most $1 + 4\varepsilon$, we need to have $x_1 \neq x_2$, but this means that s_3 has gap less than 2ε. Hence, a gap of 2ε is not achievable, but any smaller gap is (by the right layout, as $x_1 - x_2 \to 0$).

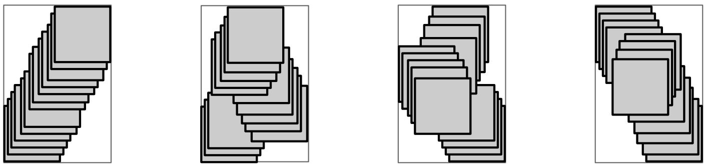


Figure 3 A staircase (left), three generalized staircases (right).

We call a layout a *generalized staircase* if each square s_i lies in one of the four corners of the bounding box of s_i and all squares in front of it. In a standard staircase, this corner is the same for all squares (i.e., the lower left corner for a staircase facing right and up).

▶ **Observation 1.** *For every instance, there is a staircase with positive gap.*

Proof. We build a staircase facing right and up as in Figure 3 (left). Then the left and lower sides of each square are completely visible, as well as parts of its right and upper side. This yields a positive gap. ◀

▶ **Definition 2.** *A layout is* reasonable *if it has positive gap, and it is ε-reasonable if it has gap larger than $\varepsilon > 0$.*

3 Squares stabbed by a point

Throughout this section, we fix a strip width $w \leq 2$ and a strip height $h \leq 2$. In this case, all squares are stabbed by a single point. Subsection 3.1 proves a (tight) upper bound on the gap of every layout, while Subsection 3.2 shows that a staircase layout of gap arbitrarily close to the supremum can efficiently be computed.

3.1 Reasonable layouts

We start with a crucial structural result about reasonable layouts in the case $w, h \leq 2$.

▶ **Proposition 3.** *In a reasonable layout, the bottom square s is not contained in the bounding box of the other squares.*

Proof. Assume otherwise and consider the two squares s_ℓ and s_r defining the left and right sides of the bounding box. If one of them is above s and the other one below, the situation is as in Figure 4 (left). Since s_ℓ and s_r overlap in both x- and y-coordinate, s has horizontal and vertical visible edges of total length at most 1 each. Thus, the visible perimeter of s is at most 2. In the other case, s_ℓ and s_r are w.l.o.g. both above s as in Figure 4 (right). Then we consider the square s_d defining the bottom side of the bounding box; w.l.o.g. s_d is left of s. In this case, s_r and s_d prove that s has no gap. ◀

▶ **Lemma 4.** *Every layout $\mathcal{A}$ of n squares has gap at most $\frac{w+h-2}{n-1}$, for $n \geq 2$.*

Proof. If $\mathcal{A}$ is unreasonable, there is nothing to prove. Otherwise, let $t_1, t_2, \ldots, t_n$ be the sequence of squares in stacking order, i.e. $t_1 \prec t_2 \prec \cdots \prec t_n$, and let the bounding box of $t_i, t_{i+1}, \ldots, t_n$ be denoted by τ_i for $1 \leq i \leq n$. Since $\mathcal{A}$ is reasonable, t_i "sticks out" of τ_{i+1} (i.e. it is not contained in it) by Proposition 3. There are two cases: t_i is a "corner square" (Figure 5 left), or a "side square" (Figure 5 middle and right). Let ΔX_i and ΔY_i quantify by how much t_i sticks out, horizontally and vertically. For a side square, one of those numbers is 0.

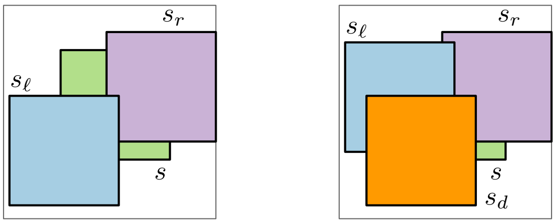


Figure 4 The visible perimeter of bottom square s is at most two if it is contained in the bounding box of the other squares.

For a corner square, the two sides of t_i incident to the corner contribute visible perimeter 2, meaning that the gap of the square is $\Delta X_i + \Delta Y_i$. For a horizontal side square as in Figure 5 (middle), the visible perimeter is at most $2 + \Delta Y_i$, and for a vertical side square (right), it is at most $2 + \Delta X_i$. In both cases, $\Delta X_i + \Delta Y_i$ is also an upper bound for the gap of the square.

This means that, for $1 \leq i < n$, the intervals corresponding to ΔX_i (as well as those corresponding to ΔY_i) span disjoint x-intervals (or y-intervals) that are also disjoint from the two intervals (one in each coordinate) of length one that is spanned by the top square. Hence,

$$\sum_{i=1}^{n-1} (\Delta X_i + \Delta Y_i) \leq (w-1) + (h-1) = w + h - 2.$$

It follows that there is some t_i with gap at most $\Delta X_i + \Delta Y_i \leq \frac{w+h-2}{n-1}$. ◀

This upper bound on the gap is easily seen to be tight.

▶ **Observation 5.** *There are instances of n squares for which a staircase has gap $\frac{w+h-2}{n-1}$.*

Proof. We consider the uniformly spaced instance ($y_i = \frac{1}{2} + (h-1)\frac{i-1}{n-1}$). Choosing $x_i = \frac{1}{2} + (w-1)\frac{i-1}{n-1}$ and $s_1 \prec s_2 \cdots \prec s_n$ leads to a staircase with $\Delta X_i = \frac{w-1}{n-1}$ and $\Delta Y_i = \frac{h-1}{n-1}$ for $1 \leq i < n$, and hence the gap is $\frac{w+h-2}{n-1}$. ◀

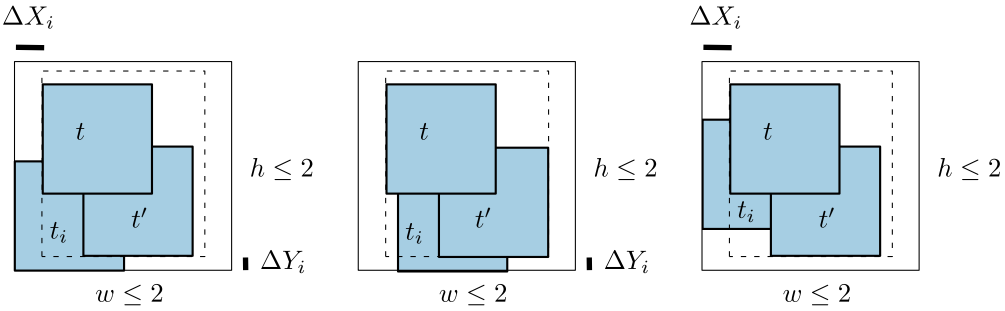


Figure 5 Proof of Lemma 4.

3.2 Computing staircases with gap arbitrarily close to the supremum

We next prove that for every reasonable layout with gap g, there is a staircase with a gap at least $g - \delta$, for any $\delta > 0$. Moreover, with $\gamma^\star$ being the supremum gap over all layouts, a staircase of gap $\gamma^\star - \delta$ can be efficiently computed.

For this, we first look at staircases in more detail. Consider a staircase of n squares, facing right and up, with centroids (x_i, y_i), $1 \leq i \leq n$. We define $\Delta y_i = y_{i+1} - y_i > 0$ and $\Delta x_i = x_{i+1} - x_i \geq 0$, for $1 \leq i \leq n-1$. If $\Delta x_i > 0$, the left and lower sides of s_i are fully visible, and the gap of s_i is $\Delta y_i + \Delta x_i$; the top square s_n has gap 2. If all Δx_i are positive, the staircase is called *proper*.

Now consider the problem of finding such a proper staircase of large gap. For this, the Δy_i are fixed, but $x_1 < x_2 < \cdots < x_n$ can be chosen freely, meaning that the values Δx_i can be any positive numbers satisfying $\sum_{i=1}^{n-1} \Delta x_i \leq w - 1$. We want to maximize the gap $\min_{i=1}^{n-1}(\Delta y_i + \Delta x_i)$ subject to the previously mentioned constraints. Due to the strict inequalities on the Δx_i's, the maximum may not exist (as pointed out in Section 2). But allowing $\Delta x_i \geq 0$, the maximum is attained by the solution of a linear program:

$$\begin{array}{lrcll} \text{maximize } g & & & & \\ \text{subject to} & \Delta x_i + \Delta y_i & \geq & g, & i = 1, \ldots, n-1 \\ & \sum_{i=1}^{n-1} \Delta x_i & \leq & w - 1 & \\ & \Delta x_i & \geq & 0, & i = 1, \ldots, n-1. \end{array} \tag{1}$$

Now we are prepared for the main result of this section which also implies that the optimal solution $g^\star$ of this linear program equals $\gamma^\star$, the supremum gap over all layouts.

▶ **Lemma 6.** *Let $\mathcal{A}$ be a reasonable layout with gap g. There exists a feasible solution of the linear program (1) with value at least g. Moreover, for every $\delta > 0$, there exists a proper staircase $\mathcal{A}'$ with gap at least $g - \delta$.*

Proof. As in the proof of Lemma 4, we consider the ΔX_i and ΔY_i, $i = 1, \ldots, n-1$, quantifying by how much the i-th square in the stacking order sticks out of the bounding box of the squares in front of it, horizontally and vertically. See Figure 6 (top left part) for an illustration, where the ΔX_i and ΔY_i values are visualized as rectangle areas. Each rectangle is spanned by a unit side and an *interval* corresponding to the value. For example, ΔX_4, the rectangle with the orange boundary, has the interval between the left sides of t_4 and t_5.

The ΔY_i rectangles are further subdivided into *blocks* whose intervals are gaps between vertically adjacent squares. For example, ΔY_4 is made of two blocks. The interval of the orange-black block is the vertical gap between the lower sides of t_4 (orange) and t_1 (black), while the interval of the black-green block is the vertical gap between the lower sides of t_1 (black) and t_5 (green). Thus, each block interval is of the form $\Delta y_j = y_{j+1} - y_j$.

As argued in the proof of Lemma 4, the ΔX_i and ΔY_i satisfy the constraints of the linear program (1), with g being the gap of $\mathcal{A}$.

We will perform discrete steps that gradually turn the ΔY_i into the prescribed $\Delta y_i = y_{i+1} - y_i$, while changing the ΔX_i into suitable Δx_i. If we can maintain the constraints of (1) throughout, we will arrive at a feasible solution of (1) – that we can interpret as a staircase – with value at least g. From this, we can construct a proper staircase with gap at least $g - \delta$, by slightly redistributing the Δx_i to make all of them positive.

We point out that the $\Delta X_i, \Delta Y_i$ are sorted by stacking order; our stepwise process will first result in values $\Delta X_i', \Delta Y_i'$ such that the $\Delta Y_i'$ are a permutation of the Δy_i. This means, we still have to sort the values accordingly before we can interpret the solution of (1) as a staircase. As the linear program is agnostic to permutations, this does not change the gap.

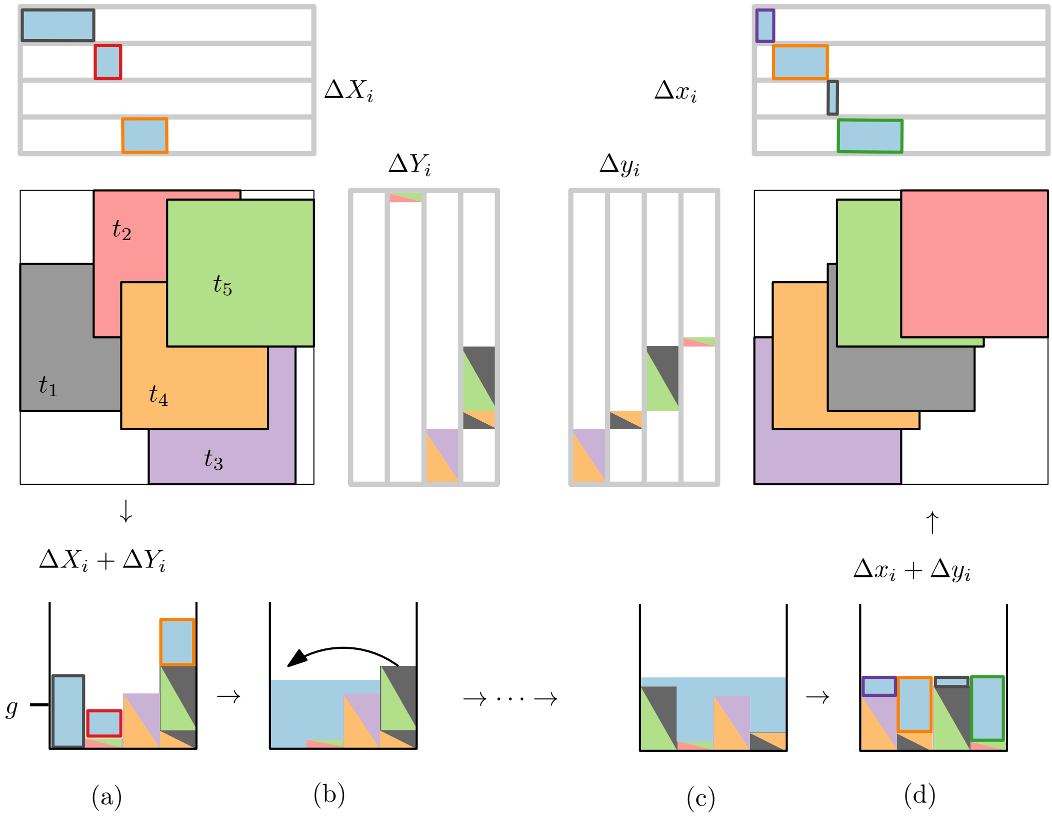


Figure 6 The proof of Lemma 6.

If the ΔY_i are already a permutation of the Δy_i, we are done after sorting them (see the end of the proof below). This is the case if and only if each ΔY_i consists of exactly one block.

But in general, some ΔY_i may have more than one block, or no block at all. In the example in Figure 6, we have $\Delta Y_4 = y_4 - y_2$ consisting of two blocks with intervals between t_4 (orange) and t_1 (black), and between t_1 (black) and t_5 (green). ΔY_1 in turn has no blocks, as t_1 does not stick out vertically.

All the ΔY_i together use all the $n - 1$ blocks. Indeed, the bounding box of the squares in front of t_i is disjoint from whatever sticks out of it, and the last bounding box only contains the top square.

Now we repeatedly move blocks from rectangles with at least two blocks to rectangles with no block. We can visually analyze this as follows: think of a basin that initially holds the ΔY_i rectangles, $i = 1, \ldots, n - 1$, as in part (a) of Figure 6. For each i, we pour ΔX_i units of water into the basin, which corresponds to the area of the rectangles with colored boundary in part (a) of Figure 6. As $\min_i(\Delta X_i + \Delta Y_i) \geq g$, the water will settle at some level $\geq g$; see part (b) of the figure. Moving a block to a "free slot" will submerge it further, and this can only increase the water level; see step (b)-(c).

In the end, we have one block $\Delta y_j = y_{j+1} - y_j$ per slot, and sorting the slots by index as in part (d) yields rectangles $\Delta y_1, \ldots, \Delta y_{n-1}$, with columns $\Delta x_1, \ldots \Delta x_{n-1}$ of water above them, such that $\min(\Delta x_i + \Delta y_i) \geq g$. In general, some Δy_i's can still be above the water level in which case the corresponding Δx_i is 0. This is our desired solution of (1) from which we can in turn build a staircase with the prescribed x- and y-gaps (upper right part of the figure). ◀

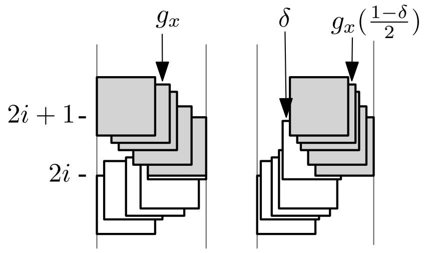


Figure 7 Squeezing staircase layouts.

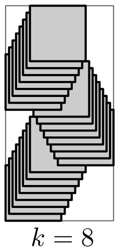


Figure 8 Arranging n uniformly spaced squares in a zigzag layout.

We remark that the linear program (1) can be efficiently solved in $O(n \log n)$ time, employing the water analogy. After sorting the Δy_i rectangles, and assuming that the water currently rises to the top of one of them, it is easy to compute in $O(1)$ time the amount of additional water required to reach the top of the next higher rectangle. Indeed, in this range, the water level is a linear function of the amount of additional water. If reaching the top of the next higher rectangle would need more water than our total budget of $w-1$ allows, we arrive at the optimal level $g^\star$ before.

4 Squares stabbed by a vertical line

Throughout this section, we consider a strip of width $w \le 2$ and arbitrary height $h > 1$, with n squares of fixed y-coordinates $\frac{1}{2} \le y_1 < y_2 < \cdots < y_n \le h - \frac{1}{2}$. Let $1/k = (h-1)/(n-1)$ be the (maximal) average y-distance between adjacent centroids in the y-order. We first show that we can asymptotically approximate the supremum gap up to a factor of 2. More precisely, as the strip remains fixed and $n \to \infty$, we have $1/k \to 0$ and thus approach a factor of 2 using Theorem 7 below. We still present our results in terms of k to make it clear what happens if the strip height h grows with n.

▶ **Theorem 7.** *Let $\gamma^\star$ be the supremum gap over all layouts. In time $O(n \log n)$, we can construct a layout with gap at least $\gamma^\star(\frac{1}{2} - O(\frac{1}{k}))$. We refer to this procedure as the* squeezing algorithm.

Proof. We partition the squares into buckets $1, \ldots, \lceil h \rceil$, where bucket i contains the squares j such that y_j rounds to i (we round up in case of a tie). The squares within each bucket are in a strip of height 2, and by Section 3, a (staircase) solution of gap arbitrarily close to the supremum can efficiently be found, in time $O(\ell \log \ell)$ per bucket, where ℓ is the number of squares in that bucket. Hence, the total time required is $O(n \log n)$.

The smallest bucket gap δ is (up to arbitrarily small error) an upper bound for $\gamma^\star$, as each layout contains a sublayout for the squares in this worst bucket. We also note that $\delta = O(\frac{1}{k})$, since there must be a bucket with $\Omega(k)$ squares to which Lemma 4 applies.

In $O(n)$ time, we now construct a layout for all squares, of gap roughly $\frac{\delta}{2}$, to prove the statement. To do so, we "squeeze" the layouts in individual buckets appropriately. We assume w.l.o.g. that the even bucket staircases are facing right and up, while the odd ones are facing left and up, as in Figure 7 (left).

Multiplying all x-gaps by $\frac{1-\delta}{2}$ while keeping the even staircases aligned left and the odd ones aligned right, see Figure 7 (left), leads to a layout where even staircase squares have $x \le 1 - \frac{\delta}{2}$, and odd ones have $x \ge 1 + \frac{\delta}{2}$. Each non-top square of each bucket still has gap $g_y + g_x \frac{1-\delta}{2}$ where g_x, g_y are the previous x-gap and y-gap in the bucket solution, and

$g = g_x + g_y \geq \delta$ is the previous gap. It follows that the new gap is at least $\delta\frac{1-\delta}{2}$. The top squares of each bucket have x-gap (and hence total gap) at least δ, by construction. The resulting layout has therefore gap at least $\delta\frac{1-\delta}{2}$. Since $\gamma^\star \lessapprox \delta = O(1/k)$, the bound follows. ◀

It is natural to ask whether squeezing the staircase layouts of individual buckets is the best we can do. For general y_i, we do not know the answer, but if the y_i are uniformly spaced, we can indeed prove that this procedure yields an asymptotically optimal gap.

Uniform spacing. For the rest of the section, we assume that $y_{i+1} - y_i = \frac{1}{k}$ for $1 \leq i < n$. In this case, the squeezing algorithm from Theorem 7 essentially produces the *zigzag* layout (see Figure 8).

▶ **Lemma 8.** *The zigzag layout has gap* $\frac{1}{k} + \frac{1}{2k-1}$.

Proof. See Figure 8. We place bundles of $\lfloor k \rfloor$ squares each, as indicated in the figure, starting from the lowest one. This layout uses precisely the $2\lfloor k \rfloor$ x-coordinates $\frac{1}{2} + \frac{i}{2\lfloor k \rfloor - 1}, i = 0, \dots, 2\lfloor k \rfloor - 1$. This means that every square has x-gap at least $\frac{1}{2\lfloor k \rfloor - 1} \geq \frac{1}{2k-1}$. The y-gap is at least $1/k$ for each square, due to uniform spacing. Both gaps are attained for example by the second-lowest square, so the bound in the lemma cannot be improved for this layout. ◀

Below, we will establish the following result, showing that the simple zigzag layout is asymptotically optimal.

▶ **Theorem 9.** *In the case of uniform spacing, every layout has gap at most* $\frac{1}{k} + \frac{1}{2k - O(\log k)}$.

In proving this, we can restrict to $1/k$-reasonable layouts, the ones achieving gap larger than $1/k$ in the first place. We also assume that $k \geq 2$.

We will start by establishing a crucial fact about such layouts, namely that most of their squares have 3 visible corners. To this end, we are going to upper-bound the number of squares with at least 2 covered corners, eventually enabling us to remove them from the layout while keeping most of the squares.

▶ **Definition 10.** *Given a layout, a* bad square *is one with at least 2 covered corners. A bad square with one vertical side covered is a* standard bad square*; see Figure 9 (left).*

▶ **Lemma 11.** *If a square s has both adjacent squares (in the y-order) in front of it, then s is a standard bad square.*

Proof. If the adjacent squares both have smaller or larger x-coordinate, then they together hide a vertical side of s, see Figure 9 (b). The other case cannot happen in a reasonable layout by Proposition 3; see Figure 9 (c)-(d). Since $k \geq 2$, the adjacent squares actually overlap vertically. ◀

Counting standard bad squares yields a bound for all bad squares.

▶ **Lemma 12.** *For each non-standard bad square, an adjacent square (in the y-order) is a standard bad square.*

Proof. Let s be a non-standard bad square. We distinguish two cases.

The first one is that an upper corner and a lower corner of s are covered. These could be adjacent corners (with some part of the connecting side visible), or antipodal corners as in Figure 10. By Lemma 11, one of the adjacent squares must be behind s; w.l.o.g. it is the next higher one b (blue).

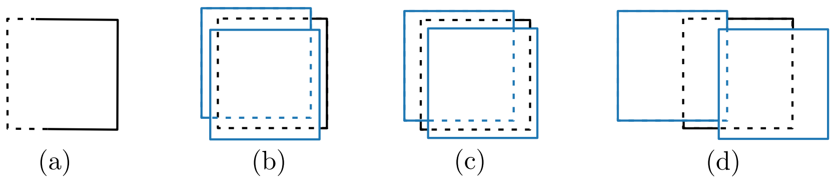


Figure 9 Bad squares: at least two covered corners; A standard bad square ((a) and (b)): one vertical side is covered.

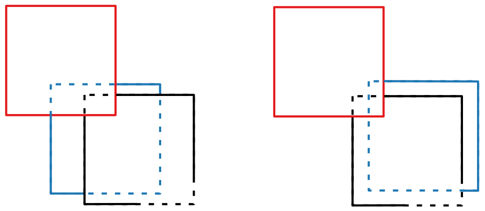


Figure 10 Case 1: A non-standard bad square (black) with an upper and a lower corner covered.

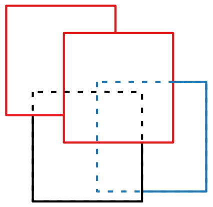


Figure 11 Case 2: A non-standard bad square (black) with two upper or two lower corners covered.

Consider the square r (red) covering the upper corner. Square b is behind s and r, and either "wedged" between them (w.r.t. to both x- and y-coordinate), or "sticking" out. The former case (Figure 10 left) cannot happen, because b would have no gap then, see Proposition 3. In the latter case, b is the required standard bad square (Figure 10 right). This uses that r is higher than b due to uniform spacing.

The second case is that two upper or two lower corners of s are covered, see Figure 11. Let us suppose w.l.o.g. that the two upper corners are covered. Then the upper side of s is covered. This implies that the next higher square b is behind s, as otherwise, s has gap at most $1/k$. Again, b is a standard bad square. ◀

Through the previous lemma, each standard bad square is "charged" by at most three bad squares (itself and the two adjacent ones).

▶ **Corollary 13.** *For every vertical window $W = [\underline{h}, \overline{h}] \subseteq [\frac{1}{2}, h - \frac{1}{2}]$ of a $1/k$-reasonable layout, the number of bad squares with $y_i \in W$ is at most three times the number of standard bad squares with $y_i \in W' = [\underline{h} - \frac{1}{k}, \overline{h} + \frac{1}{k}]$.*

It remains to count the number of standard bad squares.

▶ **Lemma 14.** *For every vertical window $W = [\underline{h}, \underline{h} + 1]$ of a $1/k$-reasonable layout, there are at most $2(\log k + 1)$ standard bad squares with $y_i \in W$.*

Proof. Let us fix the window. We count the standard bad squares with the left side covered, the overall bound follows by symmetry.

Let $s_1, \ldots, s_\ell$ be these squares; see Figure 12 (left). They must be stacked according to x-coordinate, with squares of lower x-coordinate in front of squares with higher x-coordinate. Indeed, a square s_i in front of a square s_j with smaller x-coordinate would cover a third corner of s_j, and thus a full horizontal side, resulting in no gap (we are using here that the window height is 1). The squares covering the left side of s_i are to the left of s_i and together cover all of s_i in the left half of the strip.

Because the layout is $1/k$-reasonable, each s_i has a part of each of its horizontal sides visible. They are of lengths $\sigma_i \leq \lambda_i \leq 1$ such that $\sigma_i + \lambda_i \geq 1 + 1/k$. Suppose that the squares are ordered by decreasing x-coordinate. We show that the σ_i increase exponentially with i.

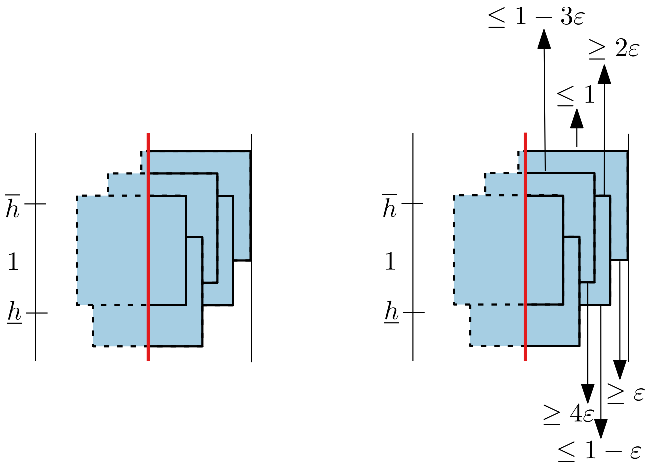


Figure 12 Counting standard bad squares with centers in a vertical window of height 1.

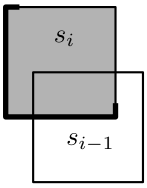


Figure 13 Proof of bitone stacking order.

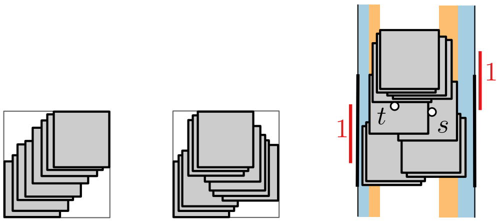


Figure 14 Two squares of different types in the subwindow.

We have $\lambda_1 \leq 1$, hence $\sigma_1 \geq \varepsilon := 1/k$; see Figure 12 (right). As a consequence, $\lambda_2 \leq 1 - \varepsilon$ (as s_2 is by at least ε further to the left than s_1). Hence, $\sigma_2 \geq 2\varepsilon$. This in turn means that s_3 is by at least $\epsilon + 2\varepsilon$ further to the left than s_1, so $\lambda_3 \leq 1 - 3\varepsilon$ and $\sigma_3 \geq 4\varepsilon$.

Continuing in this fashion, we see that $\sigma_\ell \geq 2^{\ell-1}\varepsilon \leq 1$. This implies that $\ell - 1 \leq \log(1/\varepsilon) = \log k$. ◀

▶ **Corollary 15.** *In a reasonable layout, at most* $6(\log k + 1)$ *squares out of any consecutive* $k - 1$ *squares are bad squares.*

Proof. The centers of $k - 1$ consecutive squares span a horizontal window of height $1 - 2/k$. Using Corollary 13 and the previous lemma, the number of bad squares in this window is at most $3(2(\log k + 1))$. ◀

Hence, by removing $O(\log k)$ squares per bundle of $k - 1$ squares, we obtain a layout with no bad squares left (observe that no surviving square can turn bad by removing squares). Such a layout turns out to have a rather rigid structure.

▶ **Lemma 16.** *After removal of all bad squares from a* $1/k$*-reasonable layout, there is a unique top square (fully visible), and the stacking order is determined: monotone decreasing from the top square towards the highest as well as the lowest square.*

Proof. A square with at least three visible corners (and only such squares remain) is called *down square* if the lower side is fully visible, and *up square* if the upper side is fully visible. A top square is both up and down.

Now let the squares be indexed from lowest to highest. We claim that if s_i is a down square, then s_{i-1} is also a down square that is behind s_i. To see this, consider a down square s_i and the overlapping square s_{i-1} (we have an overlap since we have removed only $O(\log k)$ squares in between); see Figure 13. It is clear that s_{i-1} must be behind s_i, and this in turn implies that s_i is also a down square.

A symmetric statement holds for up squares. Hence, starting from *any* top square, we can go both higher and lower, decreasing stacking height. In particular, we can never encounter another top square. ◀

We now proceed to the proof of Theorem 9, showing that, under uniform spacing, we cannot asymptotically beat the gap of the zigzag layout.

Proof of Theorem 9. We start with the layout obtained after removing all bad squares, as in Lemma 16. W.l.o.g. we assume that the majority of squares is below the top square, and we disregard all squares above. The stacking order then coincides with the y-order.

We now consider a window of height 4. Each square with center in that window is either left or right of the next higher square, and based on this, we call it type L, or type R. We focus on the middle subwindow of height 2. If all squares with centers in this subwindow have the same type, we have a proper staircase of $2k - O(\log k)$ squares; see Figure 14 (left).

Except $O(\log k)$ of them (the ones directly below a removed bad square), all squares have y-gap $1/k$. Let ε be the minimum x-gap among these squares. Then the layout has gap at most $1/k + \varepsilon$. But we know that the sum of all x-gaps is at most 1 (they don't overlap and "live" outside the top square), so the minimum x-gap is $1/(2k - O(\log k))$ which proves the theorem in this case.

The other case is that there are two consecutive squares of different types with centers in the subwindow, see Figure 14 (middle). In this case, we have a generalized staircase. We let s be the lower one and t the higher of the two squares, and we zoom in on the situation; see Figure 14 (right).

Both squares stick out of the ones up to 1 higher than s (otherwise, they can't have 3 visible corners). On the other hand, the squares up to 1 lower than t stick out of both s and t. It follows that – as in the previous case – the x-gaps of all involved squares live outside of the top square among them, and they do not overlap (i.e. they are disjoint). More specifically, the x-gaps of the squares above s live in the orange regions in Figure 14, while the x-gaps of the squares below t live in the blue regions.

In total, we again have $2k - O(\log k)$ squares with a y-gap of $1/k$ within the surrounding window of height 4, so the minimum x-gap is $1/(2k - O(\log k))$. ◀

5 Conclusion

We initiated the algorithmic study of optimizing the visibility of overlapping symbols by finding both a suitable drawing order and a limited displacement. This novel setting leads to various interesting and challenging problems. In this paper we focused solely on unit squares, presented structural insights, as well as several intricate approximation algorithms.

We are curious if the upper bound from Theorem 9 can be improved to one where the $O(\log k)$ term is replaced with a constant. In our approach we derive the bound by eliminating all the *bad* squares from the layout before estimating the gap. Hence, knowing that (in a vertical window of mutually intersecting squares) the number of bad squares can be as large as $\Theta(\log k)$ a radically new approach would be needed to achieve such bound improvement.

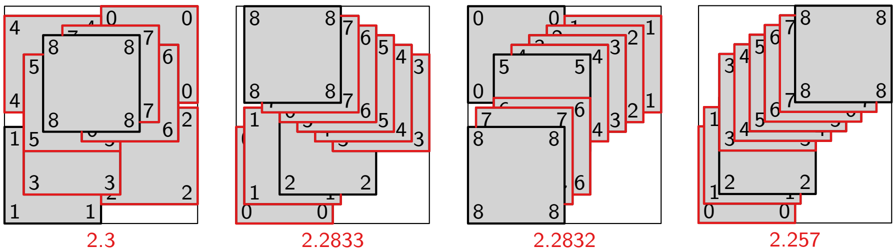


Figure 15 Four layouts for $y = \{0.5, 0.7, 0.8, 1.25, 1.35, 1.45, 1.55, 1.65, 1.75\}$. The minimum visible perimeter is indicated below each layout, the stacking order by the numbers in the corners of each square. From left to right: optimal layout; optimal layout with a stacking order matching the y-order; optimal layout with a stacking order matching the inverse y-order; optimal staircase. These layouts were computed via (I)LPs using a difference of 0.001 to turn strict inequalities into non-strict inequalities.

It is natural to wonder if the stacking order of every optimal solution follows its y-order. However, this is not always the case, refer to Figure 15 for an illustration: the optimal layout (leftmost figure) is better than the best layout when stacking order follows the y-order or the inverse y-order (second and third figures), which is better than the best layout among all staircases (rightmost figure).

An interesting and practically relevant scenario for future work are rectangular symbols. Our algorithms (constructions of layouts) can also be used for this case and yield results of high quality (see Figure 1). However, doing so loses the quality guarantees that we prove for the square case, since the resulting rectangle layouts will not optimize visible perimeter, but a variant of this measure in which horizontal and vertical visible edges have different weights. Even more challenging are settings with differently sized symbols. We leave these question to future work.

References

1 Michael A. Bekos, Benjamin Niedermann, and Martin Nöllenburg. External labeling techniques: A taxonomy and survey. *Computer Graphics Forum*, 38(3):833–860, 2019.

2 Sujoy Bhore, Robert Ganian, Guangping Li, Martin Nöllenburg, and Jules Wulms. Worbel: Aggregating point labels into word clouds. *ACM Transactions on Spatial Algorithms and Systems*, 9(3), 2023. `doi:10.1145/3603376`.

3 Sergio Cabello, Herman J. Haverkort, Marc J. van Kreveld, and Bettina Speckmann. Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3):543–565, 2010.

4 Thomas Depian, Guangping Li, Martin Nöllenburg, and Jules Wulms. Transitions in Dynamic Point Labeling. In *Proceedings of the 12th International Conference on Geographic Information Science (GIScience 2023)*, volume 277 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:19, 2023. `doi:10.4230/LIPIcs.GIScience.2023.2`.

5 Danny Dorling. *Area Cartograms: their Use and Creation*, volume 59 of *Concepts and Techniques in Modern Geography*. University of East Anglia, 1996.

6 Tim Dwyer, Kim Marriott, and Peter J. Stuckey. Fast node overlap removal. In *Proceedings of the International Symposium on Graph Drawing*, LNCS 3843, pages 153–164, 2005.

7 Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306–316, 2005.

8 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th Annual Symposium on Computational Geometry*, pages 281–288, 1991.

9 Loann Giovannangeli, Frédéric Lalanne, Romain Giot, and Romain Bourqui. Guaranteed visibility in scatterplots with tolerance. *IEEE Transactions on Visualizations and Computer Graphics*, to appear, 2023.

10 Erick Gomez-Nieto, Wallace Casaca, Luis Gustavo Nonato, and Gabriel Taubin. Mixed integer optimization for layout arrangement. In *Proceedings of the Conference on Graphics, Patterns and Images*, pages 115–122, 2013.

11 Daichi Hirono, Hsiang-Yun Wu, Masatoshi Arikawa, and Shigeo Takahashi. Constrained optimization for disoccluding geographic landmarks in 3D urban maps. In *Proceedings of the 2013 IEEE Pacific Visualization Symposium*, pages 17–24, 2013.

12 Kim Marriott, Peter Stuckey, Vincent Tam, and Weiqing He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8(2):143–171, 2003.

13 Wouter Meulemans. Efficient optimal overlap removal: Algorithms and experiments. *Computer Graphics Forum*, 38(3):713–723, 2019.

14 Soeren Nickel, Max Sondag, Wouter Meulemans, Stephen Kobourov, Jaakko Peltonen, and Martin Nöllenburg. Multicriteria optimization for dynamic Demers cartograms. *IEEE Transactions on Visualization and Computer Graphics*, 28(6):2376–2387, 2022.

15 Gabriel Nivasch, János Pach, and Gábor Tardos. The visible perimeter of an arrangement of disks. *Computational Geometry*, 47(1):42–51, 2014.

16 Sheung-Hung Poon, Chan-Su Shin, Tycho Strijk, Takeaki Uno, and Alexander Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2004. doi:10.1007/s00453-003-1063-0.

17 Nadine Schwartges, Jan-Henrik Haunert, Alexander Wolff, and Dennis Zwiebler. Point labeling with sliding labels in interactive maps. In Joaquín Huerta, Sven Schade, and Carlos Granell, editors, *Connecting a Digital Europe Through Location and Place*, pages 295–310. Springer International Publishing, 2014. doi:10.1007/978-3-319-03611-3_17.

18 Hendrik Strobelt, Marc Spicker, Andreas Stoffel, Daniel Keim, and Oliver Deussen. Rolled-out Wordles: A heuristic method for overlap removal of 2D data representatives. *Computer Graphics Forum*, 31(3pt3):1135–1144, 2012.

19 Mereke van Garderen. *Pictures of the Past – Visualization and visual analysis in archaeological context.* PhD thesis, Universität Konstanz, 2018.

20 Mereke van Garderen, Barbara Pampel, Arlind Nocaj, and Ulrik Brandes. Minimum-displacement overlap removal for geo-referenced data visualization. *Computer Graphics Forum*, 36(3):423–433, 2017.

21 Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry*, 13(1):21–47, 1999.

22 Claus O. Wilke. *Fundamentals of data visualization: a primer on making informative and compelling figures.* O'Reilly Media, 2019.

Delaunay Triangulations in the Hilbert Metric

Auguste H. Gezalyan
Department of Computer Science, University of Maryland, College Park, MD, USA

Soo H. Kim
Wellesley College, MA, USA

Carlos Lopez
Montgomery Blair High School, Silver Spring, MD, USA

Daniel Skora
Indiana University, Bloomington, IN, USA

Zofia Stefankovic
Stony Brook University, Stony Brook, NY, USA

David M. Mount
Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

The Hilbert metric is a distance function defined for points lying within the interior of a convex body. It arises in the analysis and processing of convex bodies, machine learning, and quantum information theory. In this paper, we show how to adapt the Euclidean Delaunay triangulation to the Hilbert geometry defined by a convex polygon in the plane. We analyze the geometric properties of the Hilbert Delaunay triangulation, which has some notable differences with respect to the Euclidean case, including the fact that the triangulation does not necessarily cover the convex hull of the point set. We also introduce the notion of a Hilbert ball at infinity, which is a Hilbert metric ball centered on the boundary of the convex polygon. We present a simple randomized incremental algorithm that computes the Hilbert Delaunay triangulation for a set of n points in the Hilbert geometry defined by a convex m-gon. The algorithm runs in $O(n(\log n + \log^3 m))$ expected time. In addition we introduce the notion of the Hilbert hull of a set of points, which we define to be the region covered by their Hilbert Delaunay triangulation. We present an algorithm for computing the Hilbert hull in time $O(nh \log^2 m)$, where h is the number of points on the hull's boundary.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Delaunay Triangulations, Hilbert metric, convexity, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.25

Related Version *Full Version*: https://arxiv.org/abs/2312.05987

1 Introduction

David Hilbert introduced the Hilbert metric in 1895 [15]. Given any convex body Ω in d-dimensional space, the Hilbert metric defines a distance between any pair of points in the interior of Ω (see Section 2 for definitions). The Hilbert metric has a number of useful properties. It is invariant under projective transformations, and straight lines are geodesics. When Ω is an Euclidean ball, it realizes the Cayley-Klein model of hyperbolic geometry. When Ω is a simplex, it provides a natural metric over discrete probability distributions (see, e.g., Nielsen and Sun [21,22]). An excellent resource on Hilbert geometries is the handbook of Hilbert geometry by Papadopoulos and Troyanov [23].

The Hilbert geometry provides new insights into classical questions from convexity theory. Efficient approximation of convex bodies has a wide range of applications, including approximate nearest neighbor searching both in Euclidean space [6] and more general

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 25; pp. 25:1–25:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

metrics [1], optimal construction of ε-kernels [4], solving the closest vector problem approximately [11, 12, 19, 25], and computing approximating polytopes with low combinatorial complexity [3, 5]. These works all share one thing in common – they approximate a convex body by covering it with elements that behave much like metric balls. These covering elements go under various names: Macbeath regions, Macbeath ellipsoids, Dikin ellipsoids, and $(2, \varepsilon)$-covers. Vernicos and Walsh showed that these shapes are, up to a constant scaling factor, equivalent to Hilbert balls [2, 27]. In addition the Hilbert metric behaves nicely in the context flag approximability of convex polytopes as studied by Vernicos and Walsh [28].

Other applications of the Hilbert metric include machine learning [21], quantum information theory [24], real analysis [18], graph embeddings [22], and optimal mass transport [9]. Despite its obvious appeals, only recently has there been any work on developing classical computational geometry algorithms that operate in the Hilbert metric. Nielsen and Shao characterized balls in the Hilbert metric defined by a convex polygon with m sides [20]. Hilbert balls are convex polygons bounded by $2m$ sides. Nielsen and Shao showed that Hilbert balls can be computed in $O(m)$ time, and they developed dynamic software for generating them. Gezalyan and Mount presented an $O(mn \log n)$ time algorithm for computing the Voronoi diagram of n point sites in the Hilbert polygonal metric [13] (see Figure 1(a) and (b)). They showed that the diagram has worst-case combinatorial complexity of $O(mn)$. Bumpus et al. [8] further analyzed the properties of balls in the Hilbert metric and presented software for computing Hilbert Voronoi diagrams.

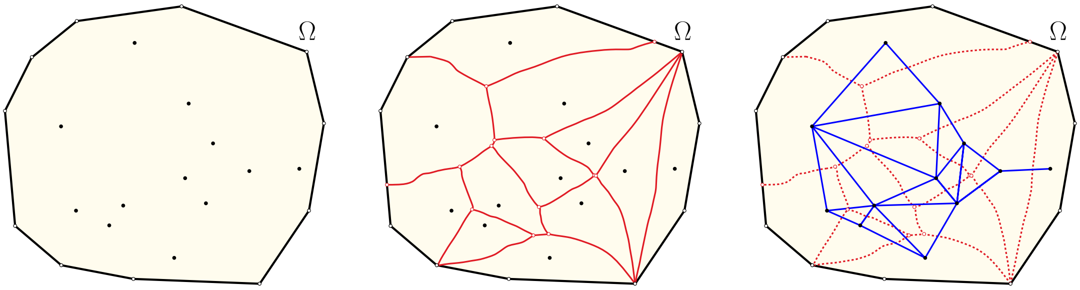


Figure 1 (a) A convex polygon Ω and sites, (b) the Voronoi diagram, and (c) the Delaunay triangulation.

In this paper, we present an algorithm for computing the Delaunay triangulation of a set P of n point sites in the Hilbert geometry defined by an m-sided convex polygon Ω. The Hilbert Delaunay triangulation is defined in the standard manner as the dual of the Hilbert Voronoi diagram (see Figure 1(c)). Our algorithm is randomized and runs in $O(n(\log n + \log^3 m))$ expected time. Excluding the polylogarithmic term in m, this matches the time of the well-known randomized algorithm for Euclidean Delaunay triangulations [14]. It is significantly more efficient than the time to compute the Hilbert Voronoi diagram, which is possible because of the smaller output size. A central element of our algorithm is an $O(\log^3 m)$ time algorithm for computing Hilbert circumcircles.

Unlike the Euclidean case, the Delaunay triangulation does not necessarily triangulate the convex hull of P. We also provide a characterization of when three points admit a Hilbert ball whose boundary contains all three points, and define the Hilbert hull, which we define to be the region covered by the Hilbert Delaunay triangulation. We give an algorithm for the Hilbert hull that runs in time $O(nh \log^2 m)$, where h is the number of points on the Hilbert hull's boundary.

2 Preliminaries

2.1 The Hilbert Metric and Hilbert Balls

The Hilbert metric is defined over the interior of a convex body Ω in $\mathbb{R}^d$ (that is, a closed, bounded, full dimensional convex set). Let $\partial\Omega$ denote Ω's boundary. Unless otherwise stated, we assume throughout that Ω is an m-sided convex polygon in $\mathbb{R}^2$. Given two points p and q in Ω, let $\overline{pq}$ denote the *chord* of Ω defined by the line through these points.

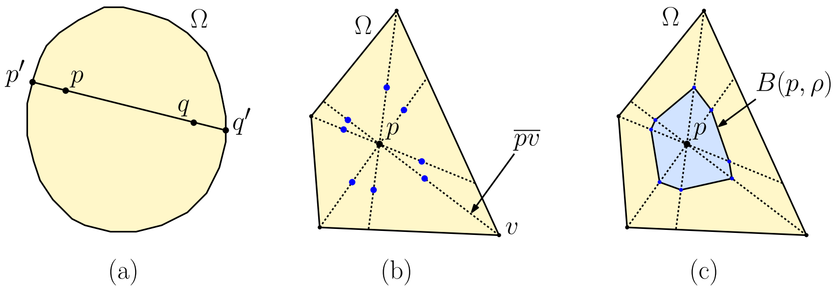


Figure 2 (a) The Hilbert distance $d_\Omega(p,q)$, (b) spokes defined by p, and (c) the Hilbert ball $B(p,\rho)$.

▶ **Definition 1** (Hilbert metric). *Given a bounded convex body Ω in $\mathbb{R}^d$ and two distinct points $p, q \in \mathrm{int}(\Omega)$, let p' and q' denote endpoints of the chord $\overline{pq}$, so that the points are in the order $\langle p', p, q, q'\rangle$. The* Hilbert distance *between p and q is defined*

$$d_\Omega(p,q) \;=\; \frac{1}{2}\ln\left(\frac{\|q-p'\|}{\|p-p'\|}\,\frac{\|p-q'\|}{\|q-q'\|}\right),$$

and define $d_\Omega(p,p) = 0$ (see Figure 2(a)).

Note that the quantity in the logarithm is the cross ratio $(q,p;p',q')$. Since cross ratios are preserved by projective transformations [20], it follows that the Hilbert distances are invariant under projective transformations. Straight lines are geodesics, but not all geodesics are straight lines under the Hilbert distance. The Hilbert distance satisfies all the axioms of a metric, and in particular it is symmetric and the triangle inequality holds [23]. When Ω is a probability simplex [7], this symmetry distinguishes it from other common methods of calculating distances between probability distributions, such as the Kullback-Leibler divergence [17].

Given $p \in \mathrm{int}(\Omega)$ and $\rho > 0$, let $B(p,\rho)$ denote the Hilbert ball of radius ρ centered at p. Nielsen and Shao showed how to compute Hilbert balls [20]. Consider the set of m chords $\overline{pv}$ for each vertex v of Ω (see Figure 2(b)). These are called the *spokes* defined by p. For each spoke $\overline{pv}$, consider the two points at Hilbert distance ρ on either side of p. $B(p,\rho)$ is the (convex) polygon defined by these $2m$ points (see Figure 2(c)). Given any line that intersects Ω, a simple binary search makes it possible to determine the two edges of Ω's boundary intersected by the line. Applying this to the line passing through p and q, it follows that Hilbert distances can be computed in $O(\log m)$ time.

2.2 The Hilbert Voronoi Diagram

Given a set P of n point sites in $\mathrm{int}(\Omega)$, the *Hilbert Voronoi diagram* of P is defined in the standard manner as a subdivision of $\mathrm{int}(\Omega)$ into regions, called *Voronoi cells*, based on which site of P is closest in the Hilbert distance. It was shown by Gezalyan and Mount [13] that each Voronoi cell is star-shaped with respect to its site. Given two sites $p, q \in P$, the (p, q)*-bisector* is the set of points that are equidistant from both sites in the Hilbert metric (see Figure 3(a)).

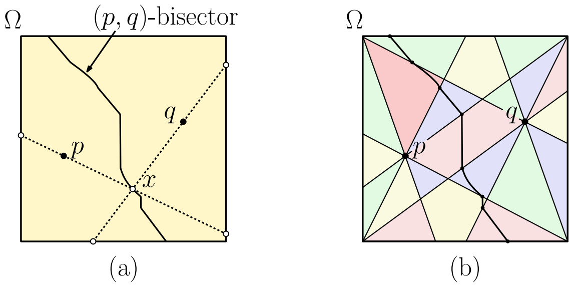


Figure 3 The (p, q)-bisector is a piecewise conic with joints on the spokes of p and q.

The distances from any point x on the (p, q)-bisector to p and q are determined by the edges of $\partial\Omega$ incident to the chords $\overline{px}$ and $\overline{qx}$. We can subdivide the interior Ω into equivalence classes based upon the identity of these edges. It is easy to see that as x crosses a spoke of either p or q, it moves into a different equivalence class. It has been shown that within each equivalence class, the bisector is a conic [8, 13]. It follows that the (p, q)-bisector is a piecewise conic, whose joints lie on sector boundaries (see Figure 3(b)). It is also known that the worst-case combinatorial complexity of the Hilbert Voronoi diagram is $\Theta(mn)$ [13].

2.3 The Hilbert Delaunay Triangulation

The *Hilbert Delaunay triangulation* of P, denoted $\mathrm{DT}(P)$, is defined as the dual of the Hilbert Voronoi diagram, where two sites p and q are connected by an edge if their Voronoi cells are adjacent. Throughout, we make the general-position assumption that no four sites of P are Hilbert equidistant from a single point in $\mathrm{int}(\Omega)$. It is easy to see that familiar concepts from Euclidean Delaunay triangulations apply, but using Hilbert balls rather than Euclidean balls. For example, two sites are adjacent in the triangulation if and only if there exists a Hilbert ball whose boundary contains both sites, and whose interior contains no sites. (The center of this ball lies on the Voronoi edge between the sites.) Also, three points define a triangle if and only if there is a Hilbert ball whose boundary contains all three points, but is otherwise empty.

Consider a triangle $\triangle pqr$ in Ω's interior. A Hilbert ball whose boundary passes through all three points is said to *circumscribe* this triangle. As we shall see in Section 4, some triangles admit no circumscribing Hilbert ball, but the following lemma shows that if it does exist, then it is unique. We refer to the boundary of such a ball as a *Hilbert circumcircle*. (The proof of this lemma, and a number of other lemmas throughout the paper are available in the full version of the paper.)

▶ **Lemma 2.** *There is at most one Hilbert circumcircle for any three non-colinear points in Ω's interior.*

Note that the non-collinear assumption is necessary. To see why, let Ω be a axis aligned rectangle, and let ℓ be a horizontal line through the rectangle's center. It is easy to construct two Hilbert balls, one above the line and one below, whose boundaries contain a segment along this line. Placing the three points within this segment would violate the lemma.

The following lemma shows that $\mathrm{DT}(P)$ is a connected, planar graph. Its proof is similar to the Euclidean case and appears in the full version of the paper.

▶ **Lemma 3.** *Given any discrete set of points P, $\mathrm{DT}(P)$ is a planar (straight-line) graph that spans P.*

Another useful property of the Delaunay triangulation, viewed as a graph, is its relationship to other geometric graph structures. The *Hilbert minimum spanning tree* of a point set P, denoted $\mathrm{MST}_\Omega(P)$, is the spanning tree over P where edge weights are Hilbert distances. The *Hilbert relative neighborhood graph* for a point set P, denoted $\mathrm{RNG}_\Omega(P)$ is a graph over the vertex set P, where two points $p, q \in P$ are joined by an edge if $d_\Omega(p,q) \leq \max(d_\Omega(p,r), d_\Omega(q,r))$, for all $r \in P \setminus \{p,q\}$. Toussaint proved that in the Euclidean metric, the minimum spanning tree is a subgraph of the relative neighborhood graph, which is a subgraph of the Delaunay graph [26]. The following (proved in the full version of the paper) shows that this holds in the Hilbert metric as well. (Since $\mathrm{MST}_\Omega(P)$ is connected, this provides an alternate proof that $\mathrm{DT}(P)$ spans P.)

▶ **Lemma 4.** *Given any discrete set of points P, $MST_\Omega(P) \subseteq RNG_\Omega(P) \subseteq \mathrm{DT}_\Omega(P)$.*

3 Hilbert Bisectors

In this section we present some utilities for processing Hilbert bisectors, which will be used later in our algorithms. We start by recalling a characterization given by Gezalyan and Mount for when a point lies on the Hilbert bisector [13]. Recall that three lines are said to be *concurrent* in projective geometry if they intersect in a common point or are parallel.

▶ **Lemma 5.** *Given a convex body Ω and two points $p, q \in \mathrm{int}(\Omega)$, consider any other point $x \in \Omega$. Let p' and p'' denote the endpoints of the chord $\overline{px}$ so the points appear in the order $\langle p', p, x, p'' \rangle$. Define q' and q'' analogously for $\overline{qx}$.*

(i) *If $x \in \mathrm{int}(\Omega)$, then it lies on the (p,q)-bisector if and only if the lines $\overleftrightarrow{pq}$, $\overleftrightarrow{p'q'}$, and $\overleftrightarrow{p''q''}$ are concurrent (see Figure 4(a)).*

(ii) *If $x \in \partial\Omega$ (implying that $x = p'' = q''$), then x is the limit point of the (p,q)-bisector if and only if there is a supporting line through x concurrent with $\overleftrightarrow{pq}$ and $\overleftrightarrow{p'q'}$ (see Figure 4(b)).*

The following utility lemmas arise as consequences of this characterization. Both involve variants of binary search. Their proofs are given in the full version of the paper.

▶ **Lemma 6.** *Given an m-sided convex polygon Ω, two points $p, q \in \mathrm{int}(\Omega)$, and any ray emanating from p, the point of intersection between the ray and the (p,q)-bisector (if it exists) can be computed in $O(\log^2 m)$ time.*

▶ **Lemma 7.** *Given an m-sided convex polygon Ω and any two points $p, q \in \mathrm{int}(\Omega)$, the endpoints of the (p,q)-bisector on $\partial\Omega$ can be computed in $O(\log^2 m)$ time.*

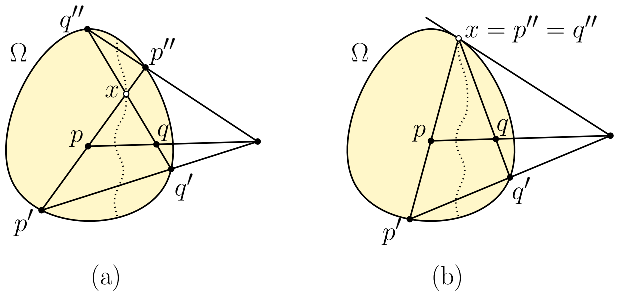


Figure 4 Properties of points on the Hilbert bisector.

4 Hilbert Circumcircles

In the Euclidean plane, any triple of points that are not collinear lie on a unique circle, that is, the boundary of an Euclidean ball. This is not true in the Hilbert geometry, however. Since Delaunay triangulations are based on an empty circumcircle condition, it will be important to characterize when a triangle admits a Hilbert circumcircle and when it does not. In this section we explore the conditions under which such a ball exists.

4.1 Balls at infinity

We begin by introducing the concept of a Hilbert ball centered at a point on the boundary of Ω. Let x be any point on $\partial\Omega$. Given any point $p \in \mathrm{int}(\Omega)$, we are interested in defining the notion of a Hilbert ball centered at x whose boundary contains p. If we think of the points on the boundary of Ω as being infinitely far from any point in $\mathrm{int}(\Omega)$, this gives rise to the notion of a ball at infinity.

Given $x \in \partial\Omega$ and $p \in \mathrm{int}(\Omega)$, let u be any unit vector directed from x into the interior of Ω (see Figure 5(a)). Given any sufficiently small positive δ, let $x_\delta = x + \delta u$ denote the point at Euclidean distance δ from x along vector u, and let $B(x_\delta, p)$ denote the Hilbert ball centered at x_δ of radius $d_\Omega(x_\delta, p)$. The following lemma shows that as δ approaches 0, this ball approaches a shape, which we call the *Hilbert ball at infinity* determined by x and u and passing through p, denoted $B_u(x,p)$ (see Figure 5(b)).

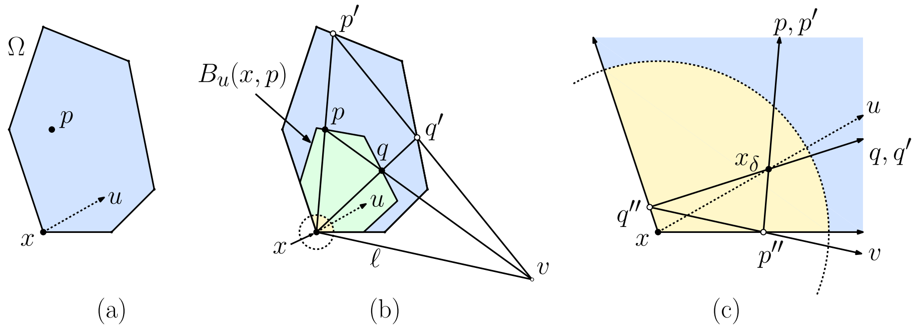


Figure 5 Constructing the Hilbert ball $B_u(x,p)$ at infinity.

▶ **Lemma 8.** *As δ approaches 0, $B(x_\delta, p)$ approaches a convex polygon lying within Ω having x on its boundary.*

A useful utility, which we will apply in Section 7, involves computing the largest empty ball centered at any boundary point x with respect to a point set P. The proof is given in the full version of the paper.

▶ **Lemma 9.** *Given a set of n points P in the interior of Ω, and any point $x \in \partial\Omega$, in $O(n \log m)$ time, it is possible to compute a point $p \in P$, such that there is a ball at infinity centered at x whose boundary passes through p, and which contains no points of P in its interior.*

Balls at infinity are not proper aspects of Hilbert geometry, but they will be convenient for our purposes. Given two points $p, q \in \operatorname{int}(\Omega)$, let x denote the endpoint of the (p, q)-bisector on $\partial\Omega$, oriented so that x lies to the left of the directed line $\overrightarrow{pq}$. (It follows from the star-shapedness of Voronoi cells that the bisector endpoints intersect $\partial\Omega$ on opposite sides of this line.) Let u denote a tangent (if x is a vertex of Ω) vector of the bisector at x. Define $B(p:q) = B_u(x, p)$. Note that this (improper) ball is both centered at and passes through x. In this sense it circumscribes the triangle $\triangle pqx$. Define $B(q:p)$ analogously for the opposite endpoint of this bisector (see Figure 6(a) and (b)).

We can now characterize the set of points r that admit a Hilbert circumcircle with respect to two given points p and q. This characterization is based on two regions, called the *overlap* and *outer regions* (see Figure 6(c)).

▶ **Definition 10** (Overlap/Outer Regions). *Given two points $p, q \in \operatorname{int}(\Omega)$:*

Overlap Region: *denoted $Z(p, q)$, is $B(p:q) \cap B(q:p)$.*

Outer Region: *denoted $W(p, q)$, is $\Omega \setminus (B(p:q) \cup B(q:p))$.*

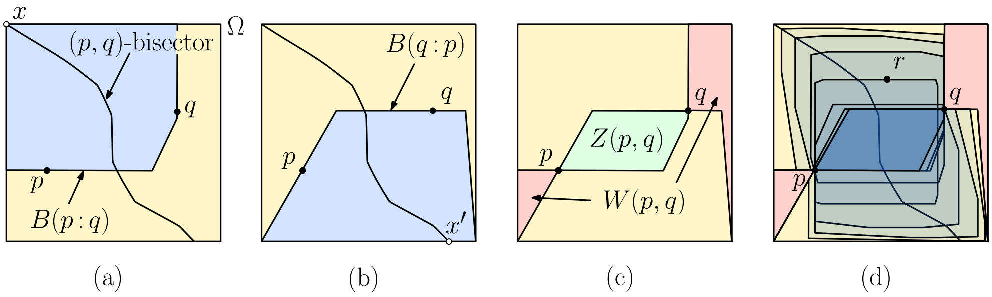


Figure 6 The overlap region $Z(p, q)$ and the outer region $W(p, q)$.

▶ **Lemma 11.** *A triangle $\triangle pqr \subseteq \operatorname{int}(\Omega)$ admits a Hilbert circumcircle if and only if $r \notin Z(p, q) \cup W(p, q)$.*

As shown in Figure 1, the Delaunay triangulation need not cover the convex hull of the set of sites. We refer to the region that is covered as the *Hilbert hull* of the sites. Later, we will present an algorithm for computing the Hilbert hull. The following lemma will be helpful. It establishes a nesting property for the overlap regions.

▶ **Lemma 12.** *If $r \in Z(p, q)$ then $Z(p, r) \subset Z(p, q)$.*

5 Computing Circumcircles

A fundamental primitive in the Euclidean Delaunay triangulation algorithm is the so-called *in-circle test* [14]. Given a triangle $\triangle pqr$ and a fourth site s, the test determines whether s lies within the circumscribing Hilbert ball for the triangle (if such a ball exists). In this section, we present an algorithm which given any three sites either computes the Hilbert circumcircle for these sites, denoted $B(p : q : r)$, or reports that no circumcircle exists. The in-circle test reduces to checking whether $d_\Omega(s, c) < \rho$, which can be done in $O(\log m)$ time as observed in Section 2.

▶ **Lemma 13.** *Given a convex m-sided polygon Ω and triangle $\triangle pqr \subset \mathrm{int}(\Omega)$, in $O(\log^3 m)$ time it is possible to compute $B(p : q : r)$ or to report that no ball exists.*

The remainder of the section is devoted to the proof. The circumscribing ball exists if and only if there is a point equidistant to all three, implying that (p, q)- and (p, r)-bisectors intersect at some point $c \in \mathrm{int}(\Omega)$. The following technical lemma shows that bisectors intersect crosswise.

▶ **Lemma 14.** *Given three non-collinear points $p, q, r \in \mathrm{int}(\Omega)$, the (p, q)- and (p, r)-bisectors have endpoints lying on $\partial\Omega$. If they intersect within $\mathrm{int}(\Omega)$, they intersect transversely in a single point (see Figure 7).*

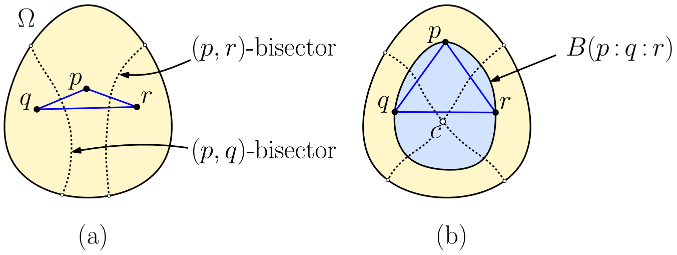


Figure 7 Circumcircles and bisector intersection.

It follows that the (p, q)- and (p, r)-bisectors subdivide the interior of Ω into either three or four regions (three if they do not intersect, and four if they do). We can determine which is this case by invoking Lemma 7 to compute the endpoints of these bisectors in $O(\log^2 m)$ time. If they alternate between (p, q) and (p, r) along the boundary of Ω, then the bisectors intersect, and otherwise they do not. In the latter case, there is no circumcircle for p, q, and r, and hence, no possibility of violating the circumcircle condition. (Note that this effectively provides an $O(\log^2 m)$ test for Lemma 11.) Henceforth, we concentrate on the former case.

Let us assume, without loss of generality, that $\triangle pqr$ is oriented counterclockwise. Let v_q denote the endpoint of the (p, q)-bisector that lies to the right of the oriented line $\overrightarrow{pq}$. Let v_r denote the endpoint of the (p, r)-bisector that lies to the left of the oriented line $\overrightarrow{pr}$ (see Figure 8(a)).

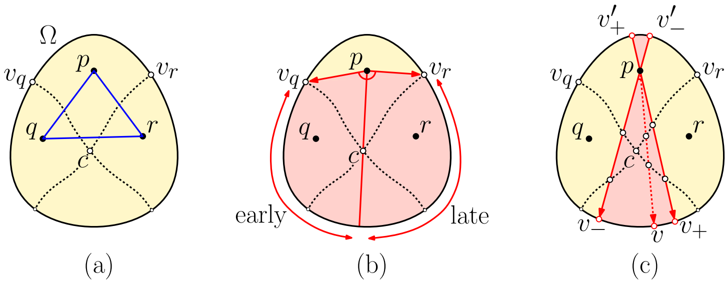


Figure 8 Computing the center of $\triangle pqr$.

Because Voronoi cells are star-shaped, it follows that the vector from p to the desired circumcenter c lies in the counterclockwise angular interval from $\overrightarrow{pv_q}$ to $\overrightarrow{pv_r}$ (see Figure 8(b)). The key to the search is the following observation. In the angular region from $\overrightarrow{pv_q}$ to $\overrightarrow{pc}$, any ray shot from p intersects the (p, q)-bisector before hitting the (p, r)-bisector (if it hits the (p, r)-bisector at all). On the other hand, in the angular region from $\overrightarrow{pc}$ to $\overrightarrow{pv_r}$, any ray shot from p intersects the (p, r)-bisector before hitting the (p, q)-bisector (if it hits the (p, q)-bisector at all). We say that the former type of ray is *early* and the latter type is *late*.

Let v_- and v_+ denote the points on $\partial\Omega$ that bound the current search interval about p (see Figure 8(c)). We will maintain the invariant that the ray $\overrightarrow{pv_-}$ is early and $\overrightarrow{pv_+}$ is late. Initially, $v_- = v_q$ and $v_+ = v_r$. Let v'_- and v'_+ denote the opposite endpoints of the chords $\overline{pv_-}$ and $\overline{pv_+}$. Consider the portion of the boundary of Ω that lies counterclockwise from v_- and v_+. If the angle $\angle v_- p v_+$ is smaller than π, also consider the portion of the boundary of Ω that lies counterclockwise from v'_- and v'_+. With each probe, we sample the median vertex v from whichever of these two boundary portions that contains the larger number of vertices. If v comes from the interval $[v_-, v_+]$, we probe along the ray $\overrightarrow{pv}$, and if it comes from the complementary interval, $[v'_-, v'_+]$, we shoot the ray from p in the opposite direction from v. We then apply Lemma 6 twice to determine in $O(\log^2 m)$ time where this ray hits the (p, q)- and (p, r)-bisectors (if at all). Based on the results, we classify this ray as being early or late and recurse on the appropriate angular subinterval. When the search terminates, we have determined a pair of consecutive spokes about p that contain c.

Because each probe eliminates at least half of the vertices from the larger of the two boundary portions, it follows that at $O(\log m)$ probes, we have located c to within a single pair of consecutive spokes around p. Since each probe takes $O(\log^2 m)$ time, the entire search takes $O(\log^3 m)$ time.

We repeat this process again for r and q. The result is three double wedges defined by consecutive spokes, one about each site. It follows from our earlier remarks from Section 2.2 that within the intersection of these regions, the bisectors are simple conics. We can compute these conics in $O(1)$ time [8] and determine their intersection point, thus yielding the desired point c. The radius ρ of the ball can also be computed in $O(1)$ time.

6 Building the Triangulation

In this section we present our main result, a randomized incremental algorithm for constructing the Delaunay triangulation $\mathrm{DT}(P)$ for a set of n sites P in the interior of an m-sided convex polygon Ω. Our algorithm is loosely based on a well-known randomized incremental algorithm for the Euclidean case [10, 14].

6.1 Orienting and Augmenting the Triangulation

In this section we introduce some representational conventions for the sake of our algorithm. First, we will orient the elements of the triangulation. Given $p, q \in \mathrm{int}(\Omega)$ define the *endpoint* of the (p, q)-bisector to be the endpoint that lies to the left of the directed line $\overrightarrow{pq}$. (It is worth repeating that it follows from the star-shapedness of Voronoi cells that the bisector endpoints intersect $\partial\Omega$ on opposite sides of this line.) The opposite endpoint will be referred to as the (q, p)-bisector endpoint. When referring to a triangle $\triangle pqr$, we will assume that the vertices are given in counterclockwise order. Also, edges in the triangulation are assumed to be directed, so we can unambiguously reference the left and right sides of a directed edge pq.

As mentioned earlier, the triangulation need not cover the convex hull of the set of sites (see Figure 9(a)). For the sake of construction, it will be convenient to augment the triangulation with additional elements, so that all of Ω is covered. First, we add elements to include the endpoints of the Voronoi bisectors on $\partial\Omega$. For each edge pq such that the external face of the triangulation lies to its left, the endpoint of the (p,q)-bisector intersects the boundary of Ω. Letting x denote this boundary point, we add a new triangle $\triangle pqx$, called a *tooth*. (See the green shaded triangles in Figure 9(b).)

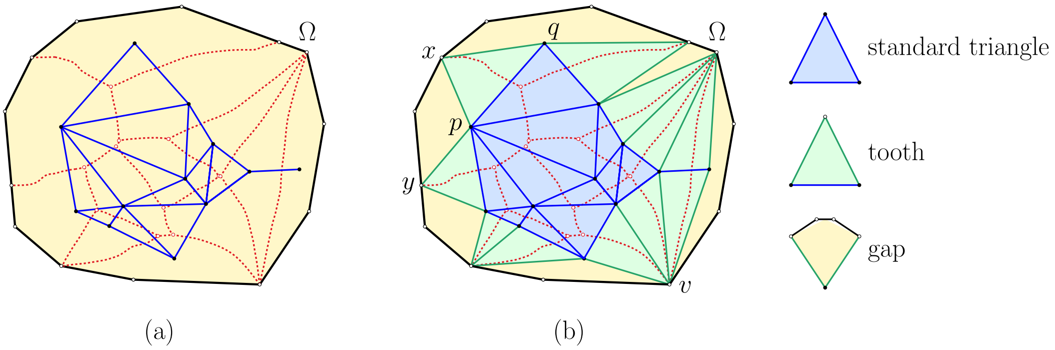


Figure 9 The augmented triangulation.

Finally, the portion of Ω that lies outside of all the standard triangles and teeth consists of a collection of regions, called *gaps*, each of which involves a single site, the sides of two teeth, and a convex polygonal chain along $\partial\Omega$. While these shapes are not triangles, they are defined by three points, and we will abuse the notation $\triangle pxy$ to refer to the gap defined by the site p and boundary points x and y (see Figure 9(b)).

Even when sites are in general position, multiple teeth can share the same boundary vertex. For example, in Figure 9(b) three teeth meet at the same boundary vertex v. In our augmented representation, it will be convenient to treat these as three separate teeth, meeting at three distinct (co-located) vertices, separated by two degenerate (zero-width) gaps. This allows us to conceptualize the region outside of the standard triangulation as consisting of an alternating sequence of teeth and gaps. The following lemma summarizes a few useful facts about the augmented representation. The proof is straightforward and appears in the full version of the paper.

▶ **Lemma 15.** *Given a set of n point sites in the interior of a convex polygon Ω:*

- *The augmented Delaunay triangulation has complexity $O(n)$.*
- *The region covered by standard triangles is connected.*
- *Each standard triangle and each tooth satisfies the empty circumcircle property.*
- *The region outside the standard triangles consists of an alternating sequence of teeth and gaps.*
- *For any $p \in \Omega$, membership in any given triangle, tooth, or gap can be determined in $O(1)$ time.*

6.2 Local and Global Delaunay

Given a set P of point sites, we say that an augmented triangulation $\mathcal{T}(P)$ is *globally Delaunay* (or simply *Delaunay*) if for each standard triangle and each tooth $\triangle pqr$, the circumscribing ball, denoted $B(p:q:r)$, exists and has no site within its interior. The incremental Euclidean Delaunay triangulation algorithm employs a local condition, which only checks this for neighboring triangles [10, 14]. Given a directed edge pq of the triangulation that is incident to two triangles or to a triangle and tooth, let a and b be the vertices of the incident triangles lying to the left and right of the pq, respectively. We say that $\mathcal{T}(P)$ is *locally Delaunay* if

$a \notin \text{int}(B(q:p:b))$ and $b \notin \text{int}(B(p:q:a))$ for all such edges. The following lemma shows that it suffices to certify the Delaunay properties locally. The proof, which appears in the full version of the paper, follows the same structure as the Euclidean case, but additional care is needed due to the existence of teeth and gaps.

▶ **Lemma 16.** *Given a set P of point sites, an augmented triangulation $\mathcal{T}(P)$ is globally Delaunay if and only if it is locally Delaunay.*

6.3 Incremental Construction

The algorithm operates by first randomly permuting the sites. It begins by inserting two arbitrary sites a and b and generating the resulting augmented triangulation. In $O(\log^2 m)$ time, we can compute the endpoints of the (a, b)-bisector. We add the edge ab and create two teeth by connecting a and b to the bisector endpoints (see Figure 10(a)). We then insert the remaining points one by one, updating the triangulation incrementally after each insertion (see Algorithm 1). Later, we will discuss how to determine which element each new site lies in, but for now, let us focus on how the triangulation is updated.

Algorithm 1 Constructs the Hilbert Delaunay triangulation of a point set P.

```
procedure DELAUNAY($P$)                          ▷ Build the Delaunay triangulation of point set $P$
    $\mathcal{T} \leftarrow$ empty triangulation
    Randomly permute $P$
    $a, b \leftarrow$ any two points of $P$                    ▷ Initialize with sites $a$ and $b$
    $x, y \leftarrow$ endpoints of the $(a, b)$-bisector                ▷ See Figure 10(a)
    Add edges $ab$, $ax$, $ay$, $bx$, $by$ to $\mathcal{T}$
    for all $p \in P \setminus \{a, b\}$ do                      ▷ Add all remaining sites
        INSERT($p, \mathcal{T}$)
    end for
    return $\mathcal{T}$
end procedure
```

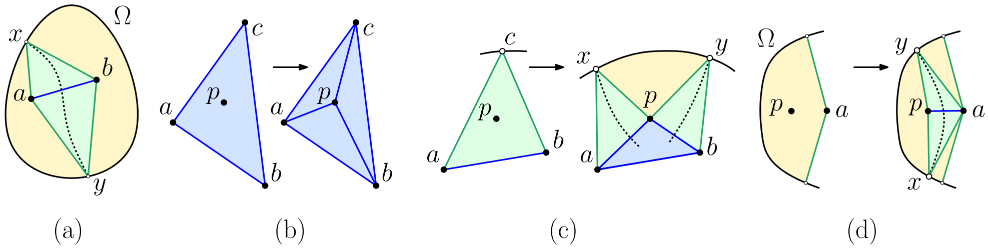


Figure 10 (a) Initialization and insertion into a (b) standard triangle, (c) tooth, (d) gap.

When we insert a point, there are three possible cases, depending on the type of element that contains the point, a standard triangle (three sites), a tooth (two sites), or a gap (one site). The case for standard triangles is the same as for Euclidean Delaunay triangulations [14], involving connecting the new site to the triangle's vertices (see Figure 10(b)). Insertion into a tooth involves removing the boundary vertex, connecting the new site to the two existing site vertices, and creating two new teeth based on the bisectors to these sites (see Figure 10(c)). Finally, insertion into a gap involves connecting the new site to the existing site vertex, and creating two new teeth based on the bisectors to this site (see Figure 10(d)).

Algorithm 2 Site insertion.

```
procedure Insert($p, \mathcal{T}$)                    ▷ Insert a new site $p$ into triangulation $\mathcal{T}$
    $\triangle abc \leftarrow$ the triangle of $\mathcal{T}$ containing $p$
    if $\triangle abc$ is a standard triangle then                        ▷ See Figure 10(b)
        Add edges $ap$, $bp$, $cp$ to $\mathcal{T}$
        FlipEdge($ab, p, \mathcal{T}$);  FlipEdge($bc, p, \mathcal{T}$);  FlipEdge($ca, p, \mathcal{T}$)
    else if $\triangle abc$ is a tooth then                               ▷ See Figure 10(c)
        Let $a$ and $b$ be sites, and $c$ be on boundary
        $x, y \leftarrow$ endpoints of the $(a, p)$- and $(p, b)$-bisectors, respectively
        Remove $c$ and edges $ca$ and $cb$ from $\mathcal{T}$
        Add edges $ap$, $bp$, $ax$, $px$, $by$, $py$ to $\mathcal{T}$
        FlipEdge($ab, p, \mathcal{T}$);
        FixTooth($\triangle apx, p, \mathcal{T}$);  FixTooth($\triangle pby, p, \mathcal{T}$)
    else                                                ▷ $\triangle abc$ is a gap; see Figure 10(d)
        Let $a$ be the site, and let $b$ and $c$ be on the boundary
        $x, y \leftarrow$ endpoints of $(a, p)$- and $(p, a)$-bisectors, respectively
        Add edges $ap$, $ax$, $px$, $ay$, $py$ to $\mathcal{T}$
        FixTooth($\triangle apx, p, \mathcal{T}$);   FixTooth($\triangle pay, p, \mathcal{T}$)
    end if
end procedure
```

On return from Insert, $\mathcal{T}$ is a topologically valid augmented triangulation, but it may fail to satisfy the Delaunay empty circumcircle conditions, and may even fail to be geometrically valid. The procedures FlipEdge and FixTooth repair any potential violations of the local Delaunay conditions.

The procedure FlipEdge is applied to all newly generated standard triangles $\triangle pab$. It is given the directed edge ab of the triangle, such that p lies to the left of this edge. It accesses the triangle $\triangle abc$ lying to the edge's right. This may be a standard triangle or a tooth. In either case, it has an associated circumcircle, which we assume has already been computed.[1] We test whether p encroaches on this circumcircle, and if so, we remove the edge ab. If $\triangle abc$ is a standard triangle, then we complete the edge-flip by adding edge pc, and then continue by checking the edges ac and cb (see Figure 11(a)). Otherwise, we remove vertex c, and compute the endpoints x and y of the (p, a)- and (b, p)-bisectors, respectively. We create two new teeth, $\triangle pax$ and $\triangle bpy$ (see Figure 11(b)). This creates a new (possibly degenerate) gap $\triangle pxy$. Later, we will show (see Lemma 17) that no further updates are needed. The algorithm is presented in Algorithm 3.

The "else" clause creates two teeth $\triangle pax$ and $\triangle bpy$. The following lemma (proved in the full version of the paper) shows that there is no need to apply FixTooth to them.

▶ **Lemma 17.** *On return from FlipEdge the teeth $\triangle pax$ and $\triangle bpy$ generated in the "else" clause are both valid.*

Finally, we present FixTooth. To understand the issue involved, consider Figure 10(d). In the figure, the newly created teeth $\triangle pax$ and $\triangle apy$ both lie entirely inside the existing gap. But, this need not generally be the case, and the newly created boundary vertex may

[1] In the Euclidean case, the in-circle test may be run from either $\triangle abc$ or $\triangle pab$, but this is not true for the Hilbert geometry. In light of Lemma 11, we do not know whether the $\triangle pab$ has a circumcircle, so we run the test from $\triangle abc$.

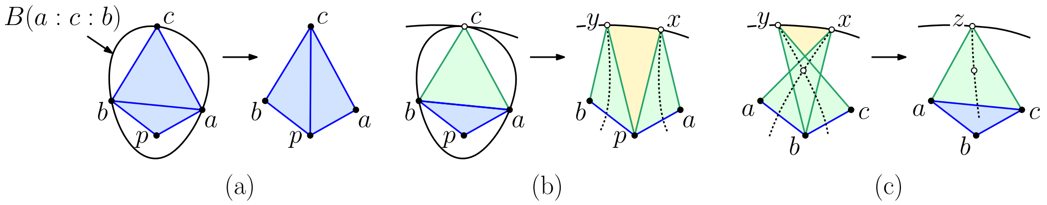


Figure 11 (a) Standard-triangle edge flip, (b) tooth edge flip, and (c) fixing a tooth.

Algorithm 3 In-circle test and edge flip.

```
procedure FlipEdge($ab, p, \mathcal{T}$)          ▷ In-circle test. New site $p$ lies to the left of edge $ab$.
    $c \leftarrow$ vertex of triangle to right of $ab$ in $\mathcal{T}$
    if $p \in B(a:c:b)$ then                       ▷ $p$ fails the in-circle test for $\triangle acb$
        Remove edge $ab$ from $\mathcal{T}$
        if $\triangle acb$ is a standard triangle then          ▷ See Figure 11(a)
            Add edge $pc$ to $\mathcal{T}$
            FlipEdge($ac, p, \mathcal{T}$);   FlipEdge($cb, p, \mathcal{T}$)      ▷ Create $\triangle pac$ and $\triangle bpc$
        else                                       ▷ $\triangle acb$ is a tooth. See Figure 11(b)
            $x, y \leftarrow$ endpoints of $(p, a)$- and $(b, p)$-bisectors, respectively
            Remove $c$ and edges $cb$ and $ca$ from $\mathcal{T}$
            Add edges $ax$, $px$, $by$, $py$ to $\mathcal{T}$       ▷ Create teeth $\triangle pax$ and $\triangle bpy$
        end if
    end if
end procedure
```

lie outside the current gap, resulting in two or more teeth that overlap each other (see, e.g., Figure 11(c)). The procedure FixTooth is given a tooth $\triangle abx$, where a and b are sites, and x is on Ω's boundary. Whenever it is invoked the new site p is either a or b. Recalling our assumption that teeth and gaps alternate around the boundary of Ω, we check the teeth that are expected to be adjacent to the current tooth on the clockwise and counterclockwise sides. (Only the clockwise case is presented in the algorithm, but the other case is symmetrical, with a and b swapped.)

Let $\triangle bcy$ denote the tooth immediately clockwise around the boundary. We test whether these triangles overlap (see Figure 11(c)). If so, we know that the (a, b)-bisector (which ends at x) and the (b, c)-bisector (which ends at y) must intersect. This intersection point is the center of a Hilbert ball circumscribing $\triangle abc$. We replace the two teeth $\triangle abx$ and $\triangle bcy$ with the standard triangle $\triangle abc$ and the tooth $\triangle acz$, where z is the endpoint of the (a, c)-bisector. If $p = a$, then we invoke FlipEdge on the opposite edge bc from p, and we invoke FixTooth on the newly created tooth. When the algorithm terminates, all the newly generated elements have been locally validated. The algorithm is presented in Algorithm 4.

Based on our earlier remarks, it follows that our algorithm correctly inserts a site into the augmented triangulation.

▶ **Lemma 18.** *Given an augmented triangulation $\mathcal{T}(P)$ for a set of sites P, the procedure* Insert *correctly inserts a new site p, resulting in the augmented Delaunay triangulation for of $P \cup \{p\}$.*

The final issue is the algorithm's expected running time. Our analysis follows directly from the analysis of the randomized incremental algorithm for the Euclidean Delaunay triangulation. We can determine which triangle contains each newly inserted point in

Algorithm 4 Check for and fix overlapping teeth.

```
procedure FixTooth($\triangle abx, p, \mathcal{T}$)                    ▷ Fix tooth $\triangle abx$ where $p = a$ or $p = b$
    Let $\triangle bcy$ be the tooth clockwise adjacent to $\triangle abx$
    if $\triangle abx$ overlaps $\triangle bcy$ then                          ▷ See Figure 11(c)
        $z \leftarrow$ endpoint of the $(a, c)$-bisector
        Remove $x$ and $y$ and edges $ax$, $bx$, $by$, and $cy$ from $\mathcal{T}$
        Add edges $ac$, $az$, and $cz$ to $\mathcal{T}$         ▷ Create triangle $\triangle abc$ and tooth $\triangle acz$
        if $p = a$ then
            FlipEdge($bc, p, \mathcal{T}$)              ▷ Check edge flip with triangle opposite $bc$
            FixTooth($\triangle acz, p, \mathcal{T}$)                    ▷ Check for further overlaps
        end if
    else
        Repeat the above for the triangle counterclockwise from $\triangle abx$ swapping $a \leftrightarrow b$
    end if
end procedure
```

amortized $O(\log n)$ expected time either by building a history-based point location data structure (as with Guibas, Knuth, and Sharir [14]) or by bucketing sites (as with de Berg et al. [10]). The analysis in the Euclidean case is based on a small number of key facts, which apply in our context as well. First, sites are inserted in random order. Second, the conflict set for any triangle or tooth consists of the points lying in the triangle's circumcircle. Both of these clearly hold in the Hilbert setting. Third, the number of structural updates induced by the insertion of site p is proportional to the degree of p following the insertion. This holds for our algorithm, because each modification to the triangulation induced by p's insertion results in a new edge being added to p (and these edges are not deleted until future insertions). Fourth, the structure is invariant to the insertion order. Finally, the triangulation graph is planar, and hence it has constant average degree. The principal difference is that the in-circle test involves computing a Hilbert circumcircle, which by Lemma 13 can be performed in $O(\log^3 m)$ time. These additional factors of $O(\log n)$ and $O(\log^3 m)$ are performed a constant number of times in expectation, for each of the n insertions. This implies our main result.

▶ **Theorem 19.** *Given a set of n points in the Hilbert geometry defined by a convex m-gon Ω, it is possible to construct the augmented Delaunay triangulation in randomized expected time $O(n(\log n + \log^3 m))$.*

7 The Hilbert Hull

As observed earlier, the triangles of the Delaunay triangulation of P do not necessarily cover the convex hull of P. The region covered by these triangles is called the *Hilbert hull* (the blue region of Figure 9). In this section, we present a simple algorithm for computing this hull for a set of n points in the Hilbert distance defined by a convex m-gon Ω as an alternative to deriving it from the Delaunay triangulation of the point-set. Our approach is roughly based on the Jarvis march algorithm [16] for computing convex hulls.

The standard Jarvis march maintains two consecutive sites on the hull, and it iteratively computes the next point in $O(n)$ time using an angular ordering induced by these two points. Our algorithm will instead maintain a current site p on the hull, and a boundary point x, such that there is an empty ball at infinity centered at x whose boundary passes through p. It uses x and p to induce an angular order to select the next point.

To start the process off, we need to identify a pair (p_0, x_0), where $p_0 \in P$, $x_0 \in \partial\Omega$, and p_0 lies on the boundary of an empty ball centered at x_0. By Lemma 9, such a pair can be computed in $O(n \log m)$ time. Assuming that the algorithm has generated an initial sequence of points on the hull, arriving at a pair (p_i, x_i) satisfying the above invariant, we compute the next pair as follows. First, we take r to be the point of P that minimizes the counterclockwise angle $\angle x_i p_i r$. Such a point has not already been added to the hull.

We then enumerate the points of $P \setminus \{p_i, r\}$ to see whether any lie in the overlap region $Z(p, r)$. Whenever we encounter such a point, we set r to this point and continue the process. After considering all the points of P, it follows from nesting properties of overlap regions (see Lemma 12) that $Z(p, r)$ is empty, and hence the (r, p)-bisector extends to the boundary of Ω in the Voronoi diagram of P. Thus, we take $p_{i+1} \leftarrow r$, and x_{i+1} is the bisector endpoint. From the remarks made after Lemma 14, we can test membership in the $Z(p, r)$ in $O(\log^2 m)$ time, and by Lemma 7, we can compute the endpoint on the bisector in $O(\log^2 m)$ time as well. Since we consider at most n points, it takes a total of $O(n \log^2 m)$ time to generate one more point on the Hilbert hull. This implies that the overall running time is $O(nh \log^2 m)$.

▶ **Lemma 20.** *Given a set of n points in the Hilbert geometry defined by a convex m-gon Ω, it is possible to construct the Hilbert hull in time $O(nh \log^2 m)$, where h is the number of points on the hull's boundary.*

8 Concluding Remarks

In this paper we presented an algorithm for computing the Delaunay triangulation of a set P of n point sites in the Hilbert geometry defined by an m-sided convex polygon Ω in expected $O(n(\log n + \log^3 m)$ time. Additionally we presented an algorithm for the Hilbert Hull, the boundary of the Hilbert Delaunay triangulation, in time $O(nh \log^2 m)$, where h is the number of points on the boundary. Supporting results, such as the algorithm for determining if it exists and computing the circumscribing ball of three non-collinear points in the interior of Ω in $O(\log^3 m)$ time, and the characterization of Hilbert balls at infinity, may be useful for further algorithmic results in the Hilbert metric. As discussed in the introduction, the Hilbert metric has a large variety of applications including convex approximation [2] [28], machine learning [21], quantum information theory [24], real analysis [18], graph embeddings [22], and optimal mass transport [9]. Extensions of algorithmic results from Euclidean to Hilbert geometry may prove of use to researchers in these fields.

References

1 Ahmed Abdelkader, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Approximate nearest neighbor searching with non-Euclidean and weighted distances. In *Proc. 30th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 355–372, 2019. doi:10.1137/1.9781611975482.23.

2 Ahmed Abdelkader and David M. Mount. Economical Delone sets for approximating convex bodies. In *Proc. 16th Scand. Workshop Algorithm Theory*, pages 4:1–4:12, 2018. doi:10.4230/LIPIcs.SWAT.2018.4.

3 Rahul Arya, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. *ACM Trans. Algorithms*, 18:1–29, 2022. doi:10.1145/3559106.

4 Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Near-optimal ε-kernel construction and related problems. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 10:1–15, 2017. doi:10.4230/LIPIcs.SoCG.2017.10.

5 Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete Comput. Geom.*, 58(4):849–870, 2017. doi:10.1007/s00454-016-9856-5.

6 Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Approximate polytope membership queries. *SIAM J. Comput.*, 47(1):1–51, 2018. doi:10.1137/16M1061096.

7 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi:10.1017/CBO9780511804441.

8 Madeline Bumpus, Caesar Dai, Auguste H. Gezalyan, Sam Munoz, Renita Santhoshkumar, Songyu Ye, and David M. Mount. Software and analysis for dynamic Voronoi diagrams in the Hilbert metric, 2023. arXiv:2304.02745.

9 Yongxin Chen, Tryphon Georgiou, and Michele Pavon. Entropic and displacement interpolation: A computational approach using the Hilbert metric. *SIAM J. Appl. Math.*, 76:2375–2396, 2016. doi:10.1137/16M1061382.

10 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2010. doi:10.1007/978-3-540-77974-2.

11 Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. Covering cubes and the closest vector problem. In *Proc. 27th Annu. Sympos. Comput. Geom.*, pages 417–423, 2011. doi:10.1145/1998196.1998264.

12 Friedrich Eisenbrand and Moritz Venzin. Approximate CVPs in time $2^{0.802n}$. *J. Comput. Sys. Sci.*, 124:129–139, 2021. doi:10.1016/j.jcss.2021.09.006.

13 Auguste H Gezalyan and David M Mount. Voronoi diagrams in the hilbert metric. In *39th International Symposium on Computational Geometry (SoCG 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

14 Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992. doi:10.1007/BF01758770.

15 D. Hilbert. Ueber die gerade Linie als kürzeste Verbindung zweier Punkte. *Math. Annalen*, 46:91–96, 1895. doi:10.1007/BF02096204.

16 Ray A Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.

17 S. Kullback and R. A. Leibler. On information and sufficiency. *Annals. Math. Stat.*, 22:79–86, 1951. doi:10.1214/aoms/1177729694.

18 Bas Lemmens and Roger Nussbaum. Birkhoff's version of Hilbert's metric and its applications in analysis, 2013. arXiv:1304.7921.

19 Márton Naszódi and Moritz Venzin. Covering convex bodies and the closest vector problem. *Discrete Comput. Geom.*, 67:1191–1210, 2022. doi:10.1007/s00454-022-00392-x.

20 Frank Nielsen and Laetitia Shao. On balls in a Hilbert polygonal geometry. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 67:1–67:4, 2017. (Multimedia contribution). doi:10.4230/LIPIcs.SoCG.2017.67.

21 Frank Nielsen and Ke Sun. Clustering in Hilbert's projective geometry: The case studies of the probability simplex and the elliptope of correlation matrices. In Frank Nielsen, editor, *Geometric Structures of Information*, pages 297–331. Springer Internat. Pub., 2019. doi:10.1007/978-3-030-02520-5_11.

22 Frank Nielsen and Ke Sun. Non-linear embeddings in Hilbert simplex geometry, 2022. arXiv:2203.11434.

23 Athanase Papadopoulos and Marc Troyanov. *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*. European Mathematical Society Publishing House, 2014. doi:10.4171/147.

24 David Reeb, Michael J. Kastoryano, and Michael M. Wolf. Hilbert's projective metric in quantum information theory. *J. Math. Physics*, 52(8), 2011. doi:10.1063/1.3615729.

25 Thomas Rothvoss and Moritz Venzin. Approximate CVP in time $2^{0.802n}$ – Now in any norm! In *Proc. 23rd Internat. Conf. on Integ. Prog. and Comb. Opt. (IPCO 2022)*, pages 440–453, 2022. doi:10.1007/978-3-031-06901-7_33.

26 Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recogn.*, 12:261–268, 1980. doi:10.1016/0031-3203(80)90066-7.

27 Constantin Vernicos. On the Hilbert geometry of convex polytopes. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 111–126. European Mathematical Society Publishing House, 2014. doi:10.48550/arXiv.1406.0733.

28 Constantin Vernicos and Cormac Walsh. Flag-approximability of convex bodies and volume growth of Hilbert geometries, 2018. arXiv:1809.09471.

No-Dimensional Tverberg Partitions Revisited

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA

Eliot W. Robson
Department of Computer Science, University of Illinois, Urbana, IL, USA

Abstract

Given a set $P \subset \mathbb{R}^d$ of n points, with diameter Δ, and a parameter $\delta \in (0,1)$, it is known that there is a partition of P into sets $P_1, \dots, P_t$, each of size $O(1/\delta^2)$, such that their convex hulls all intersect a common ball of radius $\delta\Delta$. We prove that a random partition, with a simple alteration step, yields the desired partition, resulting in a (randomized) linear time algorithm (i.e., $O(dn)$). We also provide a deterministic algorithm with running time $O(dn \log n)$. Previous proofs were either existential (i.e., at least exponential time), or required much bigger sets. In addition, the algorithm and its proof of correctness are significantly simpler than previous work, and the constants are slightly better.

We also include a number of applications and extensions using the same central ideas. For example, we provide a linear time algorithm for computing a "fuzzy" centerpoint, and prove a no-dimensional weak ε-net theorem with an improved constant.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Points, partitions, convex hull, high dimension

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.26

Related Version *Full Version*: `https://arxiv.org/abs/2306.01678` [12]

Funding Partially supported by NSF AF award CCF-2317241.

Acknowledgements The authors thank Ken Clarkson and Sandeep Sen for useful discussions.

1 Introduction

Centerpoints

A point c is an *α-centerpoint* of a set $P \subseteq \mathbb{R}^d$ of n points, if all closed halfspaces containing c also contain at least αn points of P. The parameter α is the *centrality* of c, while αn is its **Tukey depth**. The centerpoint theorem [17], which is a consequence of Helly's theorem [14], states that a $1/(d+1)$-centerpoint (denoted $\overline{\mathsf{c}}_P$) always exists.

In two dimensions, Jadhav and Mukhopadhyay [16] presented an $O(n)$ time algorithm for computing a $1/3$-centerpoint (but not the point of maximum Tukey depth). Chan et al. [4] presented an $O(n \log n + n^{d-1})$ algorithm for computing the point of maximum Tukey depth (and thus also a $1/(d+1)$-centerpoint). It is believed that $\Omega(n^{d-1})$ is a lower bound on solving this problem exactly, see [4] for details and history.

This guarantee of $1/(d+1)$-centerpoint is tight, as demonstrated by placing the points of P in $d+1$ small, equal size clusters (mimicking weighted points) in the vicinity of the vertices of a simplex. Furthermore, the lower-bound of $\lceil n/(d+1) \rceil$ is all but meaningless if d is as large as $n-1$.

Approximating centrality

A randomized $\widetilde{O}(d^9)$ time algorithm for computing a (roughly) $1/(4d^2)$ centerpoint was presented by Clarkson et al. [9], and a later refinement by Har-Peled and Jones [11] improved this algorithm to compute a (roughly) $1/d^2$ centerpoint in $\widetilde{O}(d^7)$ time, where $\widetilde{O}$ hides polylog terms. Miller and Sheehy [19] derandomized the algorithm of Clarkson et al., computing a

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 26; pp. 26:1–26:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\Omega(1/d^2)$ centerpoint in time $n^{O(\log d)}$. Developing an algorithm that computes a $1/(d+1)$-centerpoint in polynomial time (in d) in still open, although the existence of such an algorithm with running time better than $\Omega(n^{d-1})$ seems unlikely, as mentioned above.

Tverberg partitions

Consider a set P of n points in $\mathbb{R}^d$. Tverberg's theorem states that such a set can be partitioned into $k = \lfloor n/(d+1) \rfloor$ subsets, such that all of their convex-hulls intersect. Specifically, a point in this common intersection is a $1/(d+1)$-centerpoint. Indeed, a point p contained in the convex-hulls of the k sets of the partition is a k/n-centerpoint, as any halfspace containing p must also contain at least one point from each of these k subsets. Refer to the surveys [10] and [3] for information on this and related theorems.

This theorem has an algorithmic proof [20], but its running time is $n^{O(d^2)}$. To understand the challenge in getting an efficient algorithm for this problem, observe that it is not known, in strongly polynomial time, to decide if a point is inside the convex-hull of a point set (i.e., is it $1/n$-centerpoint?). Similarly, for a given point p, it is not known how to compute, in weakly or strongly polynomial time, the centrality of p. Nevertheless, a Tverberg partition is quite attractive, as the partition itself (and its size) provides a compact proof (i.e., lower bound) of its centrality. If the convex-combination realization of p inside each of these sets is given, then its k/n-centrality can be verified in linear time.

There has been significant work trying to compute Tverberg partitions with as many sets as possible while keeping the running time polynomial. The best polynomial algorithms currently known (roughly) match the bounds for the approximate centerpoint mentioned above. Specifically, it is known how to compute a Tverberg partition of size $O\big(n/(d^2 \log d)\big)$ (along with a point in the common intersection) in weakly polynomial time. See [13] and references therein.

No-dimensional Tverberg theorem

Adiprasito et al. [1] proved a no-dimensional variant of Tverberg's theorem. Specifically, for $\delta \in (0,1)$, they showed that one can partition a point set P into sets of size $O(1/\delta^2)$, such that the convex-hulls of the sets intersect a common ball of radius $\delta \operatorname{diam}(P)$. Their result is existential and does not yield an efficient algorithm. However, as the name suggests, it has the attractive feature that the sets in the partition have size that does not depend on the dimension.

Choudhary and Mulzer [7] gave a weaker version of this theorem, but with an efficient algorithm. Speculatively, given a set $P \subset \mathbb{R}^d$ of n points, and a parameter $\delta \in (0,1)$, P can be partitioned, in $O(nd \log k)$ time, into $k = O(\delta\sqrt{n})$ sets $P_1, \ldots, P_k$, each of size $\Theta(\sqrt{n}/\delta)$, such that there is a ball of radius $\delta \operatorname{diam}(P)$ that intersects the convex-hull of P_i for every i. Note that the later (algorithmic) result is significantly weaker than the previous (existential) result, as the subsets have to be substantially larger.

Thus, the question remains: Can one compute a no-dimensional Tverberg partition with the parameters of Adiprasito et al. [1] in linear time?

Centerball via Tverberg partition

As observed by Adiprasito et al. [1], a no-dimensional Tverberg partition readily implies a no-dimensional centerpoint result, where the central point is replaced by a ball. Specifically, they showed that one can compute a ball of radius $\delta \operatorname{diam}(P)$ such that any halfspace containing it contains $\Omega(\delta^2 n)$ points of P.

Centroid and sampling

The **centroid** of a point set P is the point $\overline{\mathsf{m}}_P = \sum_{p \in P} p / |P|$. The 1-**mean** price of clustering P, using q, is the sum of squared distances of the points of P to q, that is $f(q) = \sum_{p \in P} \|p - q\|^2$. It is not hard to verify that f is minimized at the centroid $\overline{\mathsf{m}}_P$. A classical observation of Inaba et al. [15] is that a sample R of size $O(1/\delta^2)$ of points from P is δ-close to the global centroid of the point set. That is, $\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_R\| \leq \delta \mathrm{diam}(P)$ with constant probability. Applications of this observation to k-means clustering and sparse coresets are well known, see Clarkson [8, Section 2.4] and references therein.

Our results

We show that the aforementioned observation of Inaba et al. implies the no-dimensional Tverberg partition. Informally, for a random partition of P into sets of size $O(1/\delta^2)$, most of the sets are in distance at most $\delta \mathrm{diam}(P)$ from the global centroid of P. By folding the far sets (i.e., "bad"), into the close sets (i.e., "good"), we obtain the desired partition. The resulting algorithm has (expected) linear running time $O(dn)$.

For the sake of completeness, we prove the specific form of the 1-mean sampling observation [15] we need in Lemma 3 – the proof requires slightly tedious but straightforward calculations. The linear time algorithm for computing the no-dimensional Tverberg partition is presented in Theorem 6, which is the main result of this paper.

In the other extreme, one wants to split the point set into two sets of equal size while minimizing their distance. We show that a set P with $2n$ points can be split (in linear time) into two sets of size n, such that (informally) the expected distance of their centroids is $\leq \mathrm{diam}(P)/\sqrt{n}$. The proof of this is even simpler (!), and the bound is tight; see Lemma 9. We present several applications:

(I) No-dimensional Centerball. In Section 3.1, we present a no-dimensional generalization of the centerpoint theorem. As mentioned above, this was already observed by Adiprasito et al. [1], but our version can be computed efficiently.

(II) Weak ε-net. A new proof of the no-dimensional version of the weak ε-net theorem with improved constants, see Section 3.2.

(III) Derandomization. The sampling mean lemma (i.e., Lemma 3) can be derandomized to yield a linear time algorithm, see Lemma 16. The somewhat slower version, Lemma 15, is a nice example of using conditional expectations for derandomization. Similarly, the halving scheme of Lemma 9 can be derandomized in a fashion similar to discrepancy algorithms [18, 5]. The derandomized algorithm, presented in Lemma 17, has linear running time $O(dn)$.

This leads to a deterministic $O(dn \log n)$ time algorithm for the no-dimensional Tverberg partition, see Lemma 18. The idea is to repeatedly apply the halving scheme, in a binary tree fashion, till the point set is partitioned into subsets of size $O(1/\delta^2)$. Both the running time and constants are somewhat worse than the randomized algorithm of Theorem 6, but it is conceptually even simpler, avoiding the need for an alteration step.

As an extra, another neat implication of the observation of Inaba et al. [15] is the dimension free version of Carathéodory's theorem [17], which we present in the full version.

Simplicity

While simplicity is in the eyes of the beholder, the authors find the brevity of the results here striking compared to previous work. In particular, our presentation here is longer than strictly necessary, as we reproduce proofs of previous known results, such as Lemma 3 and its variant Lemma 9, so our work is self contained.

2 Approximate Tverberg partition via mean sampling

In the following, for two points $p, q \in \mathbb{R}^d$, let $pq = \langle p, q\rangle = \sum_{i=1}^{d} p[i]q[i]$ denote their dot-product. Thus, $p^2 = \langle p, p\rangle = \|p\|^2$. Let P be a finite set of points in $\mathbb{R}^d$ (but any metric space equipped with a dot-product suffices), and let $\overline{\mathsf{m}}_P = \sum_{p\in P} p / |P|$ denote the **centroid** of P. The **average price** of the 1-mean clustering of P is

$$\nabla(P) = \sqrt{\sum\nolimits_{p\in P} \|p - \overline{\mathsf{m}}_P\|^2 / |P|} \leq \operatorname{diam}(P). \tag{2.1}$$

The last inequality follows as $\overline{\mathsf{m}}_P \in \mathsf{CH}(P)$, and for any $p \in P$, we have $\|p - \overline{\mathsf{m}}_P\| \leq \operatorname{diam}(P)$. This inequality can be tightened.

▶ **Lemma 1.** *We have $\nabla(P) \leq \operatorname{diam}(P)/\sqrt{2}$, and there is a point set Q in $\mathbb{R}^d$, such that*

$$\nabla(Q) \geq \left(1 - \tfrac{1}{d}\right)\tfrac{1}{\sqrt{2}}\operatorname{diam}(Q)$$

(i.e., the inequality is essentially tight).

Proof. This claim only improves the constant in our main result, and the reader can safely skip reading the proof. Let P be a set of n points in $\mathbb{R}^d$, with $\Delta = \operatorname{diam}(P)$ and $\nabla = \nabla(P)$. Assume that $\overline{\mathsf{m}}_P = 0$, as the claim is translation invariant. That is $\sum_{q\in P} q = 0$, and

$$\beta = \sum_{p,q\in P} \langle p, q\rangle = \sum_{p\in P} \left\langle p, \sum\nolimits_{q\in P} q\right\rangle = \sum_{p\in P} \langle p, 0\rangle = 0.$$

We have

$$\begin{aligned} n\nabla^2 &= \sum_{p\in P} \|p\|^2 = \sum_{p,q\in P} \frac{\|p\|^2 + \|q\|^2}{2n} = \sum_{p,q\in P} \frac{\|p\|^2 - 2\langle p, q\rangle + \|q\|^2}{2n} + \frac{2\beta}{2n} = \sum_{p,q\in P} \frac{\|p-q\|^2}{2n} \\ &\leq \sum_{p\in P, q\in P} \frac{\Delta^2}{2n} = \frac{n^2\Delta^2}{2n}. \end{aligned}$$

Implying that $\nabla^2 \leq \Delta^2/2$.

As for the lower bound, let e_i be the ith standard unit vector[1] in $\mathbb{R}^d$, and consider the point set $Q = \{e_1, \ldots, e_d\}$. We have that $\operatorname{diam}(Q) = \sqrt{2}$ and $\overline{\mathsf{m}}_Q = (1/d, \ldots, 1/d)$. Consequently,

$$\begin{aligned} \nabla(Q) &= \sqrt{\tfrac{1}{|Q|} \sum\nolimits_{q\in Q} \|q - \overline{\mathsf{m}}_Q\|^2} = \sqrt{\tfrac{|Q|}{|Q|}\left((1-1/d)^2 + (d-1)/d^2\right)} = \sqrt{\frac{(d-1)^2 + d - 1}{d^2}} \\ &= \sqrt{\frac{d-1}{d}} = \frac{\operatorname{diam}(Q)}{\sqrt{2}}\sqrt{1 - \frac{1}{d}} \geq \left(1 - \frac{1}{d}\right)\frac{1}{\sqrt{2}}\operatorname{diam}(Q). \end{aligned}$$ ◀

▶ **Definition 2.** *A subset $X \subseteq P$ is **δ-close** if the centroid of X is in distance at most $\delta\operatorname{diam}(P)$ from the centroid of P – that is, $\|\overline{\mathsf{m}}_X - \overline{\mathsf{m}}_P\| \leq \delta\operatorname{diam}(P)$.*

[1] That is, e_i is 0 in all coordinates except the ith coordinate where it is 1.

2.1 Proximity of centroid of a sample

The following is by now standard – a random sample of $O(1/\delta^2)$ points from P is δ-close with good probability, see Inaba et al. [15, Lemma 1]. We include the proof for the sake of completeness, as we require this somewhat specific form.

▶ **Lemma 3.** *Let P be a set of n points in $\mathbb{R}^d$, and $\delta \in (0,1)$ be a parameter. Let $R \subseteq P$ be a random sample of size r picked uniformly without replacement from P, where $r \geq \zeta/\delta^2$ and $\zeta > 1$ is a parameter. Then, we have $\mathbb{P}\big[\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_R\| > \delta\nabla(P)\big] < 1/\zeta$.*

Proof. Let $P = \{p_1, \ldots, p_n\}$. For simplicity of exposition, assume that $\overline{\mathsf{m}}_P = \sum_{i=1}^n \frac{1}{n} p_i = 0$, as the claim is translation invariant. For $\nabla = \nabla(P)$, and we have $\nabla^2 = \sum_{i=1}^n \frac{1}{n} p_i^2$. Let $Y = \sum_{p \in R} p = \sum_{i=1}^n I_i p_i$, where I_i is an indicator variable for p_i being in R. By linearity of expectations, we have

$$\mathbb{E}[Y] = \sum_{i=1}^n \mathbb{E}[I_i]\, p_i = \sum_{i=1}^n \frac{r}{n} p_i = r \sum_{i=1}^n \frac{1}{n} p_i = r\, \overline{\mathsf{m}}_P = 0.$$

Observe that, for $i \neq j$, we have

$$\mathbb{E}\big[I_i I_j\big] = \mathbb{P}\big[I_i = 1 \text{ and } I_j = 1\big] = \binom{n-2}{r-2} \Big/ \binom{n}{r} = \frac{(n-2)!}{(r-2)!(n-r)!} \cdot \frac{r!(n-r)!}{n!} = \frac{r(r-1)}{n(n-1)}. \tag{2.2}$$

By the above, and since $\mathbb{E}\big[I_i^2\big] = \mathbb{E}[I_i]$, we have

$$\begin{aligned}\mathbb{E}\big[\|Y\|^2\big] &= \mathbb{E}\big[\langle Y, Y\rangle\big] = \mathbb{E}\Big[\big(\sum_{i=1}^n I_i p_i\big)^2\Big] = \sum_{i=1}^n \mathbb{E}[I_i]\, p_i^2 + 2\sum_{i<j} \mathbb{E}\big[I_i I_j\big]\, p_i p_j \\ &= \sum_{i=1}^n \frac{r}{n} p_i^2 + 2\sum_{i<j} \frac{r(r-1)}{n(n-1)} p_i p_j \leq r\nabla^2 + \frac{r(r-1)n}{n-1} \Big(\sum_{i=1}^n \frac{1}{n} p_i\Big)^2 = r\nabla^2, \end{aligned} \tag{2.3}$$

using the shorthand $p_i p_j = \langle p_i, p_j \rangle$ and $p_i^2 = \langle p_i, p_i \rangle$. As (i) $r = |R|$, (ii) $\overline{\mathsf{m}}_R = Y/|R| = Y/r$, (iii) $r \geq \zeta/\delta^2$, and (iv) by Markov's inequality, we have

$$\mathbb{P}\big[\|\overline{\mathsf{m}}_R\| > \delta\nabla\big] = \mathbb{P}\bigg[\frac{\|Y\|}{r} > \delta\nabla\bigg] = \mathbb{P}\Big[\|Y\|^2 > (r\delta\nabla)^2\Big] \leq \frac{\mathbb{E}[\|Y\|^2]}{(r\delta\nabla)^2} \leq \frac{r\nabla^2}{(r\delta\nabla)^2} = \frac{1}{r\delta^2} \leq \frac{1}{\zeta}.$$

◀

Lemma 3 readily implies the no-dimensional Carathéodory theorem, see the full version for details.

2.2 Approximate Tverberg theorem

We now present the key technical lemma that will allow us to prove an approximate Tverberg theorem.

▶ **Lemma 4.** *Let P be a set of n points in $\mathbb{R}^d$, and $\delta \in (0,1)$ be a parameter, and assume that $n \gg 1/\delta^4$. Let $\nabla = \nabla(P)$. Then, one can compute, in $O(nd/\delta^2)$ expected time, a partition of P into k sets $P_1, \ldots, P_k$, and a ball $\mathscr{b}$, such that*

(i) $\forall i \;\; |P_i| \leq 4/\delta^2 + 6$,
(ii) $\forall i \;\; \mathsf{CH}(P_i) \cap \mathscr{b} \neq \emptyset$,
(iii) $\mathrm{radius}(\mathscr{b}) \leq \delta\nabla$, *and*
(iv) $k \geq n/(4/\delta^2 + 6)$.

Proof. Let $\mathscr{b} = \mathscr{b}(\overline{\mathsf{m}}_P, \delta\nabla)$. Let $\zeta = 2(1+\delta^2/8)$, and $M = \lceil \zeta/\delta^2 \rceil$. We randomly partition the points of P into $t = \lfloor n/M \rfloor > M$ sets $Q_1, \ldots, Q_t$, all of size either M or $M+1$ (this can be done by randomly permuting the points of P, and allocating each set a range of elements in this permutation). Thus, each Q_i, for $i \in [\![t]\!] = \{1, \ldots, t\}$, is a random sample according to Lemma 3 with parameter $\geq \zeta$. Thus, with probability $\geq 1 - 1/\zeta$, the set Q_i, for $i \in [\![t]\!]$, is δ-close – that is, $\left\| \overline{\mathsf{m}}_{Q_i} - \overline{\mathsf{m}}_P \right\| \leq \delta\nabla$, and Q_i is then considered to be *good*.

Let Z be the number of bad sets in $Q_1, \ldots, Q_t$. The probability of a set to be bad is at most $1/\zeta$, and by linearity of expectations, $\mathbb{E}[Z] \leq t/\zeta$. Let $\beta = t(1+\delta^2/8)/\zeta = t/2$. By Markov's inequality, we have

$$\mathbb{P}\big[Z \geq t/2\big] = \mathbb{P}[Z \geq \beta] \leq \frac{\mathbb{E}[Z]}{\beta} \leq \frac{t/\zeta}{(1+\delta^2/8)t/\zeta} = \frac{1}{1+\delta^2/8} \leq 1 - \frac{\delta^2}{16}. \tag{2.4}$$

We consider a round of sampling *successful* if $Z < \beta = t/2$. The algorithm can perform the random partition and compute the centroid for all P_i in $O(nd)$ time overall. Since a round is successful with probability $\geq \delta^2/16$, after $\lceil 16/\delta^2 \rceil$ rounds, the algorithm succeeds with constant probability. This implies that the algorithm performs, in expectation, $O(1/\delta^2)$ rounds till being successful, and the overall running time is $O(nd/\delta^2)$ time in expectation.

In the (first and final) successful round, the number of bad sets is $< t/2$ – namely, it is strictly smaller than the number of good sets. Therefore, we can match each bad set B in the partition to a unique good set G, and replace both of them by a new set $X = G \cup B$. That is, every good set absorbs at most one bad set, forming a new partition with roughly half the sets. For such a newly formed set X, we have that

$$|X| = |B| + |G| \leq 2(M+1) \leq 2\lceil \zeta/\delta^2 \rceil + 2 = 2\left\lceil \frac{2(1+\delta^2/8)}{\delta^2} \right\rceil + 2 \leq 2\left(\frac{2}{\delta^2} + 2\right) + 2 \leq \frac{4}{\delta^2} + 6.$$

The point $\overline{\mathsf{m}}_G$ is in $\mathsf{CH}(G) \subset \mathsf{CH}(X)$, and $\overline{\mathsf{m}}_G$ is in distance at most $\delta\nabla$ from the centroid of P. Thus, all the newly formed sets in the partition are in distance $\leq \delta\nabla$ from $\overline{\mathsf{m}}_P$, and $\mathsf{CH}(X) \cap \mathscr{b} \neq \varnothing$.

Finally, we have that the number of sets in the merged partition is at least $k \geq \frac{n}{4/\delta^2+6}$. ◀

▶ Remark 5. The mysterious requirement that $n \gg 1/\delta^4$, in Lemma 4, is used in the partition implicitly – the number of sets in the partition needs to be even. Thus, one set might need to be absorbed in the other sets, or more precisely two sets, because of the rounding issues. Namely, we first partition the set into groups of size M, and we need at least $2M+2$ sets in the partition to have size M (one additional last set can have size smaller than M). Thus, the proof requires that $n \geq (2M+2)M + M = (2M+3)M$. This is satisfied, for example, if $n \geq 27/\delta^4$.

▶ **Theorem 6.** *Let P be a set of n points in $\mathbb{R}^d$, and $\delta \in (0, 1/\sqrt{2})$ be a parameter, and assume that $n \gg 1/\delta^4$. Then, one can compute, in $O(nd/\delta^2)$ expected time, a partition of P into sets $P_1, \ldots, P_k$, and a ball $\mathscr{b}$, such that*

(i) $\forall i \ |P_i| \leq 2/\delta^2 + 6$,

(ii) $\forall i \ \mathsf{CH}(P_i) \cap \mathscr{b} \neq \emptyset$,

(iii) $\mathrm{radius}(\mathscr{b}) \leq \delta \, \mathrm{diam}(P)$, *and*

(iv) $k \geq n/(2/\delta^2 + 6)$.

Proof. Let $\Delta = \mathrm{diam}(P)$. Use Lemma 4 with parameter $\sqrt{2}\delta$. Observe that

$$\mathrm{radius}(\mathscr{b}) \leq \sqrt{2}\delta\nabla \leq \sqrt{2}\delta(\Delta/\sqrt{2}) = \delta\Delta,$$

by Lemma 1, where $\nabla = \nabla(P)$.

Observe that the algorithm does not require the value of $\mathrm{diam}(P)$, but rather the value of $\nabla(P)$, which can be computed in $O(nd)$ time, see Eq. (2.1). ◀

▶ **Corollary 7.** *The expected running time of Theorem 6 can be improved to $O(nd)$, with two of the guarantees being weaker:*

(I) *The sets are bigger: $\forall i \;\; |P_i| \leq 3/\delta^2 + 9$.*

(II) *And there are fewer sets: $k \geq n/(3/\delta^2 + 9)$.*

Proof. We use Lemma 4 as before, but now requiring only third of the sets to be good, and merging triples of sets to get one final good set. The probability of success is now constant, as Eq. (2.4) becomes

$$\mathbb{P}\Big[Z \geq \frac{2}{3}t\Big] = \mathbb{P}\Big[Z \geq \frac{4}{3}\cdot\frac{t}{2}\Big] = \mathbb{P}\Big[Z \geq \frac{4}{3}\beta\Big] \leq \frac{\mathbb{E}[Z]}{(4/3)\beta} \leq \frac{3}{4}.$$

Namely, the partition succeeds with probability at least $1/4$, which implies that the algorithm is done in expectation after $O(1)$ partition rounds. ◀

▶ Remark 8. The (existential) result of Adiprasito et al. [1, Theorem 1.3] has slightly worse constants, but it requires some effort to see, as they "maximize" the number of sets k (instead of minimizing the size of each set). Specifically, they show that one can partition P into k sets, with the computed ball having radius $(2+\sqrt{2})\sqrt{k/n}\,\mathrm{diam}(P)$ (intuitively, one wants k to be as large as possible). Translating into our language, we require that

$$(2+\sqrt{2})\sqrt{\frac{k}{n}} \leq \delta \implies (2+\sqrt{2})^2\frac{k}{n} \leq \delta^2 \implies k \leq n\frac{\delta^2}{(2+\sqrt{2})^2}.$$

Our result, on the other hand, states that k is at least (over-simplifying for clarify) $n\frac{\delta^2}{2}$ (for δ sufficiently small). Adiprasito et al. mention, as a side note, that their constant improves to $1+\sqrt{2}$ under certain conditions. Even then, the constant in the above theorem is better.

This improvement in the constant is small (and thus, arguably minor), but nevertheless, satisfying.

2.3 Tverberg halving

An alternative approach is to randomly halve the point set and observe that the centroids of two halves are close together. In this section, we show this line of thinking leads to various algorithms that can be derandomized efficiently. Foundational to this approach is the following lemma (which is a variant of Lemma 3).

▶ **Lemma 9.** *Let $U = \{u_1, \ldots, u_{2n}\}$ be a set of $2n$ points in $\mathbb{R}^d$ with $\Delta = \mathrm{diam}(U)$. For $i = 1, \ldots, n$, with probability $1/2$, let $p_i = u_{2i-1}, q_i = u_{2i}$, or otherwise, let $p_i = u_{2i}, q_i = u_{2i-1}$. Let $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}$. For any parameter $t \geq 1$, we have $\mathbb{P}\Big[\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\| \geq \frac{t}{\sqrt{n}}\Delta\Big] \leq \frac{1}{t^2}$.*

Proof. This follows by adapting the argument used in the proof of Lemma 3, and the details are included here for the sake of completeness.

Let $v_i = u_{2i-1} - u_{2i}$. Consider the random variable $Y = \overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q = \sum_{i=1}^n \frac{X_i v_i}{n}$, where $X_i \in \{-1, +1\}$ is picked independently with probability half. We first observe that

(i) $\mathbb{E}[Y] = \sum_{i=1}^n \mathbb{E}[X_i]\, v_i/n = 0$,

(ii) $\mathbb{E}\big[X_i^2\big] = 1$, and

(iii) for $i < j$, $\mathbb{E}\big[X_i X_j\big] = 0$.

Thus, we have

$$\begin{aligned}\mathbb{E}\big[\|Y\|^2\big] = \mathbb{E}\big[\langle Y, Y\rangle\big] &= \mathbb{E}\Big[\Big\langle \sum\nolimits_{i=1}^{n} \frac{X_i v_i}{n}, \sum\nolimits_{i=1}^{n} \frac{X_i v_i}{n} \Big\rangle\Big] \\ &= \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{E}\Big[X_i^2\Big]\, v_i^2 + 2\frac{1}{n^2}\sum_{i<j} \mathbb{E}\big[X_i X_j v_i v_j\big] \\ &= \frac{1}{n^2}\sum_{i=1}^{n} v_i^2 \le \frac{n\Delta^2}{n^2} = \frac{\Delta^2}{n}, \end{aligned} \tag{2.5}$$

since $\|v_i\| = \|u_{2i-1} - u_{2i}\| \le \operatorname{diam}(U) = \Delta$. By Markov's inequality, we have

$$\mathbb{P}\Big[\|Y\| > t\frac{\Delta}{\sqrt{n}}\Big] = \mathbb{P}\Big[\|Y\|^2 > t^2\frac{\Delta^2}{n}\Big] \le \frac{\mathbb{E}[\|Y\|^2]}{t^2\Delta^2/n} \le \frac{1}{t^2}. \qquad ◀$$

Remarks.

(A) Lemma 9 can be turned into an efficient algorithm using the same Markov's inequality argument used in Theorem 6. Specifically, for any parameter $\xi \in (0,1)$, one can compute a partition into two sets P and Q with $\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\| \le (1+\xi)\Delta/\sqrt{n}$, in $O(nd/\xi)$ expected time.

(B) Lemma 9 implies that there exists a partition P and Q of U such that

$$\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\| \le \Delta/\sqrt{n}.$$

Note that this is tight. To see this, let U be the standard basis of $\mathbb{R}^{2n}$, with its diameter $\Delta = \sqrt{2}$. For any partition P and Q of U with $|P| = |Q| = n$, we have that $\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\| = \sqrt{2\sum_{i=1}^{n} 1/n^2} = \sqrt{2/n} = \Delta/\sqrt{n}$.

(C) As in the standard algorithm for computing a δ-net via discrepancy [5, 18], one can apply repeated halving to get the desired Tverberg partition until the sets are the desired size. This provides a method for a deterministic algorithm, which we present in Section 4.3.

3 Applications

3.1 No-dimensional centerball

We present an efficient no-dimensional centerpoint theorem; the previous version [1, Theorem 7.1] did not present an efficient algorithm.

▶ **Corollary 10** (No-dimensional centerpoint). *Let P be a set of n points in $\mathbb{R}^d$ and $\delta \in (0, 1/2)$ be a parameter, where n is sufficiently large (compared to δ). Then, one can compute, in $O(nd/\delta^2)$ expected time, a ball $\mathscr{b}$ of radius $\delta\operatorname{diam}(P)$, such that any halfspace containing $\mathscr{b}$ contains at least $\Omega(\delta^2 n)$ points of P.*

Proof. Follows by applying Theorem 6 and the observation that, for any halfspace containing the computed ball $\mathscr{b}$, it must also contain at least one point from each set of the partition $P_1, \ldots, P_k$, where $k = \Omega(\delta^2 n)$. Thus, the ball $\mathscr{b}$ is as desired. ◀

3.2 No-dimensional weak ε-net theorem

Originally given by Adiprasito et al. [1, Theorem 7.3], we prove a version of the no-dimensional weak ε-net theorem with an improved dependence on the parameters. For a sequence $Q = (q_1, \ldots, q_r) \in P^r$, let $\overline{\mathsf{m}}_Q = \sum_{i=1}^{r} q_i/r$. We reprove Lemma 3 under a slightly different sampling model.

▶ **Lemma 11.** *Let P be a set of n points in $\mathbb{R}^d$, and $\delta \in (0,1/2)$ and $\zeta > 1$ be parameters. Let $r \geq \zeta/\delta^2$. For a random sequence $Q = (q_1, \ldots, q_r)$ picked uniformly at random from P^r, we have that $\mathbb{P}\Big[\big\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\big\| > \delta\Delta\Big] \leq 1/\zeta$, where $\Delta = \mathrm{diam}(P)$.*

Proof. The argument predictably follows the proof of Lemma 3, and the reader can safely skip reading it, as it adds little new. Assume that $\overline{\mathsf{m}}_P = \sum_{i=1}^n \frac{1}{n} p_i = 0$. Let $\nabla^2 = \sum_{i=1}^n \frac{1}{n} p_i^2$ and $Y = \sum_{i=1}^r q_i$. Then, $\mathbb{E}[Y] = \sum_{i=1}^r \mathbb{E}[q_i] = 0$. As $\|Y\|^2 = \langle Y, Y\rangle$, it follows that

$$\begin{aligned}\mathbb{E}\big[\|Y\|^2\big] &= \mathbb{E}\Big[\big(\sum_{i \underset{n}{=} 1}^{r} q_i\big)^2\Big] = \sum_{k=1}^{r} \mathbb{E}\Big[q_k^2\Big] + 2\sum_{i<j} \mathbb{E}[q_i q_j] \\ &= \sum_{k=1}^{r}\sum_{i=1}^{} \tfrac{1}{n} p_i^2 + 2\sum_{i<j} \mathbb{E}[q_i]\,\mathbb{E}[q_j] = r\nabla^2.\end{aligned}$$

Since $\overline{\mathsf{m}}_R = Y/r$, $r \geq \zeta/\delta^2$, and by Markov's inequality, we have

$$\mathbb{P}\big[\|\overline{\mathsf{m}}_R\| > \delta\nabla\big] = \mathbb{P}\bigg[\frac{\|Y\|}{r} > \delta\nabla\bigg] = \mathbb{P}\Big[\|Y\|^2 > (r\delta\nabla)^2\Big] \leq \frac{\mathbb{E}\big[\|Y\|^2\big]}{(r\delta\nabla)^2} \leq \frac{r\nabla^2}{(r\delta\nabla)^2} = \frac{1}{r\delta^2} \leq \frac{1}{\zeta}.$$

◀

A sequence $Q \in P^r$ **collides** with a ball $\mathscr{b}$ if $\mathscr{b}$ intersects $\mathsf{CH}(Q)$. In particular, if $\big\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q\big\| \leq \delta\Delta$, then Q collides with the ball $\mathscr{b}(\overline{\mathsf{m}}_P, \delta\Delta)$, where $\Delta = \mathrm{diam}(P)$.

▶ **Lemma 12** (Selection lemma)**.** *Let P be a set of n points in $\mathbb{R}^d$ and $\delta \in (0,1)$ be a parameter. Let $r = \lceil 2/\delta^2 \rceil$. Then, the ball $\mathscr{b} = \mathscr{b}(\overline{\mathsf{m}}_P, \delta\Delta)$ collides with at least $n^r/2$ sequences of P^r.*

Proof. Taking $\zeta = 2$, by Lemma 11, a random r-sequence from P^r has probability at least half to collide with $\mathscr{b}$, which readily implies that this property holds for half the sequences in P^r. ◀

▶ **Theorem 13** (No-dimensional weak ε-net)**.** *Let P be a set of n points in $\mathbb{R}^d$, with diameter Δ, and $\delta, \varepsilon \in (0,1)$ be parameters, where $2/\delta^2$ is an integer. Then, there exists a set $F \subset \mathbb{R}^d$ of size $\leq 2\varepsilon^{-2/\delta^2}$ balls, each of radius $\delta\Delta$, such that, for all $Y \subset P$, with $|Y| \geq \varepsilon n$, F contains a ball of radius $\delta\Delta$ that intersects $\mathsf{CH}(Y)$.*

Proof. Our argument follows Alon et al. [2]. Let $r = 2/\delta^2$. Initialize $F = \varnothing$, and let $\mathcal{H} = P^r$. If there is a set $Q \subset P$, with $|Q| \geq \varepsilon n$, where no ball of F intersects $\mathsf{CH}(Q)$, then applying Lemma 12 to Q, the algorithm computes a ball $\mathscr{b}$, of radius $\delta\Delta$, that collides with at least $(\varepsilon n)^r/2$ sequences of Q^r. The algorithm adds $\mathscr{b}$ to the set F, and removes from $\mathcal{H}$ all the sequences that collide with $\mathscr{b}$. The algorithm continues till no such set Q exists.

As initially $|\mathcal{H}| = n^r$, the number of iterations of the algorithm, and thus the size of F, is bounded by $\frac{n^r}{(\varepsilon n)^r/2} = 2/\varepsilon^r$. ◀

▶ Remark 14. In the version given by Adiprasito et al. [1, Theorem 7.3], the set F has size at most $(2/\delta^2)^{2/\delta^2}\varepsilon^{-2/\delta^2}$, while our bound is $2\varepsilon^{-2/\delta^2}$.

4 Derandomization

4.1 Derandomizing mean sampling

Lemma 3 can be derandomized directly using conditional expectations. We also present a more efficient derandomization scheme using halving in Section 4.2.

▶ **Lemma 15.** *Let P be a set of n points in $\mathbb{R}^d$. Then, for any integer $r \geq 1$, one can compute, in deterministic $O(dn^3)$ time, a subset $R \subset P$ of size r, such that $\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_R\| \leq \nabla(P)/\sqrt{r} \leq \operatorname{diam}(P)/\sqrt{2r}$, where $\nabla = \nabla(P)$, see Eq. (2.1).*

Proof. We derandomize the algorithm of Lemma 3. We assume for simplicity of exposition that $\overline{\mathsf{m}}_P = 0$. Let R be a sample of size r without replacement from P, and let $I_i \in \{0,1\}$ be the indicator for the event that $p_i \in R$.

Let $Y = \sum_{i=1}^n I_i p_i$. Then, $\overline{\mathsf{m}}_R = Y/r$, and thus $\|\overline{\mathsf{m}}_R - \overline{\mathsf{m}}_P\| = \|Y\|/r$. Consider the quantity

$$\beta = Z(x_1, \ldots, x_t) = \mathbb{E}\Big[\|Y\|^2 \;\Big|\; \mathcal{E}\Big], \qquad \mathcal{E} \equiv (I_1 = x_1, \ldots, I_t = x_t),$$

where the expectation is over the random choices of $I_{t+1}, \ldots, I_n$. At the beginning of the $(t+1)$th iteration, the values of $x_1, \ldots, x_t$ were determined in earlier iterations, and the task at hand is to decide what value to assign to x_{t+1} that minimizes $Z(x_1, \ldots, x_t, x_{t+1})$. Thus, the algorithm computes $\beta_0 = Z(x_1, \ldots, x_t, 0)$ and $\beta_1 = Z(x_1, \ldots, x_t, 1)$.

Using conditional expectations, Eq. (2.3) becomes

$$\beta = \mathbb{E}\Big[\|Y\|^2 \;\Big|\; \mathcal{E}\Big] = \sum_{i=1}^n \mathbb{E}[I_i \mid \mathcal{E}]\, p_i^2 + 2\sum_{i<j} \mathbb{E}\big[I_i I_j \;\big|\; \mathcal{E}\big]\, p_i p_j. \tag{4.1}$$

Let $\alpha = \sum_{k=1}^t x_k$, and observe that $r - \alpha$ points are left to be chosen to be in R after $\mathcal{E}$. As such, arguing as in Eq. (2.2), for $i < j$, we have

$$\mathbb{E}[I_i \mid \mathcal{E}] = \begin{cases} x_i & i \leq t \\ \frac{r-\alpha}{n-t} & i > t, \end{cases} \qquad \text{and} \qquad \mathbb{E}\big[I_i I_j \;\big|\; \mathcal{E}\big] = \begin{cases} x_i x_j & i < j \leq t \\ x_i \frac{r-\alpha}{n-t} & i \leq t < j \\ \frac{(r-\alpha)(r-\alpha-1)}{(n-t)(n-t-1)} & t < i < j. \end{cases} \tag{4.2}$$

This implies that the algorithm can compute β in quadratic time directly via Eq. (4.1). Similarly, the algorithm computes β_0 and β_1. Observe that

$$\beta = Z(x_1, \ldots, x_t) = \frac{r-\alpha}{n-t}\beta_1 + \frac{n-t-(r-\alpha)}{n-t}\beta_0.$$

Namely, β is a convex combination of β_0 and β_1. Thus, if $\beta_0 \leq \beta$ then the algorithm sets $x_{t+1} = 0$, and otherwise the algorithm sets $x_{t+1} = 1$.

The algorithm now performs n such assignment steps, for $t = 0, \ldots, n-1$, to compute an assignment of $x_1, \ldots, x_n$ such that $Z(x_1, \ldots, x_n) \leq \mathbb{E}[\|Y\|^2]$. Overall, this leads to a $O(dn^3)$ time algorithm. Specifically, the algorithm outputs a set $R \subseteq P$ of size r, such that $R = \{p_i \mid x_i = 1, i = 1, \ldots, n\}$. Observe that $Z(x_1, \ldots, x_n) = \|r\overline{\mathsf{m}}_R\|^2 \leq \mathbb{E}[\|Y\|^2]$. Thus, by Eq. (2.3) and Lemma 1, we have

$$\|\overline{\mathsf{m}}_R - \overline{\mathsf{m}}_P\| = \|\overline{\mathsf{m}}_R\| \leq \sqrt{\frac{\mathbb{E}[\|Y\|^2]}{r^2}} \leq \sqrt{\frac{r\nabla^2}{r^2}} = \frac{\nabla}{\sqrt{r}} \leq \frac{\operatorname{diam}(P)}{\sqrt{2r}}. \qquad ◀$$

With some care, the running time of the algorithm of Lemma 15 can be improved to $O(dn)$ time, but the details are tedious, and we delegate the proof of the following lemma to the full version.

▶ **Lemma 16.** *Let P be a set of n points in $\mathbb{R}^d$. Then, for any integer $r \geq 1$, one can compute, in $O(dn)$ deterministic time, a subset $R \subset P$ of size r, such that $\|\overline{\mathsf{m}}_P - \overline{\mathsf{m}}_R\| \leq \nabla(P)/\sqrt{r} \leq \operatorname{diam}(P)/\sqrt{2r}$.*

4.2 Derandomizing the halving scheme

The algorithm of Lemma 9 can be similarly derandomized.

▶ **Lemma 17.** *Let $U = \{u_1, \dots, u_{2n}\}$ be a set of $2n$ points in $\mathbb{R}^d$ with $\Delta = \mathrm{diam}(U)$. One can partition U, in deterministic $O(dn)$ time, into two equal size sets P and Q, such that $\left\| \overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q \right\| \leq \Delta/\sqrt{n}$.*

Proof. We follow Lemma 9. To this end, let $v_i = u_{2i-1} - u_{2i}$, for $i = 1, \dots, n$. Let $Y = \sum_{i=1}^n \frac{X_i v_i}{n}$, where $X_i \in \{-1, +1\}$. Next, consider the quantity

$$Z(x_1, \dots, x_t) = \mathbb{E}\Big[\|Y\|^2 \;\Big|\; \mathcal{E} \Big], \qquad \mathcal{E} \equiv (X_1 = x_1, \dots, X_t = x_t),$$

where the expectation is over the random choices of $X_{t+1}, \dots, X_n$. By Eq. (2.5), we have $Z(x_1, \dots, x_t) = \frac{1}{n^2} \sum_{i=1}^n v_i^2 + \frac{2}{n^2} \sum_{i<j} \mathbb{E}\big[X_i X_j v_i v_j \;\big|\; \mathcal{E} \big]$. The latter term is

$$\begin{aligned} \sum_{i<j} \mathbb{E}\big[X_i X_j v_i v_j \mid \mathcal{E} \big] &= \sum_{i<j : i,j \leq t} x_i x_j v_i v_j + \sum_{i<j : i \leq t < j} \mathbb{E}\big[x_i X_j v_i v_j \big] + \sum_{i<j : t<i,j} \mathbb{E}\big[X_i X_j v_i v_j \big] \\ &= \sum_{i<j \leq t} x_i x_j v_i v_j, \end{aligned}$$

as $\mathbb{E}[X_i] = \mathbb{E}\big[X_i X_j\big] = 0$. Thus, $Z(x_1, \dots, x_t) = \frac{1}{n^2} \sum_{i=1}^n v_i^2 + \frac{2}{n^2} \sum_{i<j \leq t} x_i x_j v_i v_j$. The key observation is that

$$Z(x_1, \dots, x_t) = \frac{Z(x_1, \dots, x_t, -1) + Z(x_1, \dots, x_t, +1)}{2}.$$

Our goal is to compute the assignment of $x_1, \dots, x_n$ that minimizes Z. Observe that

$$D_t = Z(x_1, \dots, x_t, +1) - Z(x_1, \dots, x_t) = \frac{2}{n^2} \Big(\sum_{i<t+1} x_i v_i \Big) v_{t+1}.$$

If $D_t \leq 0$, then the algorithm sets $x_{t+1} = +1$, otherwise the algorithm sets $x_{t+1} = -1$. The algorithm has to repeat this process for $t = 1, \dots, n$, and naively, each step takes $O(dn)$ time. Observe that if the algorithm maintains the quantity $V_t = \sum_{i=1}^t x_i v_i$, then D_t can be computed in $O(d)$ time. This determines the value of x_{t+1}, and the value of $V_{t+1} = V_t + x_{t+1} v_{t+1}$ can be maintained in $O(d)$ time. As each iteration takes $O(d)$ time, the algorithm overall takes $O(dn)$ time. By the end of this process, the algorithm will have computed an assignment $x_1, \dots, x_n$, with an associated partition of U into P and Q. By Eq. (2.5), we have $\left\| \overline{\mathsf{m}}_P - \overline{\mathsf{m}}_Q \right\|^2 \leq \mathbb{E}\big[\|Y\|^2 \big] \leq \Delta^2/n$. ◀

4.3 A deterministic approximate Tverberg partition

▶ **Lemma 18.** *Let P be a set of n points in $\mathbb{R}^d$, and $\delta \in (0, 1/4)$ be a parameter. Then, one can compute, in $O(nd \log n)$ deterministic time, a partition of P into sets $P_1, \dots, P_k$, and a ball $\mathscr{b}$, such that*

(i) $\forall i \;\; |P_i| \leq 8/\delta^2$,
(ii) $\forall i \;\; \mathsf{CH}(P_i) \cap \mathscr{b} \neq \emptyset$,
(iii) $\mathrm{radius}(\mathscr{b}) \leq \delta \mathrm{diam}(P)$, *and*
(iv) $k \geq n\delta^2/8$.

Proof. Assume for the time being that n is a power of 2. As done for discrepancy, we halve the current point set, and then continue doing this recursively (on both resulting sets), using the algorithm of Lemma 17 at each stage. Conceptually, this is done in a binary tree fashion, and doing this for i levels breaks the point set into 2^i sets. Let ℓ_i be an upper bound on the distance of the centroid of a set in the ith level from the centroid of its parent. By Lemma 17, we have $2\ell_i \leq \Delta/\sqrt{n/2^i}$ (where $i = 1$ in the top level). Thus, repeating this process for t levels, we have that the distance of any centroid at the leaves to the global centroid is bounded by

$$L_t = \sum_{i=1}^{t} \ell_i \leq \sum_{i=1}^{t} \frac{\Delta}{2\sqrt{n/2^i}} = \frac{\Delta}{\sqrt{2n}} \sum_{i=0}^{t-1} \sqrt{2^i} = \frac{\Delta}{\sqrt{2n}} \left(\frac{2^{t/2} - 1}{\sqrt{2} - 1} \right)$$
$$\leq \frac{5\Delta}{2\sqrt{2n}} 2^{t/2} = \frac{5\Delta}{2\sqrt{2}} \sqrt{\frac{1}{n/2^t}}. \tag{4.3}$$

Solving for $\frac{5}{2\sqrt{2}}\sqrt{\frac{1}{n/2^t}} \leq \delta$, we get that this holds for $n/2^t \geq 3.2/\delta^2$. We stop our halving procedure once t is large enough such that the preceding inequality no longer holds, implying the stated bound on the size of each set.

If n is not a power of 2 then we apply the above algorithm to the largest subset that has size that is a power of two, and then add the unused points in a round robin fashion to the sets computed. ◀

▶ Remark 19. If instead of keeping both halves, as done by the algorithm of Lemma 18, one throws one of the halves away, and repeats the halving process on the other half, we end up with a single sample. One can repeat this halving process until the "sample" size is $\Theta(1/\delta^2)$. Using the same argument as in Eq. (4.3) to bound the error, we obtain a sample R of size $\Theta(1/\delta^2)$, such that $\|\overline{\mathsf{m}}_R - \overline{\mathsf{m}}_P\| \leq \delta \operatorname{diam}(P)$. The running time is $\sum_i O(dn/2^i) = O(dn)$. Namely, we get a deterministic $O(dn)$ time algorithm that computes a sample with the same guarantees as Lemma 16 – this version is somewhat less flexible and the constants are somewhat worse.

5 Conclusions

Given a data set, archetypal analysis [6] aims to identify a small subset of points such that all (or most) points in the data can be represented as a sparse convex-combination of these "archtypes". Thus, for a sparse convex-combination of points, generating a point can be viewed as an "explanation" of how it is being induced by the data. It is thus natural to ask for as many *independent* explanations as possible for a point – the more such combinations, the more a point "arises" naturally from the data. Thus, an approximate Tverberg partition can be interpreted as stating that high dimensional data has certain points (i.e., the centroid) that are robustly generated by the data.

From a data-analysis point of view, an interesting open question is whether one can do better than the "generic" guarantees provided here. If, for example, a smaller radius centroid ball exists, can it be approximated efficiently? Can a sparser convex-combination of points be computed efficiently?

While these questions in the most general settings seem quite challenging, even solving them in some special cases might be interesting.

In addition, prior works consider other no-dimensional results, such as a no-dimensional version of Helly's theorem [1], and a no-dimensional version of the colorful Tverberg theorem [7]. Our work did not address these problems because of the focus on simplicity, and a possible further direction is to address these variants with extensions of the techniques used here.

References

1 Karim A. Adiprasito, Imre Bárány, Nabil H. Mustafa, and Tamás Terpai. Theorems of Carathéodory, Helly, and Tverberg without dimension. *Discrete Comput. Geom.*, 64(2):233–258, 2020. `doi:10.1007/s00454-020-00172-5`.

2 Noga Alon, Imre Bárány, Zoltán Füredi, and Daniel J. Kleitman. Point selections and weak e-nets for convex hulls. *Comb. Probab. Comput.*, 1:189–200, 1992. `doi:10.1017/S0963548300000225`.

3 Imre Bárány and Pablo Soberón. Tverberg's theorem is 50 years old: a survey. *Bull. Amer. Math. Soc. (N.S.)*, 55(4):459–492, 2018. `doi:10.1090/bull/1634`.

4 Timothy M. Chan, Sariel Har-Peled, and Mitchell Jones. Optimal algorithms for geometric centers and depth. *SIAM J. Comput.*, 51(3):627–663, 2022. `doi:10.1137/21M1423324`.

5 Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, New York, 2001. URL: `http://www.cs.princeton.edu/~chazelle/book.html`.

6 Yuansi Chen, Julien Mairal, and Zaid Harchaoui. Fast and robust archetypal analysis for representation learning. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1478–1485, USA, 2014. IEEE Computer Society. `doi:10.1109/CVPR.2014.192`.

7 Aruni Choudhary and Wolfgang Mulzer. No-dimensional Tverberg theorems and algorithms. *Discrete Comput. Geom.*, 68(4):964–996, 2022. `doi:10.1007/s00454-022-00380-1`.

8 Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 922–931. SIAM, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347183`.

9 Kenneth L. Clarkson, David Eppstein, Gary L. Miller, Carl Sturtivant, and Shang-Hua Teng. Approximating center points with iterative Radon points. *Int. J. Comput. Geom. Appl.*, 6(3):357–377, 1996. `doi:10.1142/S021819599600023X`.

10 Jesús A. De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil H. Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bull. Amer. Math. Soc. (N.S.)*, 56(3):415–511, 2019. `doi:10.1090/bull/1653`.

11 Sariel Har-Peled and Mitchell Jones. Journey to the center of the point set. *ACM Trans. Algorithms*, 17(1):9:1–9:21, 2021. `doi:10.1145/3431285`.

12 Sariel Har-Peled and Eliot Wong Robson. No-dimensional Tverberg partitions revisited. *CoRR*, abs/2306.01678, 2023. `doi:10.48550/arXiv.2306.01678`.

13 Sariel Har-Peled and Timothy Zhou. Improved approximation algorithms for Tverberg partitions. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *Proc. 30th Annu. Euro. Sympos. Alg.* (ESA), volume 204 of *LIPIcs*, pages 51:1–51:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ESA.2021.51`.

14 Eduard Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatsh. Math. Phys.*, 37(1):281–302, 1930. `doi:10.1007/BF01696777`.

15 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering (extended abstract). In Kurt Mehlhorn, editor, *Proc. 10th Annu. Sympos. Comput. Geom.* (SoCG), pages 332–339. ACM, 1994. `doi:10.1145/177424.178042`.

16 Shreesh Jadhav and Asish Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete Comput. Geom.*, 12:291–312, 1994. doi:10.1007/BF02574382.

17 J. Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Grad. Text in Math.* Springer, 2002. doi:10.1007/978-1-4613-0039-7.

18 Jirí Matoušek. *Geometric Discrepancy*, volume 18 of *Algorithms and Combinatorics*. Springer, 1999. doi:10.1007/978-3-642-03942-3.

19 Gary L. Miller and Donald R. Sheehy. Approximate centerpoints with proofs. *Comput. Geom. Theory Appl.*, 43(8):647–654, 2010. doi:10.1016/j.comgeo.2010.04.006.

20 Helge Tverberg and Sinsia Vreica. On generalizations of Radon's theorem and the ham sandwich theorem. *Eur. J. Comb.*, 14(3):259–264, 1993. doi:10.1006/eujc.1993.1029.

Optimizing Visibility-Based Search in Polygonal Domains

Kien C. Huynh
Linköping University, Sweden[1]

Joseph S. B. Mitchell
Stony Brook University, NY, USA

Linh Nguyen[2]
Stony Brook University, NY, USA

Valentin Polishchuk
Linköping University, Sweden

Abstract

Given a geometric domain P, visibility-based search problems seek routes for one or more mobile agents ("watchmen") to move within P in order to be able to see a portion (or all) of P, while optimizing objectives, such as the length(s) of the route(s), the size (e.g., area or volume) of the portion seen, the probability of detecting a target distributed within P according to a prior distribution, etc. The classic watchman route problem seeks a shortest route for an observer, with omnidirectional vision, to see all of P. In this paper we study bicriteria optimization problems for a single mobile agent within a polygonal domain P in the plane, with the criteria of route length and area seen. Specifically, we address the problem of computing a minimum length route that sees at least a specified area of P (minimum length, for a given area quota). We also study the problem of computing a length-constrained route that sees as much area as possible. We provide hardness results and approximation algorithms. In particular, for a simple polygon P we provide the first fully polynomial-time approximation scheme for the problem of computing a shortest route seeing an area quota, as well as a (slightly more efficient) polynomial dual approximation. We also consider polygonal domains P (with holes) and the special case of a planar domain consisting of a union of lines. Our results yield the first approximation algorithms for computing a time-optimal search route in P to guarantee some specified probability of detection of a static target within P, randomly distributed in P according to a given prior distribution.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Quota watchman route problem, budgeted watchman route problem, visibility-based search, approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.27

Related Version *Full Version*: https://arxiv.org/abs/2402.05420

Funding This work is partially supported by the National Science Foundation (CCF-2007275) and the Swedish Research Council and the Swedish Transport Administration.

1 Introduction

We investigate the Quota Watchman Route problem (QWRP) and the Budgeted Watchman Route problem (BWRP) for a single mobile agent (a "watchman") within a polygonal domain P in the plane. These problems naturally arise in various applications, including motion planning, search-and-rescue, surveillance, and exploration of a polygonal

[1] Work done at Stony Brook University.
[2] Corresponding author

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 27; pp. 27:1–27:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

domain, where complete coverage is not feasible due to shortage of fuel, time, etc. The QWRP seeks a route/tour that sees at least some specified area of the domain P with a shortest length, while the BWRP seeks a route/tour that sees the maximum area subject to a length constraint. Both can be seen as extensions of the well-known Watchman Route Problem (WRP) with different objectives and constraints.

The challenge in addressing the trade-off between area seen and tour length is that one is not able to exploit the optimality structure that is implied by having to see *all* of a polygon P. It is this structure, yielding an ordered sequence of "essential cuts", that allows the WRP to be solved efficiently, e.g., as an instance of the "touring polygons problem" [13].

Results. We address the challenge by establishing new structural results that enable a careful discretization and analysis, along with carefully crafted dynamic programs. We provide several new results on optimal visibility search in a polygon:

(1) We prove that the QWRP and the BWRP are (weakly) NP-hard, even in a simple polygon; this is to be contrasted with the WRP, for which exact polynomial-time algorithms are known in simple polygons.

(2) For the QWRP in a simple polygon P, we give the first fully polynomial-time approximation scheme (FPTAS), as well as a dual-approximation (with slightly more efficient running time than the FPTAS) that computes a tour having length at most $(1 + \varepsilon_1)$ times the length of an optimal tour that sees area at least A (where A is the area quota), while seeing area at least $(1 - \varepsilon_2)A$ for any $\varepsilon_1, \varepsilon_2 > 0$.

(3) For the BWRP in simple P, we compute, in polynomial time, a tour of length at most $(1 + \varepsilon)B$ that sees a region within P of area at least that seen by an optimal tour of length at most B.

(4) In a multiply connected domain, in a polygon P with holes, we provide hardness of approximations and a $(1 + \varepsilon, O(\log n))$-dual approximation ($n$ is the number of vertices of P) for the BWRP. In the special case of an arrangement of lines, we obtain polynomial-time exact algorithms for both problems.

(5) We solve two visibility-based stochastic search problems that seek to locate a static target given a prior probability distribution of its location within P: (a) compute the minimum time to achieve a specified detection probability; (b) compute a search route maximizing the probability of detection by time T for a mobile searcher.

Related Work

Chin and Ntafos introduced the classic Watchman Route Problem (WRP) [10]: compute a shortest closed route (tour) within a polygon P from which every point of P can be seen; they gave an $O(n)$-time algorithm for computing an optimal tour in a simple orthogonal polygon, and later results established polynomial-time exact algorithms for the WRP in a simple polygon P, both with and without an anchor point (depot) [5, 10, 11, 13, 26, 29, 30]. In a polygon P with holes, the WRP is NP-hard [10, 17] and is, in fact, NP-hard to approximate better than a logarithmic factor [25]; however, an $O(\log^2 n)$-approximation algorithm is known [25]. The BWRP and the QWRP are natural variants of the WRP.

Another related problem is that of maximum visibility coverage with point guards: Given an integer k, place k point guards within P to maximize the area of P seen by the guards. When k is arbitrary, the problem is NP-hard [26], since an exact solution to this problem would yield a method to compute the minimum number of guards needed to see a polygon. Viewed as a maximum coverage problem, one can greedily compute an approximation, with factor $\left(1 - \frac{1}{e}\right)$, by iteratively placing a guard that sees the most unseen area [9, 26].

The BWRP is related to the ORIENTEERING problem. Given a budget constraint and an edge-weighted graph where each vertex is associated with a prize, the objective of ORIENTEERING is to find a path/tour within the length budget maximizing the total reward of the vertices visited. On the other hand, the QWRP is related to the QUOTA TRAVELING SALESPERSON problem, which aims to minimize the distance travelled to achieve a given quota of reward. The Euclidean versions of ORIENTEERING and Quota TSP have polynomial-time approximation schemes [8, 20, 24]. Both the QWRP and the BWRP can be considered a reward (the area of P seen by the watchman) collecting process; however, the main difference lies in the continuous nature of visibility, since we see portions of the domain as we travel to checkpoints, we must take into account the area that has been seen previously.

Optimal search theory has been extensively studied in discrete, graph theoretic settings; see, e.g., [18, 19, 31]. In geometric contexts, searching and target tracking have been studied in the form of VISIBILITY-BASED PURSUIT-EVASION games. In [22], the visibility-based version of the pursuit-evasion game was introduced and formulated as a geometric problem, in which an evader moves unpredictably, arbitrarily fast within a polygonal domain, and the goal is to strategically coordinate one or multiple pursuers to guarantee a finite time of detection. See the survey [12] on visibility-based pursuit-evasion games.

2 The QWRP in a Simple Polygon

2.1 Preliminaries and Hardness Results

A *simple polygon* P is a simply connected subset of $\mathbb{R}^2$ whose boundary, ∂P, is a polygonal cycle consisting of a finite set of line segments, whose endpoints are the *vertices*, $v_1, v_2, \ldots, v_n$, of P. A vertex is *reflex* (resp. *convex*) if its internal angle is at least (resp. at most) 180 degrees. We consider polygons to be closed sets, including the interior and the boundary. We use the notation $|\cdot|$ to denote the measure of several types of objects. In the case of a segment or a route γ, $|\gamma|$ denotes its length, while for a polygon P, $|P|$ denotes the area of P. For a finite set S, $|S|$ is the cardinality of S. Based on the object within the notation, the interpretation should be apparent.

Point $x \in P$ sees point $y \in P$ if the line segment connecting them lies entirely within P. The *visibility polygon* of x, denoted $V(x)$, is the closed region of P that x sees; necessarily, $V(x)$ is a simple polygon within P. For a subset $X \subset P$, let $V(X)$ be the set of points that are seen by at least one point in X; formally, $V(X) = \bigcup_{x \in X} V(x)$. The visibility polygon of a point or a segment can be computed in time $O(n)$ for a simple polygon, or in time $O(n + h \log h)$ for a polygonal domain with n vertices and h holes [27]. Given a domain P (a simple polygon or a polygon with holes) and an area quota $0 \leq A \leq |P|$, in the QWRP, the objective is to find a *tour* (a polygonal cycle) $\gamma \subset P$ of minimum length $|\gamma|$ such that $|V(\gamma)| \geq A$; see Figure 1. Note that when $A = |P|$, the QWRP is the classic WATCHMAN ROUTE PROBLEM.

We also distinguish between the *anchored* version (in which γ must pass through a given depot point s) and the *floating* version (in which no depot is given). We provide the following NP-hardness results (proved in the full version) for both the anchored and floating cases:

▶ **Theorem 1.** *The QWRP in a simple polygon is weakly NP-hard, with or without an anchor.*

Throughout the paper, we assume a real RAM model of computation [28].

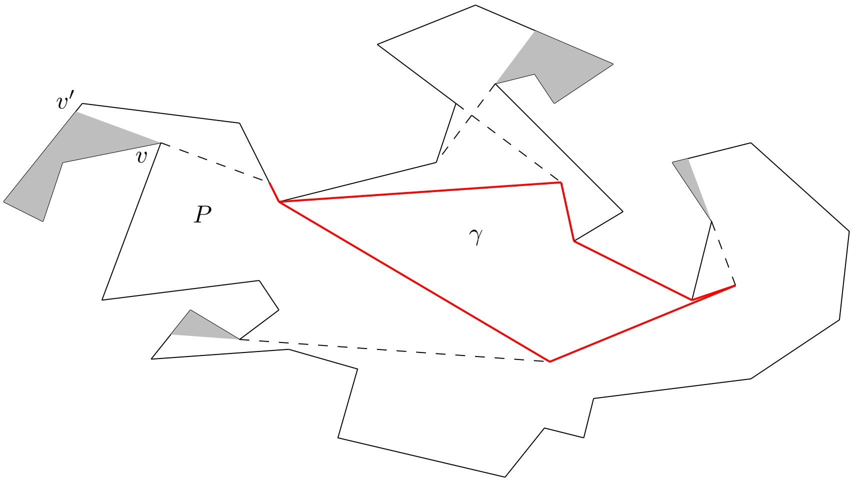


Figure 1 A route γ (red) that sees the white portion of P (the gray regions are unseen).

2.2 Structural Lemma

Let $\pi_P(x,y)$ denote the *geodesic shortest path* (shortest path constrained to stay within P) between $x \in P$ and $y \in P$; $\pi_P(x,y)$ is unique in a simple P, and is the segment xy if x sees y. For a subset $S \subseteq P$, the *relative/geodesic convex hull* of S is the minimal set that contains S and is closed under taking shortest paths. Equivalently, the relative convex hull of S is the minimum-perimeter connected subset of P that contains S. A set is *relatively convex* if it is equal to its relative convex hull, and a closed curve is relatively convex if it is the boundary of a relatively convex set. Let P_γ denote the connected region bounded by some closed polygonal chain γ. If P_γ is a (sub)polygon of P and P_γ is relatively convex, then P_γ is the relative convex hull of its convex vertices, and all reflex vertices of P_γ are necessarily reflex vertices of P. We similarly define relative convexity of an open polygonal chain γ: if γ is a connected subset of the boundary of the relative convex hull of γ, then we say that γ is relatively convex. Geodesic shortest paths and relative convex hulls have been studied extensively and can be computed efficiently [24].

An optimal solution to the QWRP in a simple polygon P satisfies a structural lemma:

▶ **Lemma 2.** *For a simple polygon P with n vertices, and no depot, an optimal QWRP tour is a relatively convex simple polygonal cycle of at most $2n$ vertices.*

Proof. Let γ be an optimal QWRP tour and let $P' = V(\gamma)$ be the visibility polygon of γ. Since γ is connected, P' is a simple subpolygon of P; some edges of P' coincide with edges of P and some are shadow chords (chords separating $V(\gamma)$ from the rest of P) supported by reflex vertices of P. Then γ is a shortest watchman route in the simple polygon P'. Thus, γ is relatively convex in P', and thus in P, and γ has at most $2n$ vertices, since P' is easily seen to have at most n vertices. (See [11, 25].)

Specifically, the polygon $P' = V(\gamma)$ is obtained from P by removing certain subpolygons ("shadow pockets") of P that are each defined by a chord, vv', extending from a reflex vertex, v, of P, along the line through v and a convex vertex of γ, to the first point v' on the boundary of P. This process introduces a (convex) vertex v' (on an edge of P, in general on its interior), and removes at least one vertex of P, on the boundary of the pocket that is cut off by the chord. Refer to Figure 1. Thus, P' has at most n vertices. For a simple polygon with n vertices, any shortest watchman route has at most $2n$ vertices [5, 10, 11, 13, 26, 29, 30]. Moreover, all reflex vertices of P' must be reflex vertices of P, hence all reflex vertices of γ must also be reflex vertices of P. ◀

If there is a specified depot $s \in \partial P$, a statement similar to Lemma 2 holds. If s is interior to P, an optimal tour $\gamma = (s, w_1, w_2, \dots, w_k, s)$ through s need not be relatively convex; however, it is "nearly" relatively convex in that the tour obtained by replacing the two edges sw_1 and $w_k s$ with the geodesic path $\pi_P(w_1, w_k)$ is relatively convex.

2.3 Dual approximation algorithm for anchored QWRP

An optimal tour for the QWRP will, in general, have (convex) vertices that are interior to P, at locations within the continuum that are not known to come from a discrete set. This poses a challenge to algorithms that are to compute solutions for the QWRP exactly or approximately. We address this challenge by discretizing an appropriate portion of the domain P using a (Steiner) triangulation whose faces are small enough that we can afford to round an optimal tour to vertices of the triangles, while increasing the length of the tour only slightly, and assuring that the rounded tour continues to see at least as much of P as the optimal tour did. We focus here on the anchored case, with a specified depot s, which we assume to be on ∂P for now.

First, we triangulate P (in $O(n)$ time [6]), including s as a vertex of the triangulation. We then overlay, centered on s, a regular square grid of pixels of side lengths δ within an axis-aligned square of size L-by-L for a length L that is at least the optimal tour length; we specify how to determine δ and L below. The overlay of the grid with the triangulation yields a partition of P into convex cells of constant complexity, each of which we triangulate, resulting in an overall Steiner triangulation of P, such that every triangle within distance $L/2$ of s has diameter at most $\sqrt{2}\delta$ and perimeter at most 4δ; we let $S_{\delta,L}$ denote the set of vertices of these triangles. We refer to $S_{\delta,L}$ as the set of *candidate turn points* for a route.

▶ **Lemma 3.** *For an optimal tour γ for the QWRP with area quota A, there exists a polygonal tour γ' whose vertices are in the set $S_{\delta,L}$ of candidates, such that γ' is relatively convex, $|\gamma'| \le |\gamma| + (8 + 4\sqrt{2})\delta n$ and $V(\gamma) \subseteq V(\gamma')$.*

Proof. Let $c_1, c_2, \dots$ be convex vertices of the optimal tour γ (γ is the relative convex hull of such vertices) and let $\sigma_1, \sigma_2, \dots$ be (closed) cells of the decomposition that contain the vertices. Let γ' be the boundary of the relative convex hull of the cells. By construction, γ' is a relatively convex tour enclosing γ, implying that any point seen by γ is also seen by γ'. Furthermore, since $s \in \partial P$, it follows that s cannot be in the interior of $P_{\gamma'}$, a subpolygon of P, thus $s \in \gamma'$.

We claim that $|\gamma'|$ is at most $|\gamma| + (8+4\sqrt{2})\delta n$. For each edge e' of γ' going from σ_i to σ_j, we can bound its length by the sum of the length of the edge e of γ going from σ_i to σ_j (γ' visits the cells containing the vertices of γ in the same order) and at most two connections from endpoints of e to vertices of σ_i, σ_j, which is no more than $2\sqrt{2}\delta$, see Figure 2, right. Additionally, the part of γ' along the perimeters of $\sigma_1, \sigma_2, \dots$ is no longer than $8\delta n$. Hence, $|\gamma'| \le |\gamma| + (8 + 4\sqrt{2})\delta n$. ◀

From Lemma 3, if $\delta = O\left(\frac{\varepsilon|\gamma|}{n}\right)$, then for approximation purposes within factor $(1+\varepsilon)$, it suffices to search for a tour whose vertices come from $S_{\delta,L}$. In fact, our algorithm returns a tour no longer than $(1+\varepsilon_1)|\gamma|$ for any $\varepsilon_1 > 0$; however, due to discretization of the area quota, we only guarantee the tour sees at least $(1-\varepsilon_2)A$ for any $\varepsilon_2 > 0$.

We now establish an ordering on the point set $S_{\delta,L}$, so that a relatively convex chain of the candidate points moves in increasing order. First, we compute $\mathcal{T}$, the tree of shortest paths rooted at s to all the candidate points; this takes $O(|S_{\delta,L}|)$ time [21]. The path from s to a candidate point s' in $\mathcal{T}$ is the geodesic shortest path $\pi_P(s, s')$. Define a *geodesic angular*

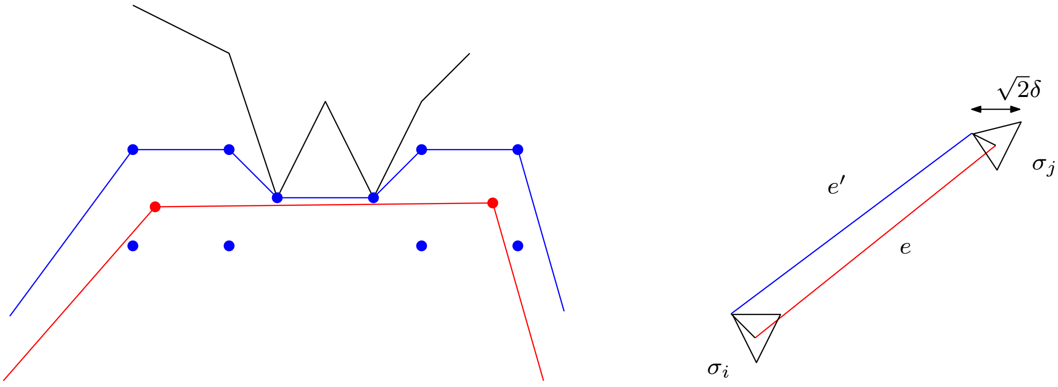


Figure 2 Left: γ' (blue) is the relative convex hull of the vertices (blue) of the cells that contain convex vertices of γ (red). Right: Each edge of γ' that traverses between two different cells σ_i, σ_j by triangle inequality, is no longer than the edge of γ between the same cells plus at most two connections to two vertices of σ_i, σ_j.

order as follows: for two candidate points s_i, s_j, if s_i is to the left of the extended geodesic shortest path between s and s_j, i.e $\pi_P(s, s_j)$ with the last segment extending up to ∂P, then s_i precedes s_j. In case of ties, we break ties by increasing distance to s. For each reflex vertex r_i of P, we add another candidate s_{r_i} to the list to account for the possibility that r_i can appear as two different vertices of a relatively convex polygonal chain; s_{r_i} obeys the aforementioned geodesic angular order but precedes every candidate point in the subtree of $\mathcal{T}$ rooted at r_i. Sort the candidates accordingly, then append $s_m := s_1$ to the end of the sorted list. Any relatively convex chain with vertices sequence oriented clockwise $(s, s_{i_1}, s_{i_2}, \ldots)$ has $1 < i_1 < i_2 < \ldots$. Without loss of generality, we consider any relatively convex polygonal chain to be oriented clockwise.

Next, we examine the optimal substructure of the problem.

▶ **Lemma 4** ([3]). *The visibility region of the geodesic shortest path $\pi_P(s, s_j)$ is the inclusion-wise minimal set among all visibility regions of all paths from s to s_j.*

Let C be a relatively convex polygonal chain from s to a candidate point s_j, and let s_i be the vertex of C immediately preceding s_j. We identify the overlap of visibility between the segment s_is_j and C_{s_i}, the subchain of C from s to s_i in Lemma 5.

▶ **Lemma 5.** $V(C_{s_i}) \cap V(s_is_j) = V(\pi_P(s, s_i)) \cap V(s_is_j)$.

Proof. Refer to Figure 3. Let $x \in P$ be a point seen by both $\pi_P(s, s_i)$ and s_is_j. Since $x \in V(\pi_P(s, s_i))$, it follows that $x \in V(C_{s_i})$ (Lemma 4). Thus, $x \in V(C_{s_i}) \cap V(s_is_j)$ and $V(\pi_P(s, s_i)) \cap V(s_is_j) \subseteq V(C_{s_i}) \cap V(s_is_j)$.

On the other hand, let $x \in P$ be seen by both C_{s_i} and s_is_j. Since $x \in V(C_{s_i}) \cap V(s_is_j)$ there exists $x_1 \in C_{s_i}$ and $x_2 \in s_is_j$ such that xx_1 and xx_2 are contained within P. Thus, the (pseudo)triangle xx_1x_2 is contained within P since P has no holes. By our ordering scheme, s_j is to the right of $\pi_P(s, s_i)$ with the last segment extended up to ∂P, while C_{s_i} is to the left of it. This implies that in the relatively convex polygon $P_{C \cup \pi_P(s,s_j)}$, x_1, x_2 are in opposite sides with respect to $\pi_P(s, s_i)$. As we pivot a line of sight around x from x_1 to x_2, it must intersect $\pi_P(s, s_i)$ at some point due to continuity as well as (relative) convexity, therefore $\pi_P(s, s_i)$ sees x. Hence, $V(C_{s_i}) \cap V(s_is_j) \subseteq V(\pi_P(s, s_i)) \cap V(s_is_j)$. ◀

Based on Lemma 5, the overlap of visibility between the segment s_is_j, for $i < j$, and any relatively convex chain C_{s_i} from s to s_i does not depend on the vertices between s and s_i. This leads to the optimal substructure utilized by our dynamic programming algorithm.

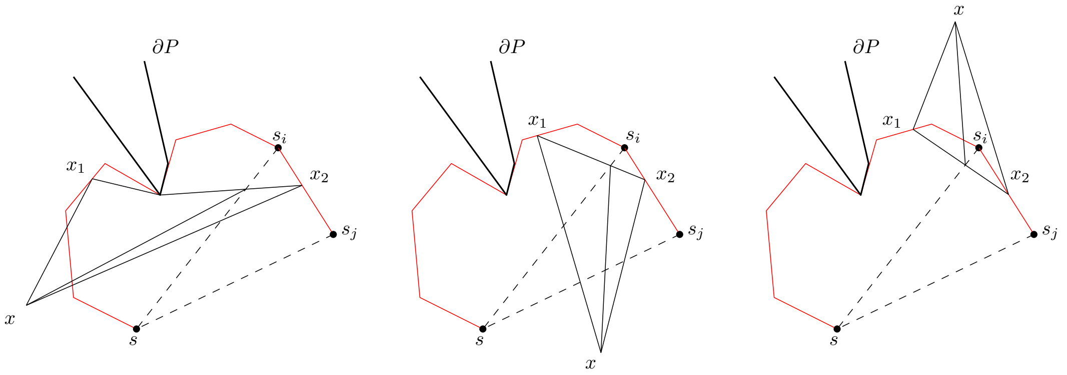


Figure 3 Proof of Lemma 5.

▶ **Lemma 6.** *C is a shortest relatively convex polygonal chain from s to s_j that sees at least some area $\overline{A}$ if and only if C_{s_i} is a shortest relatively convex polygonal chain from s to s_i that sees at least area $\overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|$.*

Proof. We write $V(C)$ as the union of 2 disjoint sets $V(C_{s_i})$ and $V(s_is_j) \setminus V(C_{s_i})$. Notice that

$$V(s_is_j) \setminus V(C_{s_i}) = V(s_is_j) \setminus (V(C_{s_i}) \cap V(s_is_j)) = V(s_is_j) \setminus (V(\pi_P(s, s_i)) \cap V(s_is_j)) \\ = V(s_is_j) \setminus V(\pi_P(s, s_i)),$$

therefore $|V(C_{s_i})| \geq \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|$. As a result, C_{s_i} must be the shortest chain to achieve a visibility area of $\overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|$, since the existence of a shorter chain contradicts the optimality of C, and vice versa. ◀

A subproblem in the dynamic program is determined by a candidate point s_j and an area quota $\overline{A}$. Let $\pi(s_j, \overline{A})$ denote the length of a shortest relatively convex polygonal chain from s to s_j that can see area at least $\overline{A}$; and let $C(s_j, \overline{A})$ denote the associated optimal chain. Initialize $\pi(s, |V(s)|) = 0$. The Bellman recursion for each subproblem with $j = 1, 2, \ldots, m$ and all values of $\overline{A}$ would be given as follows, for all $\bar{i} < j$ such that s_j sees $s_{\bar{i}}$ and $C(s_{\bar{i}}, \overline{A} - |V(s_{\bar{i}}s_j) \setminus V(\pi_P(s, s_{\bar{i}}))|) \cup s_{\bar{i}}s_j$ is relatively convex:

$$i = \arg\min_{\bar{i}} \left\{\pi(s_{\bar{i}}, \overline{A} - |V(s_{\bar{i}}s_j) \setminus V(\pi_P(s, s_{\bar{i}}))|) + |s_{\bar{i}}s_j|\right\},$$
$$\pi(s_j, \overline{A}) = \pi(s_i, \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|) + |s_is_j|,$$
$$C(s_j, \overline{A}) = C(s_i, \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|) \cup s_is_j.$$

Finally, return $C(s_m, A)$. Correctness of the algorithm follows from the principle of optimality.

Note that there always exists an optimal solution i to the above recursion such that $C(s_i, \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|) \cup s_is_j$ is relatively convex. Otherwise, we can shortcut $C(s_i, \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|) \cup s_is_j$ by connecting s_j to the closest reflex vertex (of P) or the point of tangency in $C(s_i, \overline{A} - |V(s_is_j) \setminus V(\pi_P(s, s_i))|)$.

Since $\overline{A}$ can take values from a continuous set, it is impractical to tabulate all such values. Instead, we bucket A into uniform intervals, and let the subproblems be defined by interval endpoints. We round down the area of any visibility polygon to the nearest interval. Since we sum up the area of at most $2(n-3) + \frac{2L}{\delta}$ visibility polygons (each of the $n-3$ diagonals in the triangulation of P and $\frac{L}{\delta}$ horizontal/vertical grid lines potentially has at most 2

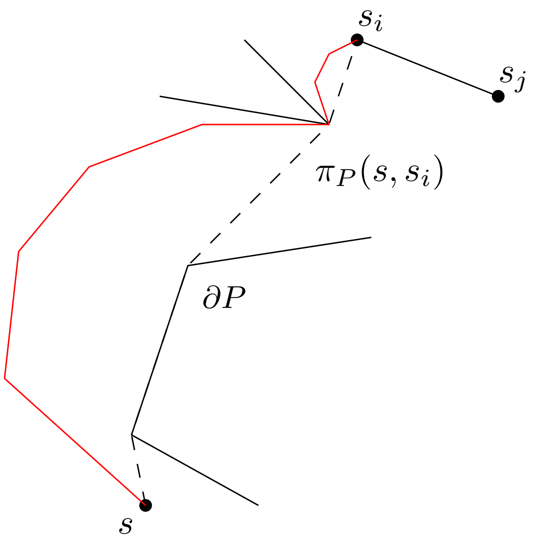


Figure 4 Solving subproblem $(s_j, \overline{A})$.

vertices of the tour returned by the dynamic programming algorithm since we enforce relative convexity), if we denote by I the length of each interval, the area lost by rounding down is at most $I\left(2(n-3)+\frac{2L}{\delta}\right)$. We run the algorithm on the "rounded down" instance with area quota $A \;-\; I\left(2(n-3)+\frac{2L}{\delta}\right)$, and since the optimal solution of the original instance is a feasible solution of the new instance, the algorithm returns a tour no longer than an optimal tour γ' (that sees at least area A).

It remains to compute an appropriate L such that an optimal tour γ is contained within the bounding box of the grid. Denote by $C_g(r)$ the geodesic disk of radius r centered at s (the locus of points whose length of the geodesic path to s is no greater than r). Let $r := r_{\min}$ where $r_{\min}$ is the smallest value of r such that $|V(C_g(r))| = A$; then, r is a lower bound on $|\gamma|$, since a tour of length r has geodesic radius at most $r/2$ and thus cannot see an area of A. We can compute r by the "visibility wave" methods in [1] by considering all $O(n^2)$ edges of the visibility graph G_v of P (nodes are vertices of P and two nodes are adjacent if they are visible to one another); we have a sequence of visibility edges hit by $C_g(r)$ for the first time in the process of increasing r, obtained by sorting the distance from every visibility edge to s in $O(n^2 \log n)$ time.

Moreover, $|\partial C_g(r)| + 2r$ is an upper bound on $|\gamma|$, since if the watchman goes from s to $\partial C_g(r)$ (s may not be on $\partial C_g(r)$), follows along $\partial C_g(r)$ then goes back to s, he sees an area of A. We argue that $|\partial C_g(r)| + 2r = O(nr)$ as follows: $\partial C_g(r)$ consists of polygonal chains that are portions of ∂P and circular arcs; the circular arcs have total length at most $2\pi r$. For each segment in the polygonal part of $\partial C_g(r)$, we can bound its length by the sum of geodesic distances from its endpoints to s (triangle inequality), which is no more than $2r$. There are at most n segments in the polygonal portions of $\partial C_g(r)$, therefore their total length is no longer than $2nr$, implying $|\gamma| \le |\partial C_g(r)| + 2r = 2nr + 2\pi r + 2r \le 6nr$.

We initialize $L := r$ and run the dynamic program with the following δ and I:

$$\delta = \frac{\varepsilon_1 L}{16 + 8\sqrt{2}n}, \qquad I = \frac{\varepsilon_1 \varepsilon_2 A}{2(n-3)\varepsilon_1 + (32 + 16\sqrt{2})n}.$$

Then, set $L := 2L$, and repeat until $L \ge 6nr$. At some point, we must have $|\gamma| \le L \le 2|\gamma|$, which means the approximate tour γ' returned by the dynamic program will be such that $V|\gamma'| \ge (1-\varepsilon_2)|V(\gamma)|$ and $|\gamma'| \le (1+\varepsilon_1)|\gamma|$. We return the shortest tour out of all tours that achieve the visibility area quota as we increase r. Since $r \leq |\gamma| = O(nr)$, the number of iterations of the doubling search is $O(\log n)$. The resulting theorem (proof details in the full version):

▶ **Theorem 7.** *Given a starting point s, a dual approximation γ' to an optimal solution γ of the QWRP, with area quota A, in a simple polygon with n vertices, satisfying $|V(\gamma')| \geq (1-\varepsilon_2)A$ and $|\gamma'| \leq (1+\varepsilon_1)|\gamma|$ for any $\varepsilon_1, \varepsilon_2 > 0$, can be computed in $O\left(\frac{n^5}{\varepsilon_1^5 \varepsilon_2} \log n\right)$ time if $s \in \partial P$ or $O\left(\frac{n^9}{\varepsilon_1^9 \varepsilon_2} \log n\right)$ time if $s \notin \partial P$.*

3 The BWRP in a Simple Polygon

▶ **Theorem 8** (proof in the full version). *The BWRP in a simple polygon is weakly NP-hard.*

3.1 Approximation algorithm for anchored BWRP

For a given budget length $B > 0$, and any fixed $\varepsilon > 0$, we compute a route of length at most $(1+\varepsilon)B$ that sees at least as much area as an area-maximizing route γ of length B.

▷ Claim 9. Without loss of generality, we can assume that B is less than the length of an optimal watchman route for P; otherwise, the solution is simply an optimal WRP tour. Hence, an optimal budgeted watchman route γ is necessarily the shortest watchman route of the subpolygon $V(\gamma)$, and so γ is relatively convex (otherwise, we can shortcut γ and expand the remaining length budget to see more area, contradicting the optimality of γ).

Since it suffices to consider only relatively convex routes for the BWRP, the observation in Lemma 5 allows us to prove the structure of an optimal BWRP tour.

▶ **Lemma 10.** *C is a relatively convex polygonal chain from s to s_j of length at most $\overline{B}$ that sees the largest area possible if and only if C_{s_i} is a relatively convex polygonal chain from s to s_i of length at most $\overline{B} - |s_i s_j|$ that sees the largest area possible.*

Proof. The proof is identical to that of Lemma 6 and hence omitted in this version. ◀

We decompose P by overlaying a triangulation and regular square grid of δ-sized pixels within an axis-aligned square of size B-by-B, centered on s, then sort the set of candidates $S_{\delta,B}$ according to the geodesic angular order defined for the QWRP. Similarly, there exists a route γ' of length at most $|\gamma| + (8+4\sqrt{2})\delta n$ with vertices in $S_{\delta,B}$.

Let $a(s_j, \overline{B})$ be the optimal area that a relatively convex polygonal chain from s to s_j that is no longer than $\overline{B}$ can see; and let $C(s_j, \overline{B})$ be the associated optimal chain of the subproblem $(s_j, \overline{B})$. Initialize $a(s, 0) = |V(s)|$. The Bellman recursion for each subproblem with $j = 1, 2, \ldots, m$ and all values of $\overline{B}$ would be given as follows, for all $\bar{i} < j$ such that s_j sees $s_{\bar{i}}$ and $C(s_{\bar{i}}, \overline{B} - |s_{\bar{i}} s_j|) \cup s_{\bar{i}} s_j$ is relatively convex

$$\begin{aligned} i &= \arg\max_{\bar{i}} \left\{ a(s_{\bar{i}}, \overline{B} - |s_{\bar{i}} s_j|) + |V(s_{\bar{i}} s_j) \setminus V(\pi_P(s, s_{\bar{i}}))| \right\} \\ a(s_j, \overline{B}) &= a(s_i, \overline{B} - |s_i s_j|) + |V(s_i s_j) \setminus V(\pi_P(s, s_i))|, \\ C(s_j, \overline{B}) &= C(s_i, \overline{B} - |s_i s_j|) \cup s_i s_j. \end{aligned}$$

Then, return $\gamma' := C(s_m, B + (8+4\sqrt{2})\delta n)$.

To bound the number of subproblems, we consider a partition of an interval of length $B + (8+4\sqrt{2})\delta n$ into uniform intervals, and round up the length of any segment $s_i s_j$ to the nearest interval endpoint. Let I be the length of each interval, we run the algorithm on a new instance with budget $B + (8 + 4\sqrt{2})\delta n + \left(2(n-3) + \frac{2B}{\delta}\right) I$ and the subproblems defined by intervals' endpoints. The optimal solution of the original instance is a feasible solution of the new instance; thus, we find a route seeing as much as the optimal route of the original instance. The values of δ and I can be set as follows:

$$\delta = \frac{\varepsilon B}{(16 + 8\sqrt{2})n}, \qquad I = \frac{\varepsilon^2 B}{4(n-3)\varepsilon + (64 + 32\sqrt{2})n},$$

so that $B + (8 + 4\sqrt{2})\delta n + \left(2(n-3) + \frac{2B}{\delta}\right) I \leq (1+\varepsilon)B$. In the full paper we prove:

▶ **Theorem 11.** *Given a starting point s, a route of length at most $(1+\varepsilon)B$ seeing at least as much area as is seen by an optimal route of length B for the BWRP in a simple polygon with n vertices can be computed in $O\left(\frac{n^5}{\varepsilon^6}\right)$ time if $s \in \partial P$ or $O\left(\frac{n^9}{\varepsilon^{10}}\right)$ time if $s \notin \partial P$.*

3.2 From anchored BWRP to an FPTAS for anchored QWRP

We can adapt the algorithm for the anchored BWRP above to obtain an FPTAS for the anchored QWRP. Let γ be an optimal QWRP tour, suppose we have some L such that $|\gamma| \leq L \leq 2|\gamma|$. We divide L into $\frac{L}{\lceil \frac{2}{\varepsilon} \rceil}$ uniform intervals, each of length no greater than $\varepsilon|\gamma|$. The smallest interval endpoint L' that is no smaller than $|\gamma|$, is a $(1+\varepsilon)$-approximation to $|\gamma|$. We can iterate through interval endpoints as the budget constraint and use the approximation algorithm for the BWRP to compute a route of length at most $(1+\varepsilon)L'$ that sees as much as an area-maximizing route of length L' does, which sees more area than does γ. The result is the following theorem, which, in contrast with the earlier Theorem 7, is not a dual approximation (allowing for a relaxation of the quota constraint), but an FPTAS for optimizing the length, subject to a hard constraint on the area seen. The running time of the algorithm in the dual approximation of Theorem 7, however, is better than that of the FPTAS, so it may be preferred in some settings.

▶ **Theorem 12** (proof in the full version). *Given a starting point s, an approximation γ' to an optimal solution γ of the QWRP, with area quota A, in a simple polygon with n vertices, satisfying $|V(\gamma')| \geq A$ and $|\gamma'| \leq (1+\varepsilon)|\gamma|$ for any $\varepsilon > 0$, can be computed in $O\left(\frac{n^5}{\varepsilon^6} \log n \log \frac{1}{\varepsilon}\right)$ if $s \in \partial P$ or $O\left(\frac{n^9}{\varepsilon^{10}} \log n \log \frac{1}{\varepsilon}\right)$ if $s \notin \partial P$.*

4 Floating QWRP and BWRP

When the starting point s of the tour is not specified (the so called "floating" case), the WRP tends to be trickier: known algorithms for the floating WRP are $O(n)$-factor slower than in the non-floating case [10, 5, 11, 30, 13, 26, 29]. If the optimal tour is not convex (but only relatively convex), one can iterate through all reflex vertices of P as choices for s, and thus reduce the floating version to the basic WRP; the same can be done for QWRP and BWRP. Thus, the remaining challenge is to find the shortest (strictly, not just relatively) convex tour.

Any convex polygon can be outer-approximated by a convex polygon with a constant number of vertices: Dudley's approximation [2, 14, 23] implies that for any length-L tour γ, there is a length-$(1+\varepsilon)L$ tour γ' with $O\left(\frac{1}{\sqrt{\varepsilon}}\right)$ vertices that sees at least as much area as γ does. To find γ' (either for QWRP or BWRP), we use the techniques from [26]: For each of the $O(n^4)$ cells of the visibility decomposition D (the visibility graph G_v of P as defined Section 2.3, with maximally extended edges), points within the cell have visibility polygons that are combinatorially equivalent, implying that the area seen by any point in

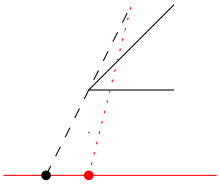


Figure 5 What is seen from the interior of a segment does not change as the segment is moved locally: whatever was hidden, but becomes seen from an interior point (red), was seen by a neighboring point (black) before the move.

the cell is given by the same formula. Moreover, if each vertex of a tour sits within a fixed cell of D and each edge of the tour passes through the same set of cells, the total area seen from the tour is given by the same formula: the interiors of the edges do not add to the area seen, so the total seen area is a function, $f(v_1, \ldots, v_k)$, of only the positions v_i of the tour's $k = O\left(\frac{1}{\sqrt{\varepsilon}}\right)$ vertices (Fig. 5). We further decompose the cells of D by lines through all of D's vertices. If the vertices of the tour are in the same cells of this refined $O(n^8)$-complex decomposition D', then the edges of the tour pass through the same cells of D. We iterate through all $O(n^{8k})$ placements of vertices of the tour into cells of D'. For each placement, finding $\vec{v}$ maximizing the seen area, $f(\vec{v})$, amounts to solving an $O(k)$-sized system of polynomial equations having $O(k)$ algebraic degree (the Lagrangian of the problem will contain the constraint that the tour's length is L, consisting of k terms and will have to be squared $O(k)$ times before becoming a polynomial). The solution can be found in $k^{O(k)}$ time [4, Section 3.4]. We summarize in the following theorems:

▶ **Theorem 13.** *Let γ be an optimal QWRP (no starting point) tour. In $n^{O\left(\frac{1}{\sqrt{\varepsilon}}\right)}$ time, a tour of length at most $(1+\varepsilon)|\gamma|$ can be found that sees at least as much area as does γ.*

▶ **Theorem 14.** *In $n^{O\left(\frac{1}{\sqrt{\varepsilon}}\right)}$ time, a BWRP (no starting point) tour of length $(1+\varepsilon)L$ can be found that sees at least as much area as does any tour of length L.*

5 Domains that are a Union of Lines

We consider the QWRP and the BWRP in a domain P that is a connected union (arrangement) of lines; it suffices to truncate the lines within a bounding box that encloses all vertices of the line arrangement, so that P is bounded. Such domains, which are a special case of polygons with holes, have been studied in the context of the Watchman Route Problem [15, 16]. In this setting, the QWRP seeks to minimize the length of a route contained within P that visits at least a specified number of (truncated) lines, and the BWRP seeks to maximize the number of lines visited by a route within P of length at most some budget. In this section we do not assume a depot s is specified.

For a set of lines $\mathcal{L}$, let $\mathcal{A}(\mathcal{L})$ denote the arrangement formed by $\mathcal{L}$. We assume that not all of the lines are parallel so that the union is connected. All of a line can be seen from any point incident on it. Let $G(\mathcal{L})$ denote the weighted planar graph with vertex $V(\mathcal{A}(\mathcal{L}))$ of intersections between lines. Two vertices in the graph are connected by an edge with Euclidean weight if they share the same edge in $\mathcal{L}$. Since the watchman is constrained to travel within $\mathcal{A}(\mathcal{L})$ and the only time the route can have turning points not in $V(\mathcal{A}(\mathcal{L}))$ is when it traces out the same edge of $G(\mathcal{L})$ consecutively in opposite directions, turning somewhere interior to that edge. However, then we can shortcut that portion of the route altogether while maintaining visibility coverage, it is easy to see that any optimal route for BWRP or QWRP is polygonal and its vertices is a subset of $V(\mathcal{A}(\mathcal{L}))$.

We first explain our results for the QWRP, then we use them to solve the BWRP. The following observation is essential to our algorithm: a line intersects a tour γ if and only if it intersects the convex hull of γ. We define the *quota intersecting convex hull* problem as follows: compute a cyclic sequence $(v_1, v_2, \ldots, v_h)$ of vertices $v_i \in V(\mathcal{A}(\mathcal{L}))$ in convex position such that the number of lines intersecting the convex polygon $(v_1, v_2, \ldots, v_h)$ is at least some specified $Q > 0$ and $\sum_i^h |\pi(v_i, v_{i+1})|$ is minimized $(v_{h+1} = v_1)$, where $\pi(s,t) = \pi_G(s,t)$ is the shortest path connecting s and t in $G(\mathcal{L})$. We show the relationship between the quota intersecting convex hull problem and the QWRP in an arrangement of lines.

▶ **Lemma 15.** *An optimal solution to the quota intersecting convex hull problem yields an optimal solution to the QWRP in an arrangement of lines.*

Proof. Suppose $(v_1, v_2, \ldots, v_h)$ is an optimal solution to the quota intersecting convex hull problem of length L intersecting Q lines. We concatenate $\pi(v_1, v_2), \ldots, \pi(v_{h-1}, v_h)$ and $\pi(v_h, v_1)$ to form γ. Every line intersecting the convex polygon $(v_1, v_2, \ldots, v_h)$ must intersect γ as well. Thus, γ is a route of length $\sum_i^h |\pi(v_i, v_{i+1})| = L$ seeing Q lines.

We claim that there is no solution γ' to the QWRP intersecting Q lines that is strictly shorter than γ. Suppose to the contrary, take the convex hull of γ', which has vertices in $V(\mathcal{A}(\mathcal{L}))$ since vertices of γ' are in $V(\mathcal{A}(\mathcal{L}))$. The vertices of the convex hull of γ' form a cyclic sequence that is feasible for the quota intersecting convex hull problem, and the length is exactly $|\gamma'|$ (or γ' could be shortened while still intersecting Q lines), which is strictly smaller than L. Thus, γ' yields a feasible cyclic sequence intersecting Q lines while the length is shorter than γ, violating the assumption that $(v_1, v_2, \ldots, v_h)$ is optimal. ◀

Note that there can be many optimal solutions to the quota intersecting convex hull problem yielding the same tour (Figure 6).

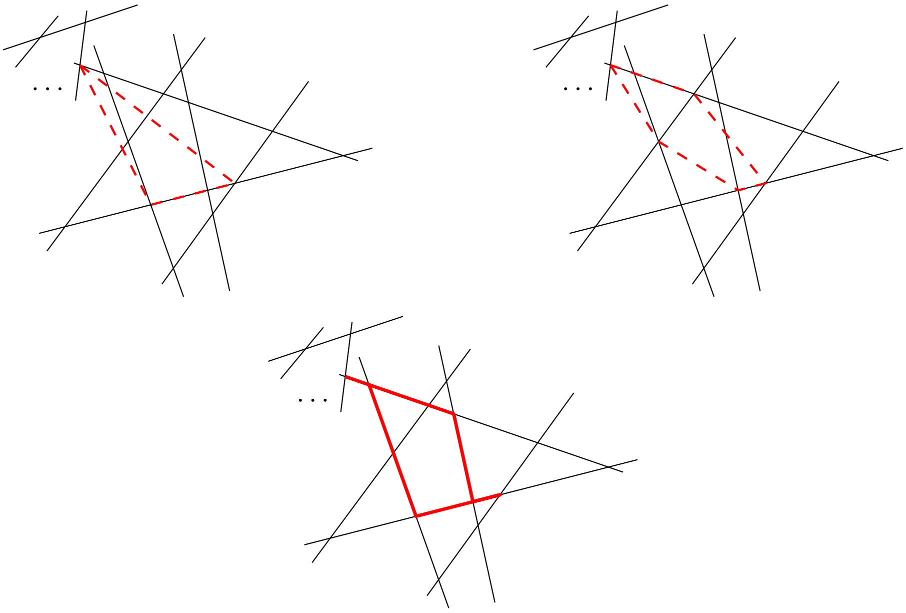


Figure 6 Multiple optimal solutions to the quota intersecting convex hull problem corresponding to the same optimal solution of the QWRP.

We give a dynamic programming algorithm to solve the quota intersecting convex hull problem. Fix one vertex to be the lowest vertex, let that vertex be v_1. We will examine all possible choices of v_1, and find the optimal cyclic sequence with each choice. Let $\{v_2, v_3, \dots, v_{m-1}\}$ be the list of vertices above v_1 sorted by increasing angle with the left horizontal ray passing through v_1, breaking ties by increasing distance to v_1. Then, set a new element $v_m := v_1$ and append it to the list. Thus, an optimal cyclic sequence $(v_1, v_{i_2}, v_{i_3}, \dots, v_m)$ has $1 < i_2 < i_3 < \dots < m$. We can restrict ourselves to ordered pairs (v_i, v_j) of consecutive vertices where $1 \le i < j \le m$ and either $i \ne 1$ or $j \ne m$ in the sequence.

For $1 \le i < j \le m$ and either $i \ne 1$ or $j \ne m$, denote by $\mathcal{L}_{i,j}$ the lines that intersect the segment $v_i v_j$ (including at the endpoints v_i, v_j). Each vertex v_j and a quota value $\overline{Q}$ define a subproblem. Let $\pi(v_j, \overline{Q})$ be the shortest length of a sequence of vertices in convex position from $(v_1, \dots, v_j)$ intersecting at least $\overline{Q}$ lines. Starting from v_1, we initialize $\pi(v_1, |\mathcal{L}_{1,1}|) = 0$ with the associated sequence (v_1). For $j = 2, \dots, m$ and $\overline{Q} = 1, 2, \dots, n$, we solve the subproblems $(v_j, \overline{Q})$ by the following Bellman recursion, for all $i < j$ such that the sequence associated with $(v_i, \overline{Q} - |\mathcal{L}_{i,j} \setminus \mathcal{L}_{1,i}|)$ and v_j are in convex position (Figure 7)

$$\pi(v_j, \overline{Q}) = \min_i \left\{ \pi(v_i, \overline{Q} - |\mathcal{L}_{i,j} \setminus \mathcal{L}_{1,i}|) + |\pi(v_i, v_j)| \right\}.$$

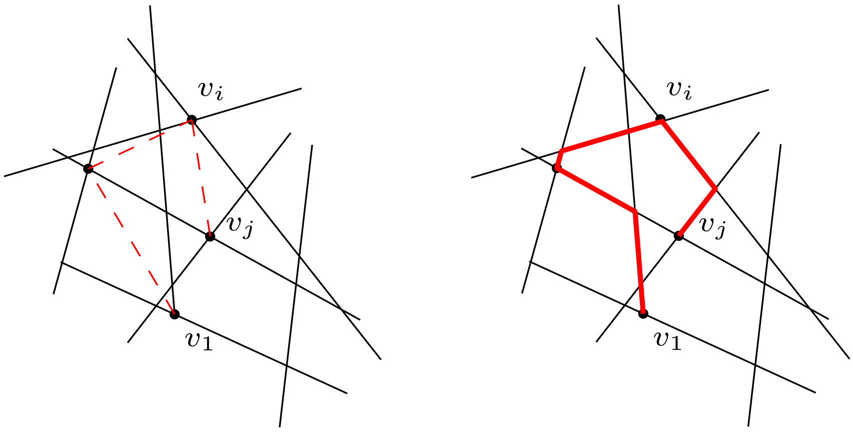


Figure 7 Solving subproblem $(v_j, \overline{Q})$. The sequence of vertices in convex position is drawn on the left with a dashed red chain, and the corresponding part of γ is drawn with solid red segments on the right.

Correctness of the algorithm follows from these two claims:

- Given a sequence of vertices in convex position $(v_1, \dots, v_i, v_j)$, any line intersecting both $\pi(v_i, v_j)$ and the subsequence from v_1 to v_i, $(v_1, \dots, v_i)$ must intersect the segment $v_1 v_i$ and vice versa due to continuity and convexity. If a sequence $(v_1, \dots, v_i, v_j)$ is the shortest among all sequences from v_1 to v_j intersecting at least $\overline{Q}$ lines, then the subsequence from v_1 to v_i is the shortest sequence from v_1 to v_i intersecting $\overline{Q} - |\mathcal{L}_{i,j} \setminus \mathcal{L}_{1,i}|$.
- The sequence $(v_1, \dots, v_i)$ associated with optimal solution i to the Bellman recursion is such that $(v_1, \dots, v_i) \cup (v_j)$ are in convex position.

▶ **Theorem 16** (proof in the full version). *The QWRP and the BWRP in an arrangement of lines can be solved in $O(n^7)$ time.*

▶ Remark 17. When $Q = n$, the algorithm solves the classic WRP in an arrangement of lines, thus our result improves upon the $O(n^8)$ solution in [16].

6 The QWRP and BWRP in a Polygon With Holes

6.1 Hardness of approximation

▶ **Theorem 18** (proof in the full version). *The QWRP in a polygon with holes cannot be approximated, in polynomial time, within a factor of* $c \log n$ *for some constant* $c > 0$*, unless* $P = NP$.

▶ **Theorem 19** (proof in the full version). *The BWRP in a polygon with holes cannot be approximated, in polynomial time, within a factor of* $(1 - \varepsilon)$ *for arbitrary* $\varepsilon > 0$*, unless* $P = NP$.

6.2 Approximation algorithm for the BWRP in a polygon with holes

We decompose P into small convex cells and obtain the set of candidates $S_{\delta,B}$ as in the case with the BWRP in a simple polygon.

▶ **Theorem 20.** *There exists a route* γ' *whose vertices are a subset of* $S_{\delta,B}$ *such that* $V(\gamma) \subseteq V(\gamma')$ *and* $|\gamma'| \leq (1 + \varepsilon)B$.

Proof. Consider an edge e of γ whose endpoints lie in cells σ_i and σ_j. We append $\partial\sigma_i$ and $\partial\sigma_j$ to γ. If the endpoints of e are not vertices of σ_i and σ_j, we replace e with a set of edges whose endpoints are candidate points. The procedure can be described with a physical analog: we slide an elastic string between the two intersection points of e with σ_i and σ_j towards the exterior of γ until the two endpoints coincide with vertices of σ_i and σ_j, the string is pulled taut and never passes through a hole; see Figure 8. The result is a geodesic path e' between two vertices of σ_i and σ_j that is no longer than $|e| + 2\sqrt{2}\delta$.

Repeating the process for every edge of γ, we obtain γ'. If a point x seen by γ is inside of $P_{\gamma'}$, the extended line of vision between x and γ must intersect with γ' since there is no hole between γ and γ'. If x is outside of $P_{\gamma'}$, then the line of vision between x and γ must intersect γ' due to the Jordan Curve Theorem. Thus, γ' sees everything that γ sees, moreover γ' passes through s, a vertex in the decomposition. Since γ has $O(n^2)$ vertices [25], for an appropriate choice of $\delta = O\left(\frac{\varepsilon B}{n^2}\right)$ we have $|\gamma'| \leq (1 + \varepsilon)B$. ◀

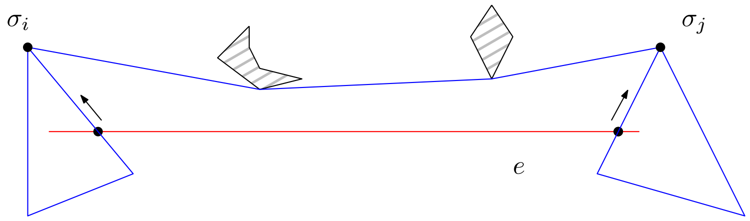


Figure 8 Replacing each edge (red) with the perimeters of the two cells containing its endpoints and a geodesic path of the same homotopy type (blue).

We apply a known result for the SUBMODULAR ORIENTEERING problem [7]: Given a weighted directed graph G, two nodes s and t (which need not be distinct), a budget $B > 0$, and a monotone submodular reward function defined on the nodes, find an s-t walk that maximizes the reward, under the constraint that the length of the walk is no greater than B.

Let G_1 be the visibility graph on the candidates set with Euclidean edge weights. Let G_2 be the line graph of G_1: nodes of G_2 correspond to edges of G_1, and two nodes in G_2 are adjacent if their respective edges in G_1 are incident. The weight of an edge of G_2 is the sum of the weights of the two edges in G_1 corresponding to its endpoints, divided by two, thus a closed walk of length B in G_1 corresponds to a closed walk of length B in G_2 and vice versa. We apply the approximation algorithm from [7] on G_2 to compute a closed walk from any node in G_2 corresponding to an edge incident with s, with the area of visibility as the reward function and budget $(1+\varepsilon)B$. The reason for using the line graph G_2 is that, in the Submodular Orienteering problem, rewards are associated with nodes, while in the context of the BWRP, rewards are accumulated when traversing edges of the visibility graph G_1. We obtain the following:

▶ **Theorem 21** (proof in the full version). *Given a polygon P with holes with n vertices, let $\beta \geq 2$ be any constant of choice and OPT be the maximum area that a route of length B can see. The BWRP has a dual approximation algorithm that computes a tour of length at most $(1+\varepsilon)B$ that sees an area of at least $\Omega\left(\frac{OPT \log \beta}{\log n}\right)$, with running time $\left(\frac{n}{\varepsilon}\log B\right)^{O\left(\beta \log \frac{n}{\varepsilon} / \log \beta\right)}$.*

7 Optimal Visibility-based Search for a Randomly Distributed Target

Our results can be applied to solve two problems of searching a randomly distributed static target in a simple polygon P: Given a prior distribution of the target's location in P, (1) compute a route that achieves a given detection probability within the minimum amount of time, where the target is detected if the watchman can see it; and (2) (dual to (1)) for a given time budget T, compute a search route maximizing the probability of detecting the target by time T. Denote by $\mu(.)$ the probability measure on all subsets of P; $\mu(P_1)$ is the probability measure of $P_1 \subseteq P$, i.e the probability that the target is in P_1, then

- $0 \leq \mu(.) \leq 1, \mu(\varnothing) = 0, \mu(P) = 1,$
- $\mu(P_1 \cup P_2) = \mu(P_1) + \mu(P_2)$ if $P_1 \cap P_2 = \varnothing$.

We assume that we have access to $\mu(.)$ via an oracle: Given a triangular region in P, the oracle returns its probability measure in $O(1)$ time. Thus, for a point or a segment, the probability measure of its visibility region can be computed in $O(n)$ time. Furthermore, if the watchman has constant speed, a time constraint/objective is equivalent to that of length. An optimal search route for each problem can be computed using the algorithms given with probability measure instead of area.

References

1 Esther M. Arkin, Alon Efrat, Christian Knauer, Joseph S. B. Mitchell, Valentin Polishchuk, Günter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016.

2 Efim M. Bronshteyn and L. D. Ivanov. The approximation of convex sets by polyhedra. *Siberian Mathematical Journal*, 16(5):852–853, 1975.

3 Kevin Buchin, Valentin Polishchuk, Leonid Sedov, and Roman Voronov. Geometric secluded paths and planar satisfiability. In *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

4 John Canny. *The Complexity of Robot Motion Planning*. MIT press, 1988.

5 Svante Carlsson, Håkan Jonsson, and Bengt J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22:377–402, 1999.

6 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

7 Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 245–253. IEEE, 2005.

8 Ke Chen and Sariel Har-Peled. The Euclidean orienteering problem revisited. *SIAM Journal on Computing*, 38(1):385–397, 2008.

9 Otfried Cheong, Alon Efrat, and Sariel Har-Peled. Finding a guard that sees most and a shop that sells most. *Discrete & Computational Geometry*, 37:545–563, 2007.

10 Wei-Pang Chin and Simeon Ntafos. Optimum watchman routes. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 24–33, 1986.

11 Wei-Pang Chin and Simeon Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991.

12 Timothy H. Chung, Geoffrey A. Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous Robots*, 31:299–316, 2011.

13 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 473–482, 2003.

14 Richard M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.

15 Adrian Dumitrescu, Joseph S. B. Mitchell, and Paweł Żyliński. The minimum guarding tree problem. *Discrete Mathematics, Algorithms and Applications*, 6(01):1450011, 2014.

16 Adrian Dumitrescu, Joseph S. B. Mitchell, and Paweł Żyliński. Watchman routes for lines and line segments. *Computational Geometry*, 47(4):527–538, 2014.

17 Adrian Dumitrescu and Csaba D. Tóth. Watchman tours for polygons with holes. *Computational Geometry*, 45(7):326–333, 2012.

18 James N. Eagle. The optimal search for a moving target when the search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

19 James N. Eagle and James R. Yee. An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Operations Research*, 38(1):110–114, 1990.

20 Lee-Ad Gottlieb, Robert Krauthgamer, and Havana Rika. Faster algorithms for orienteering and k-tsp. *Theoretical Computer Science*, 914:73–83, 2022.

21 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 1–13, 1986.

22 Leonidas J. Guibas, Jean-Claude Latombe, Steven M. LaValle, David Lin, and Rajeev Motwani. Visibility-based pursuit-evasion in a polygonal environment. In *Algorithms and Data Structures: 5th International Workshop, WADS'97 Halifax, Nova Scotia, Canada August 6–8, 1997 Proceedings 5*, pages 17–30. Springer, 1997.

23 Sariel Har-Peled and Mitchell Jones. Proof of Dudley's convex approximation. *arXiv preprint*, 2019. `arXiv:1912.01977`.

24 Joseph S. B. Mitchell. Geometric shortest paths and network optimization. *Handbook of Computational Geometry*, 334:633–702, 2000.

25 Joseph S. B. Mitchell. Approximating watchman routes. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 844–855, 2013.

26 Simeon Ntafos and Markos Tsoukalas. Optimum placement of guards. *Information Sciences*, 76(1-2):141–150, 1994.

27 Joseph O'Rourke. Visibility. In *Handbook of Discrete and Computational Geometry*, pages 875–896. Chapman and Hall/CRC, 2017.

28 Michael Ian Shamos. *Computational Geometry.* Yale University, 1978.

29 Xuehou Tan. Fast computation of shortest watchman routes in simple polygons. *Information Processing Letters*, 77(1):27–33, 2001.

30 Xuehou Tan, Tomio Hirata, and Yasuyoshi Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *International Journal of Computational Geometry & Applications*, 9(03):319–323, 1999.

31 K. E. Trummel and J. R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.

Search-Space Reduction via Essential Vertices Revisited: Vertex Multicut and Cograph Deletion

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

Ruben F. A. Verhaegh
Eindhoven University of Technology, The Netherlands

Abstract

For an optimization problem Π on graphs whose solutions are vertex sets, a vertex v is called *c-essential* for Π if all solutions of size at most $c \cdot \text{OPT}$ contain v. Recent work showed that polynomial-time algorithms to detect c-essential vertices can be used to reduce the search space of fixed-parameter tractable algorithms solving such problems parameterized by the size k of the solution. We provide several new upper- and lower bounds for detecting essential vertices. For example, we give a polynomial-time algorithm for 3-Essential detection for Vertex Multicut, which translates into an algorithm that finds a minimum multicut of an undirected n-vertex graph G in time $2^{\mathcal{O}(\ell^3)} \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of vertices in an optimal solution that are *not* 3-essential. Our positive results are obtained by analyzing the integrality gaps of certain linear programs. Our lower bounds show that for sufficiently small values of c, the detection task becomes NP-hard assuming the *Unique Games Conjecture*. For example, we show that $(2 - \varepsilon)$-Essential detection for Directed Feedback Vertex Set is NP-hard under this conjecture, thereby proving that the existing algorithm that detects 2-essential vertices is best-possible.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Linear programming; Theory of computation → Rounding techniques; Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractability, essential vertices, integrality gap

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.28

Related Version *Full Version*: https://arxiv.org/abs/2404.09769

Funding *Bart M. P. Jansen*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

1 Introduction

Preprocessing is an important tool for dealing with NP-hard problems. The idea is that before starting a time-consuming computation on an input, one first exhaustively applies simple transformation steps that provably do not affect the desired output, but which make the subsequently applied solver more efficient. Preprocessing is often highly effective in practice [1, 34].

There have been several attempts to theoretically explain the speed-ups obtained by preprocessing. The concept of kernelization [12, 14], phrased in the language of parameterized complexity theory [9, 10], is one such attempt. Recently, Bumpus, Jansen, and de Kroon [4] proposed an alternative framework for developing and analyzing polynomial-time preprocessing algorithms that reduce the search space of subsequently applied algorithms for NP-hard graph problems. They presented the first positive and negative results in this framework, which revolves around the notion of so-called *c-essential vertices.* In this paper, we revisit this notion by providing new preprocessing results and new hardness proofs.

To be able to discuss our results, we first introduce and motivate the concept of c-essential vertices and the corresponding algorithmic preprocessing task. Our results apply to optimization problems on graphs in which the goal is to find a minimum-size vertex set that hits all obstacles of a certain kind. The (Undirected) Vertex Multicut problem is a prime example. Given an undirected graph G, annotated by a collection $\mathcal{T}$ consisting of pairs of terminal vertices, the goal is to find a minimum-size vertex set whose removal disconnects all terminal pairs. The decision version of this problem is NP-complete, but *fixed-parameter tractable* parameterized by the size of the solution: there is an algorithm by Marx and Razgon [26] that, given an n-vertex instance together with an integer k, runs in time $2^{\mathcal{O}(k^3)} \cdot n^{\mathcal{O}(1)}$ and outputs a solution of size at most k, if one exists. The running time therefore scales exponentially in the size of the solution, but polynomially in the size of the graph. This yields a great potential for preprocessing: if an efficient preprocessing phase manages to identify some vertices $S \subseteq V(G)$ that are guaranteed to be part of an optimal solution, then finding a solution of size k in G reduces to finding a solution of size $k - |S|$ in $G - S$, thereby reducing the running time of the applied algorithm and its search space. To be able to give guarantees on the amount of search-space reduction achieved, the question becomes: under which conditions can a polynomial-time preprocessing algorithm identify vertices that belong to an optimal solution?

Essential vertices. The approach that Bumpus et al. [4] take when answering this question originates from the idea that it may be feasible to detect vertices as belonging to an optimal solution when they are *essential* for making an optimal solution. This is formalized as follows. For a real number $c \geq 1$ and fixed optimization problem Π on graphs whose solutions are vertex subsets, a vertex v of an input instance G is called *c-essential* if vertex v is contained in all c-approximate solutions to the instance. Hence a c-essential vertex is not only contained in all optimal solutions, but even in all solutions whose size is at most $c \cdot \textsc{opt}$. To obtain efficient preprocessing algorithms with performance guarantees, the goal then becomes to develop polynomial-time algorithms to detect c-essential vertices in the input graph, when they are present.

For some problems like Vertex Cover (in which the goal is to find a minimum-size vertex set that intersects each edge), it is indeed possible to give a polynomial-time algorithm that, given a graph G, outputs a set S of vertices that is part of an optimal vertex cover and contains all 2-essential vertices. For optimization problems whose structure is more intricate, like Odd Cycle Transversal, finding c-essential vertices from scratch still seems like a difficult task. Bumpus et al. [4] therefore formulated a slightly easier algorithmic task related to detecting essential vertices and proved that solving this simpler task is sufficient to be able to achieve search-space reduction. For a vertex hitting set problem Π whose input is a (potentially annotated) graph G and whose solutions are vertex sets hitting all (implicitly defined) constraints, we denote by $\textsc{opt}_\Pi(G)$ the cardinality of an optimal solution to G. The detection task is formally defined as follows, for each real $c \geq 1$.

c-Essential detection for Π
Input: A (potentially annotated) graph G and integer k.
Task: Find a vertex set $S \subseteq V(G)$ such that:
G1 if $\textsc{opt}_\Pi(G) \leq k$, then there is an optimal solution in G containing all of S, and
G2 if $\textsc{opt}_\Pi(G) = k$, then S contains all c-essential vertices.

The definition above simplifies the detection task by supplying an integer k in addition to the input graph, while only requiring the algorithm to work correctly for certain ranges of k. The intuition is as follows: when k is correctly guessed as the size of an optimal solution, the preprocessing algorithm should find all c-essential vertices, and is allowed to find additional vertices as long as they are part of an optimal solution. Bumpus et al. [4] give a *dove-tailing*-like scheme that manages to use algorithms for c-Essential detection for Π to give improved fixed-parameter tractable running times for solving Π from scratch. The exponential dependence of the running time of the resulting algorithm is not on the *total* size of the solution, but only on the number of vertices in the solution that are *not* c-essential. Hence their results show that large optimal solutions can be found efficiently, as long as they are composed primarily out of c-essential vertices. For example, they prove that a minimum vertex set intersecting all odd cycles (a solution to Odd Cycle Transversal) can be computed in time $2.3146^{\ell} \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of vertices in an optimal solution that are not 2-essential and which are therefore avoided by at least one 2-approximation. Apart from polynomial-time algorithms for c-Essential detection for Π for various combinations of Π and c, they also prove several *lower bounds*. One of their main lower bounds concerns the Perfect Deletion problem, whose goal is to obtain a perfect graph by vertex deletions [18]. They rule out the existence of a polynomial-time algorithm for c-Essential detection for Perfect Deletion for any $c \geq 1$, assuming FPT $\neq$ W[1]. (They even rule out detection algorithms running in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f.)

We continue exploring the framework of search-space reduction by detecting essential vertices, from two directions. We provide both upper bounds (new algorithms for c-Essential detection for Π) as well as lower bounds. We start by discussing the upper bounds.

Our results: Upper bounds. The Vertex Multicut problem is the subject of our first results. The problem played a pivotal role in the development of the toolkit of parameterized algorithms for graph separation problems and stood as a famous open problem for years, until being independently resolved by two teams of researchers [3, 26]. The problem is not only difficult to solve exactly, but also to approximate: Chawla et al. [7] proved that, assuming Khot's [21] *Unique Games Conjecture* (UGC), it is NP-hard to approximate the edge-deletion version of the problem within any constant factor. A simple transformation shows that the same holds for the vertex-deletion problem.

Our first result (Theorem 4.2) is a polynomial-time algorithm for 3-Essential detection for Vertex Multicut, which is obtained by analyzing the integrality gap of a restricted type of linear program associated with the problem. Using known results, this preprocessing algorithm translates directly into search-space reduction for the current-best FPT algorithms for solving Vertex Multicut. This results in an algorithm (Corollary 4.3) that computes an optimal vertex multicut in an n-vertex graph in time $2^{\mathcal{O}(\ell^3)} \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of vertices in an optimal solution that are *not* 3-essential.

Our approach for essential detection also applies for the variation of Vertex Multicut on *directed* graphs. Since the directed setting is more difficult to deal with, vertices have to be slightly more essential to be able to detect them, resulting in a polynomial-time algorithm for 5-Essential detection for Directed Vertex Multicut (Theorem 4.5). This detection algorithm does not directly translate into running-time guarantees for FPT algorithms, though, as Directed Vertex Multicut is W[1]-hard parameterized by the size of the solution [29]. (When the solution is forbidden from deleting terminals, the directed problem is already $W[1]$-hard with four terminal pairs, although the case of three terminal pairs is FPT [17].)

Our second positive result concerns the Cograph (Vertex) Deletion problem. Given an undirected graph G, it asks to find a minimum-size vertex set S such that $G-S$ is a cograph, i.e., the graph $G - S$ does not contain the 4-vertex path P_4 as an induced subgraph. The problem is motivated by the fact that efficient algorithms for solving optimization problems on cographs can often be extended to work on graphs which are *close* to being cographs, as long as a deletion set is known [6, §6]. The decision version of Cograph Deletion is NP-complete due to the generic results of Lewis and Yannakakis [24]. Parameterized by the size k of the desired solution, Cograph Deletion is fixed-parameter tractable via the method of bounded-depth search trees [5]: branching on vertices of a P_4 results in a running time of $4^k \cdot n^{\mathcal{O}(1)}$. Nastos and Gao [27] proposed a refined branching strategy by exploiting the structure of *P_4-sparse graphs*, improving the running time to $3.115^k \cdot n^{\mathcal{O}(1)}$, following earlier improvements via the interpretation of Cograph Deletion as a 4-Hitting Set problem [13, 16, 28]. The latter viewpoint also gives a simple polynomial-time 4-approximation. Whether a $(4 - \varepsilon)$-approximation can be computed in polynomial time is unknown; Drescher poses this [11, §8 Question 5] as an open problem for *vertex-weighted* graphs.

Our second result (Lemma 4.6) is a polynomial-time algorithm for 3.5-Essential detection for Cograph Deletion. It directly translates into an FPT algorithm (Corollary 4.8) that, given a graph G, outputs a minimum set S for which $G - S$ is a cograph in time $3.115^\ell \cdot n^{\mathcal{O}(1)}$; here ℓ is the number of vertices in an optimal solution that are *not* 3.5-essential. Similarly as for Vertex Multicut, our detection algorithm arises from a new bound of 2.5 on the integrality gap of a restricted version of a natural linear-programming relaxation associated to the deletion problem.

The fact that our algorithm detects 3.5-essential vertices is noteworthy. It is known [4, §8] that for any $c \geq 1$, an algorithm for c-Essential detection for Π follows from an algorithm that computes a factor-c approximation for the problem of finding a minimum-size solution avoiding a given vertex v. In this setting, a 4-approximation algorithm for Cograph Deletion easily follows since the problem is a special case of d-Hitting Set. We consider it interesting that we can obtain a detection algorithm whose detection constant $c = 3.5$ is strictly better than the best-known approximation ratio 4 for the problem.

Since our positive results all arise from bounding the integrality gap of certain restricted LP-formulations, we also study the integrality gap of a standard Cograph Deletion LP and prove it to be 4 (Theorem 4.9) using the probabilistic method. This provides a sharp contrast to the gap of 2.5 in our restricted setting.

Our results: Lower bounds. Our second set of results concerns lower bounds, showing that for certain combinations of Π and c there are no efficient algorithms for c-Essential detection for Π under common complexity-theoretic hypotheses. In their work, Bumpus et al. [4] identified several problems Π such as Perfect Deletion for which the detection problem is intractable for *all* choices of c. Their proofs are based on the hardness of FPT-approximation for Dominating Set [30]. The setting for our lower bounds is different. We analyze problems for which the detection task is polynomial-time solvable for *some* essentiality threshold c, and investigate whether polynomial-time algorithms can exist for a smaller threshold $c' < c$.

Our most prominent lower bound concerns the Directed Feedback Vertex Set problem (DFVS), which has attracted a lot of attention from the parameterized complexity community [8, 25]. It asks for a minimum vertex set S of a *directed* graph G for which $G - S$ is acyclic. Svensson proved that under the UGC [32], the problem is NP-hard to approximate to within any constant factor. Nevertheless, a polynomial-time algorithm for 2-Essential detection for DFVS was given by Bumpus et al. [4, Lemma 3.3]. We prove (Theorem 5.2) that the detection threshold 2 achieved by their algorithm is likely optimal: assuming the

UGC, the detection problem for $c' = 2 - \varepsilon$ is NP-hard for any $\varepsilon \in (0, 1]$. To prove this, we show that an algorithm with $c' = (2 - \varepsilon)$ would be able to distinguish instances with small solutions from instances with large solutions, while the hardness of approximation result cited above [32] show this task to be NP-hard under the UGC.

Apart from Directed Feedback Vertex Set, we provide two further lower bounds. For the Vertex Cover (VC) problem, an algorithm to detect 2-essential vertices is known [4]. Assuming the UGC, we prove (Theorem 5.6) that $(1.5 - \varepsilon)$-Essential detection for VC is NP-hard for all $\varepsilon \in (0, 0.5]$. A simple transformation then shows $(1.5 - \varepsilon)$-Detection for Vertex Multicut is also NP-hard under the UGC. These bounds leave a gap with respect to the thresholds of the current-best detection algorithms (2 and 3, respectively). We leave it to future work to close the gap.

Organization. The remainder of the paper is organized as follows. In Section 2 we give preliminaries on graphs and linear programming. Section 3 introduces our formalization for hitting set problems on graphs and provides the connection between integrality gaps and detection algorithms. Section 4 contains our positive results, followed by the negative results in Section 5. We conclude with some open problems in Section 6. Due to space limitations, the proofs of statements marked (★) are deferred to the full version of this paper [20].

2 Preliminaries

We consider finite simple graphs, some of which are directed. Directed graphs or objects defined on directed graphs will always be explicitly indicated as such. We use standard notation for graphs and parameterized algorithms. We re-iterate the most relevant terminology and notation, but anything not defined here may be found in the textbook by Cygan et al. [9] or in the previous work on essential vertices [4].

Graph notation. We let P_ℓ denote the path graph on ℓ vertices. The weight of a path in a vertex-weighted graph is the sum of the weights of the vertices on that path, including the endpoints. Given two disjoint vertex sets S_1 and S_2 in a *(directed)* graph G, we call a third vertex set $X \subseteq V(G)$ a *(directed)* (S_1, S_2)-separator in G if it intersects every *(directed)* (S_1, S_2)-path in G. Note that X may intersect S_1 and S_2. If S_1 or S_2 is a singleton set, we may write the single element of the set instead to obtain a (v, S_2)-separator for example. Menger's theorem relates the maximum number of pairwise vertex-disjoint paths between two (sets of) vertices to the minimum size of a separator between those two (sets of) vertices. We consider the following formulation of the theorem:

▶ **Theorem 2.1** ([31, Corollary 9.1a]). *Let G be a directed graph and let $s, t \in V(G)$ be non-adjacent. Then the maximum number of internally vertex-disjoint directed (s, t)-paths is equal to the minimum size of a directed (s, t)-separator that does not include s or t.*

A *fractional (directed)* (S_1, S_2)-separator is a weight function that assigns every vertex in a graph a non-negative weight such that every *(directed)* (S_1, S_2)-path has a weight of at least 1. The total weight of a fractional *(directed)* separator is the sum of all vertex weights.

Linear programming notation. We employ well-known concepts from linear programming and refer to a textbook for additional background [31]. A solution to a linear program (LP) where all variables are assigned an integral value is called an *integral* solution. As we only consider LPs with a one-to-one correspondence between its variables and the vertices in a graph, integral solutions admit an alternative interpretation as vertex sets: the set of vertices whose corresponding variables are assigned a positive value. We use the interpretations of integral solutions as variable assignments or vertex sets interchangeably. We say that a minimization LP has an integrality gap of at most c for some $c \in \mathbb{R}$ if the cost of an optimal integral solution is at most c times the cost of an optimal fractional solution.

3 Essential vertices for Vertex Hitting Set problems

Our positive contributions all build upon the same result from Bumpus et al. [4, Theorem 4.1], which relates integrality gaps of certain LPs to the existence of c-Essential detection algorithms. A slightly generalized formulation of this can be found below as Theorem 3.1. First, we introduce the required background and notation.

The result indicates a strategy towards constructing a polynomial-time algorithm for c-Essential detection for Π for a vertex selection problem Π, by considering a specific special variant of that problem, that we refer to as its *v-Avoiding* variant. It is defined almost identically to the original problem Π, but the input additionally contains a distinguished vertex $v \in V(G)$ which is explicitly forbidden to be part of a solution.

The original theorem from Bumpus et al. [4] is specifically targeted at $\mathcal{C}$-Deletion problems for *hereditary* graph classes $\mathcal{C}$. A graph class $\mathcal{C}$ is said to be hereditary when it exhibits the property that all induced subgraphs of a graph in $\mathcal{C}$ are again in $\mathcal{C}$. The corresponding $\mathcal{C}$-Deletion problem is that of finding a minimum size vertex set whose removal turns the input graph into one contained in $\mathcal{C}$. We remark however that the theorem holds for a broader collection of problems, namely those that can be described as *Vertex Hitting Set problems*. To define which problems qualify as a Vertex Hitting Set problem, we first recall the definition of the well-known optimization problem Hitting Set, on which our definition of Vertex Hitting Set problems is based.

Hitting Set	
Input:	A universe U and a collection $\mathcal{S} \subseteq 2^U$ of subsets of U.
Feasible solution:	A set $X \subseteq U$ such that $X \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.
Objective:	Find a feasible solution of minimum size.

We define *Vertex Hitting Set problems* as vertex selection problems that can be described as a special case of Hitting Set where the universe U is the vertex set of the input graph and the collection $\mathcal{S}$ is encoded implicitly by the graph.

This definition in particular contains all $\mathcal{C}$-Deletion problems for hereditary graph classes $\mathcal{C}$. This is because every hereditary graph class can be characterized by a (possibly infinite) set of forbidden induced subgraphs. A graph G is in $\mathcal{C}$ if and only if none of its induced subgraphs are isomorphic to a forbidden induced subgraph. Therefore, a $\mathcal{C}$-Deletion instance G is equivalent to the Hitting Set instance $(V(G), \mathcal{S})$, with $\mathcal{S}$ being the collection of all the vertex subsets that induce a forbidden subgraph in G.

Now, as mentioned, the v-Avoiding variants of vertex selection problems are of particular interest. A useful consequence of considering Vertex Hitting Set problems as special cases of Hitting Set, is that this yields a well-defined canonical LP formulation for such problems that can easily be modified to describe their v-Avoiding variant. This LP formulation is based on the following standard LP for a Hitting Set instance $(U, \mathcal{S})$, which uses variables x_u for every $u \in U$:

$$\begin{aligned} &\text{minimize} && \sum_{u \in U} x_u && \\ &\text{subject to:} && \sum_{u \in S} x_u \geq 1 && \text{for every } S \in \mathcal{S} \\ & && 0 \leq x_u \leq 1 && \text{for every } u \in U \end{aligned}$$

To describe the v-Avoiding variant of a Vertex Hitting Set problem, this LP can simply be modified by adding the constraint $x_v = 0$. For a given Vertex Hitting Set problem Π, a graph G and a vertex $v \in V(G)$, we denote the resulting LP as $\mathrm{LP}_\Pi(G, v)$.

Although the original theorem from Bumpus et al. [4] makes a statement about $\mathcal{C}$-deletion problems only, it is not too hard to see that this statement also holds for any other Vertex Hitting Set problem. We therefore present this result as the following slight generalization.

▶ **Theorem 3.1.** *Let Π be a Vertex Hitting Set problem and let $c \in \mathbb{R}_{\geq 1}$. Then there exists a polynomial-time algorithm for $(c+1)$-Essential detection for Π if the following two conditions are met:*

1. *For all G and $v \in V(G)$, there is a polynomial-time separation oracle for $\mathrm{LP}_\Pi(G, v)$.*
2. *For all G and $v \in V(G)$ for which $\{v\}$ solves Π on G, the integrality gap of $\mathrm{LP}_\Pi(G, v)$ is at most c.*

This statement admits a proof that is almost identical to the proof by Bumpus et al. [4, Theorem 4.1]. At any point in that proof where the assumption is used that Π is a $\mathcal{C}$-Deletion problem for some hereditary $\mathcal{C}$, this assumption may be replaced by the property that any superset of a solution to Π is also a solution. This property is satisfied for every Vertex Hitting Set problem. Otherwise, no changes to the proof are required. We therefore refer the reader to this prior work for the details of the proof.

Many known results about the approximation of Hitting Set or about the integrality gap of Hitting Set LPs consider the restriction to d-Hitting Set. This is the problem obtained by requiring every $S \in \mathcal{S}$ in the input to be of size at most d for some positive integer d. Both upper bounds and lower bounds are known for the integrality gaps of the standard LP describing d-Hitting Set instances. The standard LP is the linear program given above for the general Hitting Set problem.

It is well-known that this LP has an integrality gap of at most d and that there exist instances for which this bound is tight. This result is for example mentioned as an exercise in a book on approximation algorithms [33, Exercise 15.3], framed from the equivalent perspective of the Set Cover problem.

4 Positive results

This section contains our positive results for essential vertex detection. For three different problems Π and corresponding values of c, we provide polynomial-time algorithms for c-Essential detection for Π. The first two of these, being strongly related, are presented in Section 4.1. There, we provide c-Essential detection algorithms for Vertex Multicut and Directed Vertex Multicut with $c = 3$ and $c = 5$ respectively. Afterward, we provide a 3.5-Essential detection algorithm for the Cograph Deletion problem in Section 4.2.

4.1 Vertex Multicut

Our first two positive results concern the well-studied Vertex Multicut problem and its directed counterpart Directed Vertex Multicut. These are optimization problems defined as follows.

> *(Directed)* Vertex Multicut
>
> **Input:** A *(directed)* graph G and a set of *(ordered)* vertex pairs $\mathcal{T} = \{(s_1, t_1), \ldots, (s_r, t_r)\}$ called the terminal pairs.
>
> **Task:** Find a minimum size vertex set $S \subseteq V(G)$ such that there is no $(s_i, t_i) \in \mathcal{T}$ for which $G - S$ contains a *(directed)* (s_i, t_i)-path.

We start by observing that both problems are Vertex Hitting Set problems: if we let $\mathcal{P}_{\mathcal{T}}(G)$ be the collection of vertex subsets that form a *(directed)* (s_i, t_i)-path in G, then the *(Directed)* Vertex Multicut instance $(G, \mathcal{T})$ is equivalent to the Hitting Set instance $(V(G), \mathcal{P}_{\mathcal{T}}(G))$. This interpretation of the problems as special cases of Hitting Set is also captured by the standard LP formulations of the problems, on which the v-Avoiding LP below is based:

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V(G)} x_u \\ \text{subject to:} \quad & \sum_{u \in V(P)} x_u \geq 1 \quad \text{for every } \textit{(directed)} \text{ path } P \text{ from some } s_i \text{ to } t_i \\ & x_v = 0 \\ & 0 \leq x_u \leq 1 \quad \text{for } u \in V(G) \end{aligned}$$

The set of constraints in this LP formulation not only depends on the structure of the input graph G, but also on the set $\mathcal{T}$ of terminal pairs. Hence, we denote the LP above as $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ for undirected G or as $\mathrm{LP}_{\mathrm{DVM}}(G, \mathcal{T}, v)$ for directed G. The standard LP formulations of Vertex Multicut and Directed Vertex Multicut are obtained by simply removing the constraint $x_v = 0$.

The undirected case. We start with the undirected version of the problem and show in Lemma 4.1 that $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ has an integrality gap of at most 2 for all Vertex Multicut instances $(G, \mathcal{T})$ where $v \in V(G)$ is such that $\{v\}$ is a solution. This bound yields a polynomial-time algorithm for 3-Essential detection for Vertex Multicut as presented in Theorem 4.2.

▶ **Lemma 4.1.** *Let $(G, \mathcal{T})$ be a Vertex Multicut instance with some $v \in V(G)$ such that $\{v\}$ is a solution for this instance. Then $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ has an integrality gap of at most 2.*

Proof. Let $\mathbf{x} = (x_u)_{u \in V(G)}$ be an optimal solution to $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ and let $z = \sum_{u \in V(G)} x_u$ be its value. If we interpret the values of x_u, as given by $\mathbf{x}$, as vertex weights, then by definition of the LP, all (s_i, t_i)-paths have weight at least 1 for all $\{s_i, t_i\} \in \mathcal{T}$. Moreover, all such paths must pass through v because $\{v\}$ is a solution, so we know for every $\{s_i, t_i\} \in \mathcal{T}$ that all (s_i, v)-paths or all (t_i, v)-paths (or both) have weight at least $\frac{1}{2}$.

We proceed by stating a reformulation of this property. Let $D \subseteq V(G)$ be the set of all vertices u such that every (u, v)-path has weight at least $\frac{1}{2}$. Then, the above property can also be described as follows: for every $\{s_i, t_i\} \in \mathcal{T}$, at least one of s_i and t_i is in D.

Using this alternate formulation, it follows that every (v, D)-separator X is also a valid solution to the given Vertex Multicut instance. To see this, consider an arbitrary (s_i, t_i)-path P for some arbitrary $\{s_i, t_i\} \in \mathcal{T}$. Since $\{v\}$ is a solution, P intersects v. If $s_i \in D$, then the fact that X is a (v, D)-separator implies that X intersects the subpath of P between v and s_i. The same holds for t_i. Since at least one of s_i and t_i is in D, it follows that X must intersect P. Because P was an arbitrary (s_i, t_i)-path for an arbitrary terminal pair $\{s_i, t_i\}$, X hits all such paths and therefore it is a vertex multicut.

Now to prove that $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ has an integrality gap of at most 2, it suffices to show that there exists a (v, D)-separator $X \subseteq V(G)$ of size at most $2z$ that does not contain v. To see that this is indeed the case, we start by constructing a fractional (v, D)-separator $f \colon V(G) \to \mathbb{R}$ of weight at most $2z$ and with $f(v) = 0$. We obtain f by simply doubling the values given by $\mathbf{x}$, i.e.: $f(u) := 2x_u$ for all $u \in V(G)$. This step is inspired by a proof from Golovin, Nagarajan, and Singh that shows an upper bound on the integrality gap of a Multicut variant in trees [15].

We observe that indeed $f(v) = 2 \cdot x_v = 0$, since $\mathbf{x}$ is a solution to $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$, which requires that $x_v = 0$. Furthermore, D was constructed such that all paths from v to a vertex in D have a weight of at least $\frac{1}{2}$ under the vertex weights as given by $\mathbf{x}$. Hence, under the doubled weights of f, all such paths have a weight of at least 1, witnessing that f is in fact a fractional (v, D)-separator.

The final step of the proof is now to show that the existence of this *fractional* (v, D)-separator of weight $2z$ implies the existence of an *integral* (v, D)-separator of size at most $2z$ that does not contain v. To do so, we use Menger's theorem on the auxiliary directed graph G' obtained from G by turning all undirected edges into bidirected edges, while adding a sink node t with incoming edges from all vertices in D.

Consider a maximum collection $\mathcal{P}$ of internally vertex-disjoint directed (v, t)-paths in G'. Let $X \subseteq V(G') \setminus \{v, t\}$ be a directed (v, t)-separator in G' of size $|\mathcal{P}|$, whose existence is guaranteed by Theorem 2.1. The construction of G' ensures that X is a (v, D)-separator in G that does not contain v, and therefore corresponds to an integral solution to $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$. To bound the integrality gap by 2, it therefore suffices to prove that $|\mathcal{P}| = |X| \leq 2z$.

For each (v, t)-path $P \in \mathcal{P}$ in G', the prefix obtained by omitting its endpoint t yields a (v, D)-path in G. Since f is a fractional (v, D)-separator, it must assign every such prefix of $P \in \mathcal{P}$ a weight of at least 1. Because $f(v) = 0$ and because the paths in $\mathcal{P}$ are internally vertex-disjoint, we find that the total weight of f must be at least $|\mathcal{P}| = |X|$. Since the weight of f is at most $2z$, we find that $|\mathcal{P}| = |X| \leq 2z$. This concludes the proof. ◀

We can even construct Vertex Multicut instances $(G, \mathcal{T})$ that are solved by some $\{v\} \subseteq V(G)$ for which the integrality gap of $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ is arbitrarily close to 2, showing that the bound in Lemma 4.1 is tight. To construct such an instance, let G be a (large) star graph, let $v \in V(G)$ be its center and let $\mathcal{T} = \binom{V(G) \setminus \{v\}}{2}$. Clearly, $\{v\}$ is a solution to the Vertex Multicut instance $(G, \mathcal{T})$.

To determine the integrality gap of $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$, we first note that any solution to the Vertex Multicut instance that avoids v must, at least, include all but one of the leaves from G. Any such set is indeed a solution, which shows that the smallest integral solution to $\mathrm{LP}_{\mathrm{VM}}(G, \mathcal{T}, v)$ has value $|V(G)| - 2$. A smaller fractional solution to the program may be obtained by assigning every leaf of G a value of $\frac{1}{2}$, which would yield a solution with a total value of $\frac{1}{2} \cdot (|V(G)| - 1)$. Observe that such a construction of G, $\mathcal{T}$, and v can be used to get an LP with an integrality gap arbitrarily close to 2 by having the star graph G be arbitrarily large.

Regardless of the bound on the integrality gap being tight, Lemma 4.1 and Theorem 3.1 combine to prove the following result.

▶ **Theorem 4.2.** (★) *There exists a polynomial-time algorithm for* 3-*Essential detection for Vertex Multicut.*

The algorithm to detect 3-essential vertices leads in a black-box fashion to a search-space reduction guarantee for the current-best algorithm for solving Vertex Multicut due to Marx and Razgon [26]. This follows from a result of Bumpus et al. [4, Theorem 5.1] (cf. [20, Theorem A.1]). While they originally stated their connection between essential detection and search-space reduction for $\mathcal{C}$-Deletion problems, it is easy to see that the same proof applies for any Vertex Hitting Set problem: the only property of $\mathcal{C}$-Deletion that is used in their proof is that for any vertex set $X \subseteq V(G)$, a vertex set $Y \subseteq V(G-X)$ is a solution to $G-X$ if and only if $X \cup Y$ is a solution to G; this property holds for any Vertex Hitting Set problem.

▶ **Corollary 4.3.** *There is an algorithm that, given a Vertex Multicut instance* $(G, \mathcal{T})$ *on* n *vertices, outputs an optimal solution in time* $2^{\mathcal{O}(\ell^3)} \cdot n^{\mathcal{O}(1)}$, *where* ℓ *is the number of vertices in an optimal solution that are* not *3-essential.*

The directed case. Keeping in mind the techniques used to prove Lemma 4.1, we proceed to the next problem: Directed Vertex Multicut. By similar arguments, we find the v-Avoiding LP of this problem to have a bounded integrality gap as well. However, adaptations to these arguments are required to take the directions of edges into consideration, yielding a higher bound on the integrality gap of the directed version of the problem.

▶ **Lemma 4.4.** (★) *Let* $(G, \mathcal{T})$ *be a Directed Vertex Multicut instance with some* $v \in V(G)$ *such that* $\{v\}$ *is a solution for it. Then* $\mathrm{LP}_{\mathrm{DVM}}(G, \mathcal{T}, v)$ *has an integrality gap of at most* 4.

Similar to the undirected setting, this upper bound on the integrality gap leads to the following algorithmic result when combined with Theorem 3.1.

▶ **Theorem 4.5.** *There exists a polynomial-time algorithm for* 5-*Essential detection for Directed Vertex Multicut.*

This statement admits a proof that is almost identical to the proof of Theorem 4.2, since the shortest-path algorithm that provides the separation oracle of the Vertex Multicut LP can also take directed graphs as input.

4.2 Cograph Deletion

Our next positive result concerns the Cograph Deletion problem. As this is a specific case of $\mathcal{C}$-Deletion, this is more in line with the original research direction for c-Essential detection introduced by Bumpus et al. [4], where a framework was built around $\mathcal{C}$-Deletion problems. The Cograph Deletion problem is defined as follows.

Cograph Deletion

Input: An undirected graph G.

Task: Find a minimum size set $S \subseteq V(G)$ such that $G - S$ is a cograph (i.e.: $G - S$ does not contain a path on 4 vertices as an induced subgraph).

We start by observing that the Cograph Deletion problem is a Vertex Hitting Set problem: if we let $\mathcal{P}_4(G)$ be the collection of vertex subsets that induce a P_4 in G, then the Cograph Deletion instance G is equivalent to the Hitting Set instance $(V(G), \mathcal{P}_4(G))$. Again, motivated by Theorem 3.1, we study the v-Avoiding LP for this problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V(G)} x_u & \\ \text{subject to:} \quad & \sum_{u \in V(H)} x_u \geq 1 & \text{for every induced subgraph } H \text{ of } G \text{ isomorphic to } P_4 \\ & x_v = 0 & \\ & 0 \leq x_u \leq 1 & \text{for } u \in V(G) \end{aligned}$$

For a given graph G and vertex $v \in V(G)$, we denote the LP above as $\mathrm{LP}_{\mathrm{CD}}(G, v)$. Whenever v is such that $G - v$ is a cograph, the resulting LP admits a simple upper bound on the integrality gap. This bound is derived from the observation that the v-Avoiding Cograph Deletion problem is a special case of 3-Hitting Set: the vertex sets to be hit in the problem are the triplets of vertices that, together with v, induce a P_4 in G. As the natural LP describing 3-Hitting Set has an integrality gap of at most 3, it follows that the natural LP formulation of v-Avoiding Cograph Deletion, to which the above LP is equivalent, also has an integrality gap of at most 3.

This section is dedicated to proving a stronger result than this trivial bound. We prove that, whenever v is such that $G - v$ is a cograph, $\mathrm{LP}_{\mathrm{CD}}(G, v)$ has an integrality gap of at most 2.5. To prove this, we use a method inspired by iterative rounding [19], where an approximate integral solution can be obtained by solving the LP, picking all vertices that receive a large enough value, updating the LP to no longer contain these vertices and repeating these steps until a solution is found.

For our purposes, we consider values of at least 0.4 to be “large enough”. However, we will see that an extension to the original method is required since $\mathrm{LP}_{\mathrm{CD}}(G, v)$ is not guaranteed to always have an optimal solution that assigns at least one vertex a value of ≥ 0.4. This issue is reflected in the inductive proof below by having the step case split into two subcases. The first of these deals with the standard iterative rounding setup, while the second subcase deals with the possibility of an optimal solution not assigning any vertex a large value.

▶ **Lemma 4.6.** *Let G be a graph and let $v \in V(G)$ be such that $G - v$ is a cograph. Then $\mathrm{LP}_{\mathrm{CD}}(G, v)$ has an integrality gap of at most 2.5.*

Proof. We prove the statement by induction on the value of an optimal fractional solution to the linear program.

First, consider as base case that $\mathrm{LP}_{\mathrm{CD}}(G, v)$ has an optimal fractional solution of value 0. Then this solution is the all-zero solution. This is also an integral optimum solution to the program, so the integrality gap of the program is 1 and the claim holds.

Next, let $\mathbf{x} = (x_u)_{u \in V(G)}$ be an optimal solution to $\mathrm{LP}_{\mathrm{CD}}(G, v)$, let $z = \sum_{u \in V(G)} x_u$ be its value and let $V_{\geq 0.4} \subseteq V(G)$ be the set of vertices that are assigned a value of at least 0.4 in this solution. We distinguish two cases.

Case 1. Suppose $V_{\geq 0.4} \neq \emptyset$. Consider the pair $(G - V_{\geq 0.4}, v)$ and note that $(G - V_{\geq 0.4}) - v$, being an induced subgraph of $G - v$, is a cograph. Also note that the restriction of $\mathbf{x}$ to $G - V_{\geq 0.4}$ is a feasible solution to $\mathrm{LP}_{\mathrm{CD}}(G - V_{\geq 0.4}, v)$. This solution has a value of $z - \sum_{u \in V_{\geq 0.4}} x_u \leq z - 0.4 \cdot |V_{\geq 0.4}|$, which is strictly smaller than z by the assumption that $V_{\geq 0.4} \neq \emptyset$. If we let z_{res} be the value of an optimal solution to $\mathrm{LP}_{\mathrm{CD}}(G - V_{\geq 0.4}, v)$, then this implies that $z_{\mathrm{res}} \leq z - 0.4|V_{\geq 0.4}| < z$ as well.

Then, by the induction hypothesis, an integral solution V_{res} to $\text{LP}_{\text{CD}}(G - V_{\geq 0.4}, v)$ with $|V_{\text{res}}| \leq 2.5z_{\text{res}}$ exists. To prove that $\text{LP}_{\text{CD}}(G, v)$ has an integrality gap of at most 2.5, we proceed by showing that $V_{\text{res}} \cup V_{\geq 0.4}$ is an integral solution to $\text{LP}_{\text{CD}}(G, v)$ with value at most $2.5z$. We start by arguing that $V_{\text{res}} \cup V_{\geq 0.4}$ is a valid integral solution.

First note that neither V_{res} nor $V_{\geq 0.4}$ contains v since both $\text{LP}_{\text{CD}}(G - V_{\geq 0.4}, v)$ and $\text{LP}_{\text{CD}}(G, v)$ require $x_v = 0$. Therefore, the union of these two sets also does not contain v. Secondly, note that V_{res} (by construction of $\text{LP}_{\text{CD}}(G - V_{\geq 0.4}, v)$) contains a vertex from every induced P_4 in G that does not already have a vertex in $V_{\geq 0.4}$. As such, $V_{\text{res}} \cup V_{\geq 0.4}$ contains a vertex from every induced P_4 in G, which makes it a feasible solution to $\text{LP}_{\text{CD}}(G, v)$.

Knowing this, it remains to prove that $V_{\text{res}} \cup V_{\geq 0.4}$ has size at most $2.5z$. Recall that we derived $z_{\text{res}} \leq z - 0.4 \cdot |V_{\geq 0.4}|$. We can use this inequality to make the following derivation:

$$\begin{aligned}
|V_{\text{res}} \cup V_{\geq 0.4}| \leq |V_{\text{res}}| + |V_{\geq 0.4}| &\leq 2.5z_{\text{res}} + |V_{\geq 0.4}| && \text{by definition of } V_{\text{res}} \\
&\leq 2.5\,(z - 0.4 \cdot |V_{\geq 0.4}|) + |V_{\geq 0.4}| && \text{by the above inequality} \\
&= 2.5z - |V_{\geq 0.4}| + |V_{\geq 0.4}| \;=\; 2.5z && \text{since } 2.5 \cdot 0.4 = 1
\end{aligned}$$

Case 2. Suppose $V_{\geq 0.4} = \emptyset$. Let $V^* \subseteq V(G) \setminus \{v\}$ be the set of vertices other than v that are part of at least one induced P_4 in G. To prove that $\text{LP}_{\text{CD}}(G, v)$ has an integrality gap of at most 2.5, we show that the smaller set of $V^* \cap N_G(v)$ and $V^* \setminus N_G(v)$ is a solution to the program with size at most $2.5z$.

We start by proving that $V^* \cap N_G(v)$ and $V^* \setminus N_G(v)$ are both feasible solutions to $\text{LP}_{\text{CD}}(G, v)$. We do so using an observation about the structure of the graph P_4. Observe that this graph has the property that each vertex has at least one neighbor and at least one non-neighbor. Since v is part of every induced P_4 in G by assumption, this means that every induced P_4 in G contains both a neighbor and a non-neighbor of v.

The above observation implies that $V^* \cap N_G(v)$ and $V^* \setminus N_G(v)$ both intersect all induced subgraphs of G isomorphic to P_4. Hence, both of these sets are feasible solutions to $\text{LP}_{\text{CD}}(G, v)$. It remains to prove that the smaller of the two sets has a size of at most $2.5z$.

Since $V^* \cap N_G(v)$ and $V^* \setminus N_G(v)$ form a partition of V^* into two parts, the smaller of the two will always be of size at most $|V^*|/2$. Therefore, it suffices to show that $|V^*|/2 \leq 2.5z$. To prove this, we start by showing that the assumption that $V_{\geq 0.4} = \emptyset$ implies that $x_w \geq 0.2$ for all vertices $w \in V^*$.

We prove this property by contradiction, so suppose there is some vertex $w \in V(G) \setminus \{v\}$ that is part of an induced P_4, but which has $x_w < 0.2$. Let H be an induced subgraph of G that is isomorphic to P_4 and with $w \in V(H)$. Because $G - v$ is a cograph, v is contained in every induced P_4 and in particular $v \in V(H)$. By definition of $\text{LP}_{\text{CD}}(G, v)$, we have $x_v = 0$. By the assumption that $V_{\geq 0.4} = \emptyset$, the two vertices in $V(H) \setminus \{w, v\}$ have value at most 0.4, so $\sum_{u \in V(H)} x_u < 1$, which contradicts the validity of $\mathbf{x}$.

Knowing that $x_w \geq 0.2$ for all $w \in V^*$, it follows that $z = \sum_{u \in V(G)} x_u \geq 0.2|V^*|$. Rewriting this inequality, we obtain $|V^*|/2 \leq 2.5z$. ◀

At the moment, we are not aware of any examples of pairs (G, v) where $G - v$ is a cograph and for which $\text{LP}_{\text{CD}}(G, v)$ has an integrality gap of 2.5. Therefore, the bound above does not have to be tight and the integrality gap of such programs may even be as small as 2. However, there do exist pairs (G, v) where $G - v$ is a cograph and for which $\text{LP}_{\text{CD}}(G, v)$ has an integrality gap arbitrarily close to 2.

Such a pair (G, v) may be obtained by constructing G as the union of m disjoint edges and adding the vertex v to it which is adjacent to exactly one endpoint of each of these m edges. Then, any integral solution to $\text{LP}_{\text{CD}}(G, v)$ must include, at least, one endpoint from all but one of the original m edges. Any such set of vertices is in fact a feasible integral solution, so a smallest integral solution has size $m - 1$.

An optimal fractional solution may be obtained by assigning all m neighbors of v a value of 0.5, which yields a total value of $m/2$. Hence, the integrality gap of $\mathrm{LP}_{\mathrm{CD}}(G, v)$ is $\frac{m-1}{m/2} = 2 \cdot \frac{m-1}{m}$, which can be arbitrarily close to 2 for arbitrarily large m.

Like earlier, the upper bound on the integrality gap shown in Lemma 4.6 leads to the following algorithmic result.

▶ **Theorem 4.7.** (★) *There exists a polynomial-time algorithm for* 3.5-*Essential detection for Cograph Deletion.*

The algorithm to detect 3.5-essential vertices leads to a search-space reduction guarantee for the current-best parameterized algorithm for Cograph Deletion [27] via Theorem 5.1 by Bumpus et al. [4].

▶ **Corollary 4.8.** *There is an algorithm that, given a Cograph Deletion instance G on n vertices, outputs an optimal solution in time $3.115^{\ell} \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of vertices in an optimal solution that are* not *3.5-essential.*

In the full version of this paper [20, Section 4.2.1] we contrast the integrality gap of 2.5 for the v-avoiding version of Cograph Deletion to the standard version for the problem, for which we provide the following lower bound using the probabilistic method.

▶ **Theorem 4.9.** (★) *For all $\varepsilon > 0$, the integrality gap of the standard Cograph Deletion LP is larger than $4 - \varepsilon$.*

5 Hardness results

In this section, we show two main hardness results regarding essential detection algorithms. The first of these concerns Directed Feedback Vertex Set (DFVS). The objective in this problem is to find a smallest vertex set S in a directed input graph G such that $G - S$ is acyclic. We slightly abuse notation by using the acronym DFVS to denote both a (not necessarily optimal) solution to a given input and the name of the problem itself. Additionally, we let DFVS(G) denote the size of a smallest DFVS in G. The hardness result obtained for DFVS can be extended to Directed Vertex Multicut. The second result concerns Vertex Cover (VC) and it can be extended to other vertex hitting set problems on undirected graphs, including Vertex Multicut.

Our results are based on the hardness assumption posed by the Unique Games Conjecture (UGC) [21]. Although the conjecture has remained open since its introduction in 2002, many conditional hardness results in the area of approximation algorithms follow from it. Before stating our first new hardness result, we mention the known result it is derived from, which itself is an implication of the UGC. By the nature of the UGC, many results derived from it show the conditional hardness of distinguishing between two types of problem inputs: one with a very small solution and one with a very large solution. Indeed, we derive our hardness from one such result due to Svensson [32, Theorem 1.1] that implies the following.

▶ **Lemma 5.1.** (★) *Assuming the UGC, the following problem is NP-hard for any integer $r \geq 2$ and sufficiently small constant $\delta > 0$. Given a directed n-vertex graph G, distinguish between the following two cases:*

- $\mathrm{DFVS}(G) \leq \left(\frac{1-\delta}{r} + \delta\right) n$
- $\mathrm{DFVS}(G) \geq (1 - \delta) n$

We use this formulation to prove the following.

▶ **Theorem 5.2.** *Assuming the UGC, $(2 - \varepsilon)$-Essential detection for DFVS is NP-hard for any $\varepsilon \in (0, 1]$.*

Proof. Let $\varepsilon \in (0,1]$ be given. We can assume w.l.o.g. that $\frac{2}{\varepsilon}$ is integral. If not, we could consider some $\varepsilon' < \varepsilon$ such that $\frac{2}{\varepsilon'}$ is integer and prove hardness for $(2-\varepsilon')$-essential detection. As a $(2-\varepsilon)$-essential detection algorithm is also a valid algorithm for $(2-\varepsilon')$-essential detection, this would imply the hardness of $(2-\varepsilon)$-essential detection as well.

Now, we use Lemma 5.1 as a starting point for hardness. To do so, let G be an arbitrary directed graph on n vertices. To use Lemma 5.1, we show how to reduce G into a directed graph G', such that solving $(2-\varepsilon)$-Essential detection for DFVS on G' allows us to distinguish between $\mathrm{DFVS}(G) \leq \left(\frac{1-\delta}{r} + \delta\right) n$ and $\mathrm{DFVS}(G) \geq (1-\delta)n$ for some integer $r \geq 2$ and arbitrarily small $\delta > 0$. We assume w.l.o.g. that $n \cdot \varepsilon/2$ is integer. If not, we could consider the graph obtained by having $2/\varepsilon$ independent copies of G instead, as the minimum size of a DFVS relative to the total graph size would be the same. We proceed by explaining the reduction, after which we prove its correctness.

Our reduction starts with the directed graph G and depends on the value of ε. The full version of this paper contains a visual example [20, Figure 1]. We start the construction of G' as a copy of G. To avoid confusion between vertices in G and G', we denote the current vertex set of G' as P. Next, we expand the graph with two additional sets of vertices Q_{in} and Q_{out}. These sets each consist of $m := (1 - \frac{\varepsilon}{2})n$ vertices, which is integer by our assumptions on n and ε. We denote the vertices of Q_{in} as $q_1, \ldots, q_m$ and the vertices of Q_{out} as $q'_1, \ldots, q'_m$. We define $Q := Q_{\text{in}} \cup Q_{\text{out}}$.

We complete the construction of G' by adding more arcs to it. For every $i \in [m]$, we add the arc (q_i, q'_i). For every $p \in P$ and $q_i \in Q_{\text{in}}$, we add the arc (p, q_i). For every $p \in P$ and $q'_i \in Q_{\text{out}}$, we add the arc (q'_i, p). This completes the construction of G'. Observe that it ensures that (p, q_i, q'_i) is a directed cycle for every $p \in P$ and $i \in [m]$.

To prove the correctness of this reduction, we show that the output of an algorithm for $(2-\varepsilon)$-Essential detection for DFVS on G' can be used as subroutine to distinguish between $\mathrm{DFVS}(G) \leq \left(\frac{1-\delta}{r} + \delta\right) n$ and $\mathrm{DFVS}(G) \geq (1-\delta)n$ for some integer $r \geq 2$ and arbitrarily small $\delta > 0$. In particular, we show that this is possible for $r = \frac{4}{\varepsilon}$, which is integer by the assumption that $\frac{2}{\varepsilon}$ is integer. From now on, we fix $r = \frac{4}{\varepsilon}$ and $\delta > 0$ to be arbitrarily small so that $\delta \leq \frac{\varepsilon}{4}$ in particular.

Now, suppose that an algorithm for $(2-\varepsilon)$-Essential detection for DFVS exists and let $S \subseteq V(G')$ be its output when run on G' with k set to n. (Recall, k represents a guess for (an upper bound) of the size of an optimal solution in G'. In this setting, that would be a guess for the size of a minimum size DFVS in G'.) We show that the following two implications hold:

▷ Claim 5.3. (★) If $\mathrm{DFVS}(G) \leq \left(\frac{1-\delta}{r} + \delta\right) n$, then $|S| < n$.

▷ Claim 5.4. (★) If $\mathrm{DFVS}(G) \geq (1-\delta)n$, then $|S| = n$.

Then, simply checking the size of the output set S suffices to distinguish between $\mathrm{DFVS}(G) \leq \left(\frac{1-\delta}{r} + \delta\right) n$ and $\mathrm{DFVS}(G) \geq (1-\delta)n$. From Lemma 5.1, we know that this distinction is NP-hard to make under the UGC, meaning that $(2-\varepsilon)$-Essential detection for DFVS is also NP-hard when assuming the UGC. To prove Theorem 5.2, it therefore suffices to prove Claim 5.3 and Claim 5.4. The full proofs can be found in the full version.

Proof sketch for Claim 5.3. Let X be a smallest DFVS in G. It follows from the construction of G' that the set $X \cup Q_{\text{in}}$ is a DFVS in G'. Its size is strictly smaller than n, which follows from our choice of r and by δ being arbitrarily small. Since we invoke the algorithm for $(2-\varepsilon)$-Essential detection for DFVS with $k = n$, by Property (G1) the set S must be a subset of some smallest DFVS in G'. This implies that $|S| < n$, proving the claim. ◁

Proof sketch for Claim 5.4. Suppose that $\mathrm{DFVS}(G) \geq (1-\delta)n$. By construction of G', the set P is a DFVS in G' so that $\mathrm{DFVS}(G') \leq |P| = n$. We aim to show that P is in fact the unique smallest DFVS in G', by showing that all vertices in P are $(2-\varepsilon)$-essential in G' and therefore cannot be avoided in any $(2-\varepsilon)$-approximate solution, let alone in an optimal solution. Assuming the $(2-\varepsilon)$-essentiality of the vertices in P, it follows from Property (G2) that the set S (the output of running a $(2-\varepsilon)$-Essential detection for DFVS algorithm on G' with k set to n) must contain all of P, so $|S| \geq n$. By Property (G1), no other vertices can be in S, so $|S| = n$.

To establish the claim, it therefore suffices to prove that all vertices of P are $(2-\varepsilon)$-essential. By construction of G', each vertex $p \in P$ forms a directed cycle with each of the m pairs (q_i, q_i') in Q. Any solution avoiding p therefore contains at least m vertices from Q, but also contains at least $\mathrm{DFVS}(G) \geq (1-\delta)n$ vertices from P to hit all cycles of $G'[P] = G$. Our choice of m and δ ensure $m + (1-\delta)n \geq (2-\varepsilon)n \geq (2-\varepsilon)\mathrm{DFVS}(G')$. ◁

This concludes the proof of Theorem 5.2. ◀

The lower bound of Theorem 5.2 yields an analogous lower bound for Directed Vertex Multicut, since the set of solutions for Directed Feedback Vertex Set on a graph G equals the set of solutions to the Directed Vertex Multicut instance obtained from G by introducing a terminal pair (u_2, u_1) for every arc (u_1, u_2) of G.

▶ **Corollary 5.5.** *Assuming the UGC, $(2-\varepsilon)$-Essential detection for Directed Vertex Multicut is NP-hard for any $\varepsilon > 0$.*

By applying the proof technique above, but starting from a result about hardness of approximation for d-Hitting Set [22], we prove the following lower bound for Vertex Cover. It implies the same lower bound for Undirected Vertex Multicut.

▶ **Theorem 5.6.** (★) *Assuming the UGC, $(1.5-\varepsilon)$-Essential detection for VC is NP-hard for any $\varepsilon \in (0, 0.5]$.*

▶ **Corollary 5.7.** (★) *Assuming the UGC, $(1.5-\varepsilon)$-Essential detection for Vertex Multicut is NP-hard for any $\varepsilon \in (0, 0.5]$.*

6 Conclusion and discussion

We revisited the framework of search-space reduction via the detection of essential vertices. The improved running-time guarantees for fixed-parameter tractable algorithms that result from our detection algorithms give insight into which types of inputs of NP-hard vertex hitting set problems can be solved efficiently and optimally: not only the inputs whose total solution size is small, but also those in which all but a small number of vertices of an optimal solution are essential. Our detection algorithms arise by analyzing the integrality gap for the v-Avoiding version of the corresponding LP-relaxation, which only has to be analyzed for inputs in which $\{v\}$ is a singleton solution. Our results show that the integrality gaps in this setting are much smaller than for the standard linear program of the hitting set formulation.

For Directed Feedback Vertex Set, our lower bound shows that the polynomial-time algorithm that detects 2-essential vertices is best-possible under the UGC. For Vertex Cover and Vertex Multicut, our lower bounds do not match the existing upper bounds. It would be interesting to close these gaps.

Our positive results rely on standard linear programming formulations of the associated hitting set problems. In several scenarios, algorithms based on the standard linear program can be improved by considering stronger relaxations such as those derived from the Sherali-Adams hierarchy or Lasserre-hierachy (cf. [23]). For example, Aprile, Drescher, Fiorini, and

Huynh [2] proved that for the Cluster Vertex Deletion problem (which asks to hit all the induced P_3 subgraphs) the integrality gap of the standard LP-formulation is 3, but decreases to 2.5 using the first round of the Sherali-Adams hierarchy. Applying $(1/\varepsilon)^{\mathcal{O}(1)}$ rounds further decreases the gap to $2 + \varepsilon$. Can such hierarchies lead to better algorithms for c-Essential detection?

So far, the notion of c-essentiality has been explored for vertex hitting set problems on graphs. For other optimization problems whose solutions are subsets of objects (for example, edge subsets, or subsets of tasks in a scheduling problem) one can define c-essential objects as those contained in all c-approximate solutions. Does this notion have interesting applications for problems that are not about graphs?

References

1 Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. Technical Report 16-44, ZIB, Takustr.7, 14195 Berlin, 2016. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-60370.

2 Manuel Aprile, Matthew Drescher, Samuel Fiorini, and Tony Huynh. A tight approximation algorithm for the cluster vertex deletion problem. *Math. Program.*, 197(2):1069–1091, 2023. doi:10.1007/S10107-021-01744-W.

3 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.

4 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *Proceedings of the 30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.30.

5 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

6 Leizhen Cai. Parameterized complexity of vertex colouring. *Discret. Appl. Math.*, 127(3):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.

7 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006. doi:10.1007/S00037-006-0210-9.

8 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Rodney G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Publishing Company, Incorporated, 2012.

11 Matthew Drescher. *Two Approaches to Approximation Algorithms for Vertex Deletion Problems.* PhD thesis, Université libre de bruxelles, 2021. URL: https://knavely.github.io/knavely.gitub.io/thesis.pdf.

12 Michael R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 276–277, 2006. doi:10.1007/11847250_25.

13 Henning Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. *Algorithmica*, 57(1):97–118, 2010. doi:10.1007/S00453-008-9199-6.

14 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing.* Cambridge University Press, 2019.

15 Daniel Golovin, Viswanath Nagarajan, and Mohit Singh. Approximating the k-multicut problem. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 621–630. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109625.

16 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. doi:10.1007/S00453-004-1090-5.

17 Meike Hatzel, Lars Jaffke, Paloma T. Lima, Tomás Masarík, Marcin Pilipczuk, Roohani Sharma, and Manuel Sorge. Fixed-parameter tractability of DIRECTED MULTICUT with three terminal pairs parameterized by the size of the cutset: twin-width meets flow-augmentation. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3229–3244. SIAM, 2023. doi:10.1137/1.9781611977554.CH123.

18 Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013. doi:10.1016/J.TCS.2012.03.013.

19 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Comb.*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

20 Bart M. P. Jansen and Ruben F. A. Verhaegh. Search-space reduction via essential vertices revisited: Vertex multicut and cograph deletion, 2024. arXiv:2404.09769.

21 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, STOC 2002*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

22 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/J.JCSS.2007.06.019.

23 Monique Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003. doi:10.1287/MOOR.28.3.470.16391.

24 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

25 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.

26 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.

27 James Nastos and Yong Gao. Bounded search tree algorithms for parametrized cograph deletion: Efficient branching rules by exploiting structures of special graph classes. *Discret. Math. Algorithms Appl.*, 4(1), 2012. doi:10.1142/S1793830912500085.

28 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003. doi:10.1016/S1570-8667(03)00009-1.

29 Marcin Pilipczuk and Magnus Wahlström. Directed multicut is W[1]-hard, even for four terminal pairs. *ACM Trans. Comput. Theory*, 10(3):13:1–13:18, 2018. doi:10.1145/3201775.

30 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.

31 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

32 Ola Svensson. Hardness of vertex deletion and project scheduling. *Theory Comput.*, 9:759–781, 2013. doi:10.4086/toc.2013.v009a024.

33 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

34 Karsten Weihe. Covering trains by stations or the power of data reduction. In *Algorithms and Experiments (ALEX98)*, pages 1–8, 1998. URL: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.2173.

Edge Multiway Cut and Node Multiway Cut Are Hard for Planar Subcubic Graphs

Matthew Johnson
Durham University, UK

Barnaby Martin
Durham University, UK

Sukanya Pandey
Utrecht University, The Netherlands

Daniël Paulusma
Durham University, UK

Siani Smith
University of Bristol, UK
Heilbronn Institute for Mathematical Research, Bristol, UK

Erik Jan van Leeuwen
Utrecht University, The Netherlands

Abstract

It is known that the weighted version of Edge Multiway Cut (also known as Multiterminal Cut) is NP-complete on planar graphs of maximum degree 3. In contrast, for the unweighted version, NP-completeness is only known for planar graphs of maximum degree 11. In fact, the complexity of unweighted Edge Multiway Cut was open for graphs of maximum degree 3 for over twenty years. We prove that the unweighted version is NP-complete even for planar graphs of maximum degree 3. As weighted Edge Multiway Cut is polynomial-time solvable for graphs of maximum degree at most 2, we have now closed the complexity gap. We also prove that (unweighted) Node Multiway Cut (both with and without deletable terminals) is NP-complete for planar graphs of maximum degree 3. By combining our results with known results, we can apply two meta-classifications on graph containment from the literature. This yields full dichotomies for all three problems on $\mathcal{H}$-topological-minor-free graphs and, should $\mathcal{H}$ be finite, on $\mathcal{H}$-subgraph-free graphs as well. Previously, such dichotomies were only implied for $\mathcal{H}$-minor-free graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases multiway cut, planar subcubic graph, complexity dichotomy, graph containment

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.29

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2211.12203

Acknowledgements The authors wish to thank Jelle Oostveen for his helpful comments.

1 Introduction

In this paper we consider the unweighted edge and node versions of the classic Multiway Cut problem, which is one of the most central separation/clustering graph problems with applications in, for example, computer vision [3, 6] and multi-processor scheduling [26].

To define these problems, let $G = (V, E)$ be a graph. For a subset S of either vertices or edges of G, let $G - S$ denote the graph obtained from G after deleting all elements, either vertices (and incident edges) or edges, of S. Now, let $T \subseteq V$ be a set of specified vertices

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 29; pp. 29:1–29:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that are called the *terminals* of G. A set $S \subseteq E$ is an *edge multiway cut* for (G,T) if every connected component of $G - S$ contains at most one vertex of T. In order words, removing S pairwise disconnects the terminals of T. We define the notion of a *node multiway cut* $S \subseteq V$ in the same way, but there are two versions depending on whether or not S can contain vertices of T. This leads to the following three decision problems, where the second one is also known as Unrestricted Node Multiway Cut and the third one as Restricted Node Multiway Cut or Node Multiway Cut with Undeletable Terminals.

Edge Multiway Cut
Input: A graph G, a set of terminals $T \subseteq V$ and an integer k.
Question: Does (G,T) have an edge multiway cut $S \subseteq E$ of size at most k?

Node Multiway Cut with Deletable Terminals
Input: A graph G, a set of terminals $T \subseteq V$ and an integer k.
Question: Does (G,T) have a node multiway cut $S \subseteq V$ of size at most k?

Node Multiway Cut
Input: A graph G, a set of terminals $T \subseteq V$ and an integer k.
Question: Does (G,T) have a node multiway cut $S \subseteq V \setminus T$ of size at most k?

In Weighted Edge Multiway Cut, we are given a function $\omega : E(G) \to \mathbb{Q}^+$. The goal is to decide if (G,T) admits an edge multiway cut of total weight at most k. If $\omega \equiv 1$, then we obtain Edge Multiway Cut. Similarly, we can define weighted variants of both versions of Node Multiway Cut with respect to a node weight function $\omega : V(G) \to \mathbb{Q}^+$.

The above problems have been studied extensively; see, for example, [2, 8, 9, 10, 11, 12, 15, 16, 17, 19, 20, 22, 23]. The problems can be thought of as the natural dual problems of the Steiner Tree problem. In their famous study of Edge Multiway Cut, Dahlhaus et al. [13] showed that it is NP-complete even if the set of terminals has size $|T| = 3$. Garg et al. [16] showed the same for Node Multiway Cut. We note that this is a tight result: if $|T| = 2$, then both problems reduce to the Minimum Cut problem. The latter problem can be modelled as a maximum flow problem, and hence is well known to be solvable in polynomial time [14]. Note that Node Multiway Cut with Deletable Terminals is trivially polynomial-time solvable for any fixed $|T|$.

Our Focus. A graph is *subcubic* if it has maximum degree at most 3. Our goal in this paper is to answer the following question:

What is the computational complexity of Edge Multiway Cut *and both versions of* Node Multiway Cut *for planar subcubic graphs?*

Motivation. Our first reason is due to a complexity gap that was left open in the literature for over twenty years. That is, in addition to their NP-completeness result for $|T| = 3$, Dahlhaus et al. [13] also proved that Weighted Edge Multiway Cut is NP-complete on planar subcubic graphs using integral edge weights. Any edge of integer weight j can be replaced by j parallel edges (and vice versa) without changing the problem. Hence, their reduction implies that Edge Multiway Cut is NP-complete on planar graphs of maximum degree at most 11 [13, Theorem 2b].

Dahlhaus et al. [13] write that "*The degree bound of* 11 *is not the best possible. Using a slight variant on the construction and considerably more complicated arguments, we believe it can be reduced at least to* 6", but no further arguments were given. Even without the planarity condition and only focussing on the maximum degree bound, the hardness result of Dahlhaus et al. [13] is still best known. Given that the problem is polynomial-time solvable if the maximum degree is 2, this means that there is a significant complexity gap that has yet to be addressed.

To the best of our knowledge, there is no explicit hardness result in the literature that proves NP-completeness of either version of Node Multiway Cut on graphs of any fixed degree or on planar graphs. However, known and straightforward reductions (see e.g. [16, 23]) immediately yield NP-hardness on planar subcubic graphs for Node Multiway Cut with Deletable Terminals (see Theorem 1.2), but only on planar graphs of maximum degree 4 for Node Multiway Cut (see Proposition 3.1).

Our second reason is the central role planar subcubic graphs play in complexity dichotomies of graph problems restricted to graphs that do not contain any graph from a set $\mathcal{H}$ as a topological minor[1] or subgraph; such graphs are said to be *$\mathcal{H}$-topological-minor-free* or *$\mathcal{H}$-subgraph-free*, respectively. For both the topological minor containment relation [24] and the subgraph relation (see [18]) meta-classifications exist. To apply these meta-classifications, a problem must satisfy certain conditions, in particular being NP-complete for subcubic planar graphs for the topological minor relation, and being NP-complete for subcubic graphs for the subgraph relation. These two conditions are *exactly what is left to prove* for Edge Multiway Cut and both versions of Node Multiway Cut. In contrast, the results of [1, 13, 24] and the aforementioned reductions from [16, 23] imply that all three problems are fully classified for $\mathcal{H}$-minor-free graphs: the problems are polynomial-time solvable if $\mathcal{H}$ contains a planar graph and NP-complete otherwise (see also [18]). Hence, determining the complexity status of our three problems for planar subcubic graphs is a pressing issue for obtaining new complexity classifications for $\mathcal{H}$-topological-minor-free graphs and $\mathcal{H}$-subgraph-free-graphs.

Our third reason is the rich tradition to investigate the NP-completeness of problems on subcubic graphs and planar subcubic graphs (see e.g. the list given by Johnson et al. [18]) which continues till this day, as evidenced by recent NP-completeness results for subcubic graphs (e.g. [4, 27]) and planar subcubic graphs (e.g. [5, 28]). We also note that Edge Multicut, the standard generalization of Edge Multiway Cut where given pairs of terminals must be disconnected by an edge cut', is NP-complete even on subcubic trees [7].

For the above reasons, the fact that the complexity status of our three problems restricted to (planar) subcubic graphs has remained open this long is unexpected.

1.1 Our Results

The following three results fully answer our research question.

▶ **Theorem 1.1.** Edge Multiway Cut *is* NP-*complete for planar subcubic graphs.*

▶ **Theorem 1.2.** Node Multiway Cut with Deletable Terminals *is* NP-*complete for planar subcubic graphs.*

▶ **Theorem 1.3.** Node Multiway Cut *is* NP-*complete for planar subcubic graphs.*

We prove Theorem 1.1 in Section 2 and Theorems 1.2 and 1.3 in the Section 3.

[1] A graph G contains a graph H as a *topological minor* if G can be modified into H by a sequence of edge deletions, vertex deletions and vertex dissolutions, where a vertex dissolution is the contraction of an edge incident to a vertex of degree 2 whose (two) neighbours are non-adjacent.

In spirit, our construction for Edge Multiway Cut is similar to the one by Dahlhaus et al. [13] for graphs of maximum degree 11. For non-terminal vertices of high degree, a local replacement by a (sub)cubic graph is relatively easy. However, for terminal vertices of high degree, a local replacement strategy seems impossible. Hence, the fact that terminals in the construction of Dahlhaus et al. [13] can have degree up to 6 becomes a crucial bottleneck. To ensure that our constructed graph has maximum degree 3, we therefore need to build different gadgets. We then leverage several deep structural properties of the edge multiway cut in the resulting instance, making for a significantly more involved and technical correctness proof. Crucially, we first prove NP-completeness for a weighted version of the problem on graphs of maximum degree 5, in which each terminal is incident with exactly one edge of weight 3. Then we replace weighted edges and high-degree vertices with appropriate gadgets.

The NP-completeness for Node Multiway Cut for planar subcubic graphs follows from the NP-hardness of Edge Multiway Cut by constructing the line graph of input graph. The hardness for Node Multiway Cut with Deletable Terminals on planar subcubic graphs follows from a straightforward reduction from Vertex Cover.

1.2 Consequences

As discussed above, we immediately have the following dichotomy.

▶ **Corollary 1.4.** *For every $\Delta \geq 1$,* Edge Multiway Cut *and both versions of* Node Multiway Cut *on graphs of maximum degree Δ are polynomial-time solvable if $\Delta \leq 2$, and* NP*-complete if $\Delta \geq 3$.*

From a result of Robertson and Seymour [24], it follows that any problem Π that is NP-hard on subcubic planar graphs but polynomial-time solvable for graphs of bounded treewidth can be fully classified on $\mathcal{H}$-topological minor-free graphs. Namely, Π is polynomial-time solvable if $\mathcal{H}$ contains a subcubic planar graph and NP-hard otherwise. It is known that Edge Multiway Cut and both versions of Node Multiway Cut satisfy the second property [1]. As Theorems 1.1–1.3 show the first property, we obtain the following dichotomy.

▶ **Corollary 1.5.** *For every set of graphs $\mathcal{H}$,* Edge Multiway Cut *and both versions of* Node Multiway Cut *on $\mathcal{H}$-topological-minor-free graphs are polynomial-time solvable if $\mathcal{H}$ contains a planar subcubic graph, and* NP*-complete otherwise.*

Let the *ℓ-subdivision* of a graph G be the graph obtained from G after replacing each edge uv by a path of $\ell + 1$ edges with end-vertices u and v. A problem Π is NP-hard *under edge subdivision of subcubic graphs* if for every integer $j \geq 1$ there is an $\ell \geq j$ such that: if Π is NP-hard for the class $\mathcal{G}$ of subcubic graphs, then Π is NP-hard for the class G^{ℓ} consisting of the ℓ-subdivisions of the graphs in $\mathcal{G}$. Now say that Π is polynomial-time solvable on graphs of bounded treewidth and NP-hard for subcubic graphs and under edge subdivision of subcubic graphs. The meta-classification from Johnson et al. [18] states that for every *finite* set $\mathcal{H}$, Π on $\mathcal{H}$-subgraph-free graphs is polynomial-time solvable if $\mathcal{H}$ contains a graph from $\mathcal{S}$, and NP-hard otherwise. Here, $\mathcal{S}$ is the set consisting of all disjoint unions of zero or more paths and subdivided claws (4-vertex stars in which edges may be subdivided). Results from Arnborg, Lagergren and Seese [1] and Johnson et al. [18] show the first two properties. Theorems 1.1–1.3 show the last property. Thus, we obtain:

▶ **Corollary 1.6.** *For every finite set of graphs $\mathcal{H}$,* Edge Multiway Cut *and both versions of* Node Multiway Cut *on $\mathcal{H}$-subgraph-free graphs are polynomial-time solvable if $\mathcal{H}$ contains a graph from $\mathcal{S}$, and* NP*-complete otherwise.*

2 The Proof of Theorem 1.1

In this section, we show that Edge Multiway Cut is NP-complete on subcubic graphs. We reduce the problem from Planar 2P1N-3SAT, which is a restricted version of 3-SAT. Given a CNF-formula Φ with the set of variables X and the set of clauses C, the *incidence graph* of the formula is the graph $G_{X,C}$ which is a bipartite graph with one of the partitions containing a vertex for each variable and the other partition containing a vertex for each clause of Φ. There exists in $G_{X,C}$ an edge between a variable-vertex and a clause-vertex if and only if the variable appears in the clause. We define Planar 2P1N-3SAT as follows.

Planar 2P1N-3SAT
Input: A set $X = \{x_1, \dots, x_n\}$ of variables and a CNF formula Φ over X and clause set C with each clause containing at most three literals and each variable occurring twice positively and once negatively in Φ such that $G_{X,C}$ is planar.
Question: Is there an assignment $\mathcal{A} : X \to \{0, 1\}$ that satisfies Φ?

The above problem was shown to be NP-complete by Dahlhaus et al. [13]. By their construction, each variable occurs in at least two clauses having size 2. This property becomes important later in our NP-completeness proof.

We need two further definitions. Recall that in Weighted Edge Multiway Cut, we are given a function $\omega : E(G) \to \mathbb{Q}^+$ in addition to G, T, k. The goal is to decide if (G, T) admits an edge multiway cut of total weight at most k. If the image of ω is the set X, we denote the corresponding Weighted Edge Multiway Cut problem as X-Edge Multiway Cut. Also note that if an edge/node multiway cut S has smallest possible size (weight) among all edge/node multiway cuts for the pair (G, T), then S is a *minimum(-weight)* edge/node multiway cut.

We show the reduction in two steps. In the first step, we reduce from Planar 2P1N-3SAT to $\{1, 2, 3, 6\}$-Edge Multiway Cut restricted to planar graphs of maximum degree 5 where the terminals all have degree 3. In the second step, we show how to make the instance unweighted while keeping it planar and making its maximum degree bounded above by 3.

▶ **Theorem 1.1** (Restated). Edge Multiway Cut *is* NP*-complete for planar subcubic graphs.*

Proof. Clearly, Edge Multiway Cut is in NP. We reduce Edge Multiway Cut from Planar 2P1N-3SAT. Let Φ be a given CNF formula with at most three literals in each clause and each variable occurring twice positively and once negatively.

We assume that each clause has size at least 2 and every variable occurs in at least two clauses of size 2. Let $X = \{x_i \mid 1 \leq i \leq n\}$ be the set of variables in Φ and $C = \{c_j \mid 1 \leq j \leq m\}$ be the set of clauses. We assume that the incidence graph $G_{X,C}$ is planar. By the reduction of Dahlhaus et al. [13], Planar 2P1N-3SAT is NP-complete for such instances.

We now describe the graph construction. For each vertex of $G_{X,C}$ corresponding to a clause c_j in C, we create a clause gadget (depending on the size of the clause), as in Figure 1. For each vertex of $G_{X,C}$ corresponding to a variable $x_i \in X$, we create a variable gadget, also shown in Figure 1. The gadgets have two terminals each (marked as red squares in Figure 1), a positive and a negative one. In a variable gadget, the positive terminal is attached to the diamond and the negative one to the hat, by edges of weight 3; refer to Figure 1. In a clause gadget, each literal corresponds to a triangle, with these triangles connected in sequence, and the positive and negative terminal are attached to triangles at the start and end of the sequence, again by edges of weight 3.

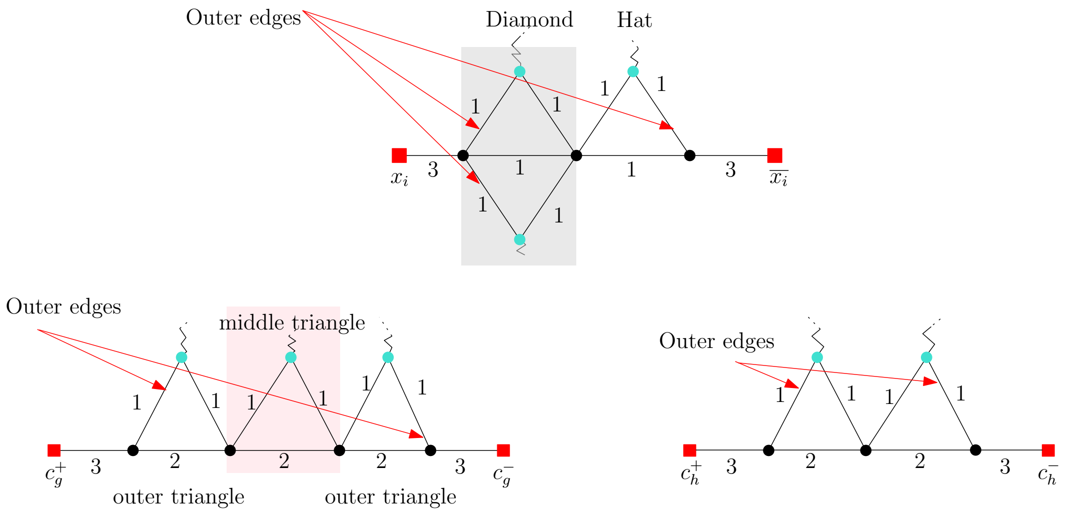


Figure 1 The gadgets for the variables (top) as well as those for the clauses (bottom). The bottom-left gadget corresponds to a clause with three literals whereas the bottom-right one corresponds to a clause with two literals. The terminals are depicted as red squares.

Each degree-2 vertex in a gadget (marked blue in Figure 1) acts as a connector. For a variable x_i, if $x_i \in c_j$ and $x_i \in c_k$ for clauses c_j, c_k, then we connect the degree-2 vertices of the diamond of x_i to some degree-2 vertex of the gadgets for c_j and c_k, each by an edge of weight 6. If $\overline{x_i} \in c_l$ for clause c_l, then we connect the degree-2 vertex of the hat of x_i and some degree-2 vertex on the gadget for c_l, again by an edge of weight 6. These connecting edges are called *links*. A link structure is depicted in Figure 2, while an example of such variable and clause connections is depicted in Figure 4. By the assumptions on Φ, we can create the links such that each degree-2 vertex in the variable gadget is incident on exactly one link and corresponds to one occurrence of the variable. Similarly, each degree-2 vertex of a clause gadget is incident on exactly one link.

The graph thus created is denoted by G. We can construct G in such a way that it is planar, because $G_{X,C}$ is planar and has maximum degree 3. Note that G has maximum degree 5. Let T be the set of terminals in the constructed graph G. Note that G has a total of $2n + 2m$ terminals.

We observe that all edges in G have weight at most 6. Non-terminal vertices are incident on edges of total weight at most 8. Crucially, terminals are incident on edges of total weight at most 3.

We introduce some extra notions to describe the constructed graph G. The edges of the two triangles adjacent to a link are called *connector edges*. The edge of such a triangle that is not adjacent to the link is called the *base* of the triangle. The connector edges closest to the terminals are called *outer edges*, as indicated in Figure 1. The structure formed by the two pairs of connector edges and the link is called the *link structure*; see Figure 2. Since each variable occurs twice positively and once negatively in Φ, the constructed graph G has exactly $3n$ link structures.

We now continue the reduction to obtain an unweighted planar subcubic graph. We replace all the edges in G of weight greater than 1 by as many parallel edges between their end-vertices as the weight of the edge. Each of these parallel edges has weight 1. We refer to this graph as G'. Next, for each vertex v in G' of degree greater than 3, we replace v by a large honeycomb (hexagonal grid), as depicted in Figure 3, of 1000×1000 cells (these

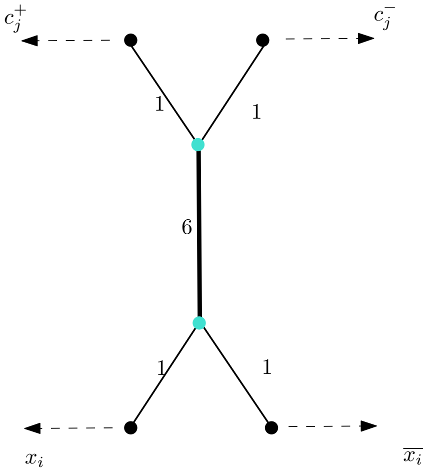


Figure 2 The figure shows a link structure formed by the connector edges of a clause-triangle and its corresponding variable-triangle. The two bases that complete the triangles are not drawn.

numbers are picked for convenience and not optimized). The neighbours of v, of which there are at most eight by the construction of G, are now attached to distinct degree-2 vertices on the boundary of the honeycomb such that the distance along the boundary between any pair of them is 100 cells of the honeycomb. These degree-2 vertices on the boundary are called the *attachment points* of the honeycomb. The edges not belonging to the honeycomb that are incident on these attachment points are called *attaching edges*. In the construction, we ensure that the attaching edges occur in the same cyclical order on the boundary as the edges to the neighbors of v originally occured around v. Let the resultant graph be $\tilde{G}$.

Note that the degree of any vertex in $\tilde{G}$ is at most 3. For terminals, this was already the case in G'. Note that, therefore, terminals were not replaced by honeycombs to obtain $\tilde{G}$. For non-terminals, this is clear from the construction of G' and $\tilde{G}$. Moreover, all the edge weights of $\tilde{G}$ are equal to 1, and thus we can consider it unweighted. Also, all the replacements can be done as to retain a planar embedding of G and hence, $\tilde{G}$ is planar. $\tilde{G}$ has size bounded by a polynomial in $n+m$ and can be constructed in polynomial time. Finally, we set $k = 7n+2m$.

For the sake of simplicity, we shall first argue that Φ is a YES instance of PLANAR 2P1N-3SAT if and only if (G,T,k) is a YES instance of $\{1,2,3,6\}$-EDGE MULTIWAY CUT. Later, we show that the same holds for $\tilde{G}$ by proving that no edge of any of the honeycombs is ever present in any minimum edge multiway cut in $\tilde{G}$.

Suppose that $\mathcal{A}$ is a truth assignment satisfying Φ. Then, we create a set of edges $S \subseteq E(G)$, as follows:

- If a variable is set to “true” by $\mathcal{A}$, then add to S all the three edges of the hat in the corresponding gadget. If a variable is set to “false” by $\mathcal{A}$, then add to S all the five edges of the diamond.
- For each clause, pick a true literal in it and add to S all the three edges of the clause-triangle corresponding to this literal.
- Finally, for each link structure with none of its edges in S yet, add the two connector edges of its clause-triangle to S.

▷ Claim 2.1. S is an edge multiway cut of (G,T) of weight at most $7n+2m$.

Proof. For each variable, either the positive literal is true, or the negative one. Hence, either all the three edges of its hat are in S or all the five edges of the diamond. Therefore, all the paths between terminal pairs of the form $x_i - \overline{x_i}$, for all $1 \leq i \leq n$, are disconnected in $G - S$.

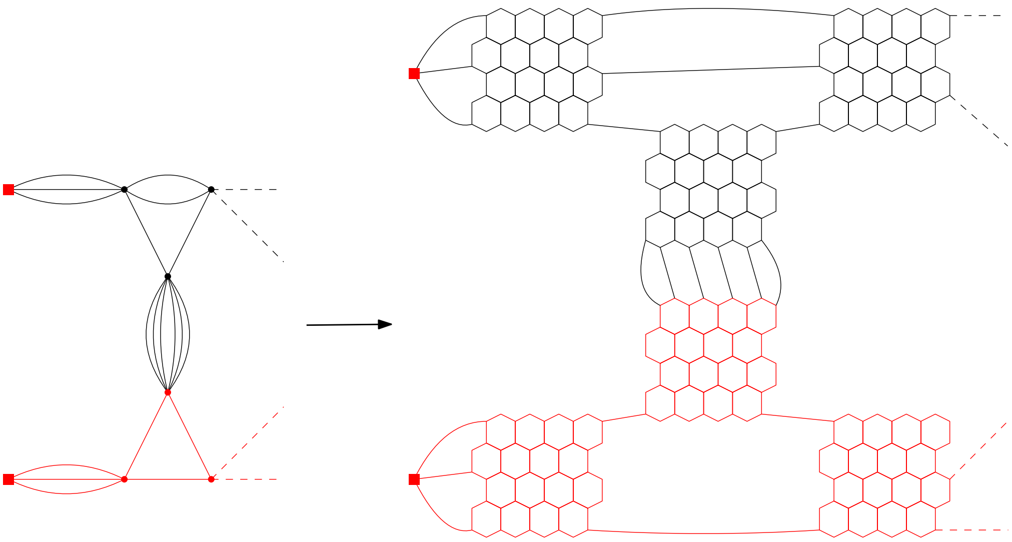


Figure 3 Construction of $\tilde{G}$ from G' by replacing every edge of weight greater than 1 by as many parallel edges as its weight and then replacing the vertices of degree greater than 3 by a honeycomb of size 1000 × 1000.

Consider the link structure in Figure 2. By our choice of S, at least one endpoint of each link in $G - S$ is a vertex of degree 1, hence a dead end. Therefore, no path connecting any terminal pair in $G - S$ passes through any link. As all the paths in G between a variable-terminal and a clause-terminal must pass through some link, we know that all terminal pairs of this type are disconnected in $G - S$. Since $\mathcal{A}$ is a satisfying truth assignment of Φ, all the edges of one triangle from every clause gadget are in S. Hence, all the paths between terminal pairs of the form $c_j^+ - c_j^-$, for all $1 \leq j \leq m$, are disconnected in $G - S$. Hence, S is an edge multiway cut.

It remains to show that the weight of S is at most $7n + 2m$. Since $\mathcal{A}$ satisfies each clause of Φ, there are exactly m triangle-bases of weight 2 from the clause gadgets in S. Similarly, the variable gadgets contribute exactly n bases to S. Finally, for each of the $3n$ link structures, by the definition of S and the fact that $\mathcal{A}$ is a satisfying assignment, either the two connector edges of the variable-triangle are in S or the two connector edges of the clause-triangle. Together, they contribute a weight of $6n$ to the total weight of S. Therefore, S is an edge multiway cut in G of weight at most $7n + 2m$. ◁

Conversely, assume that (G, T, k) is a YES instance of $\{1, 2, 3, 6\}$-EDGE MULTIWAY CUT. Hence, there exists an edge multiway cut of (G, T) of weight at most $7n + 2m$. We shall demonstrate an assignment that satisfies Φ. Before that, we shall discuss some structural properties of a minimum-weight edge multiway cut. In the following arguments, we assume that the clauses under consideration have size three, unless otherwise specified. While making the same arguments for clauses of size 2 is easier, we prefer to argue about clauses of size three for generality.

▷ Claim 2.2 (adapted from [13]). If e is an edge in G incident on a non-terminal vertex v of degree > 2 such that e has weight greater than or equal to the sum of the other edges incident on v, then there exists a minimum-weight edge multiway cut in G that does not contain e.

The above claim implies that there exists a minimum-weight multiway cut containing no such edge e. To see this, note that an iterative application of the local replacement used in Claim 2.2 would cause a conflict in the event that the replacement is cyclical. Suppose

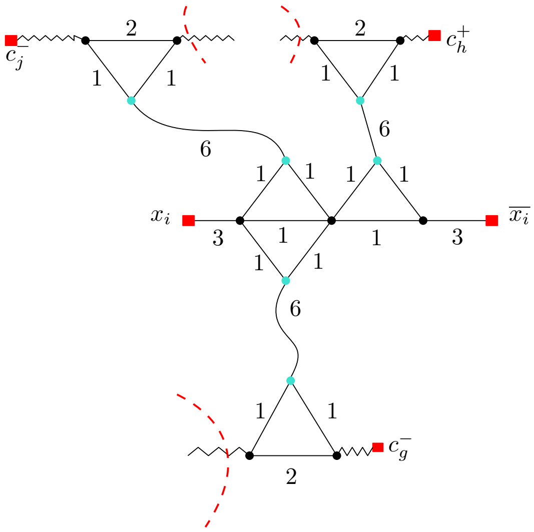


Figure 4 Shown in the figure is the variable interface of x_i. The positive literal x_i occurs in the clauses c_j and c_g, whereas $\overline{x_i}$ occurs in c_h. No terminal is reachable from the vertex closest to the red dashed lines in the direction of the paths crossed by it.

that the edges are replaced in the sequence $e \to e_1 \to \ldots \to e_r \to e$. Then the weight of e_1, denoted by $w(e_1)$ must be strictly less than the weight of e. Similarly, $w(e_i) < w(e_j)$ for $i < j$. This would mean that $w(e) < w(e)$, which is a contradiction.

▷ Claim 2.3 ([13]). If a minimum-weight edge multiway cut contains an edge of a cycle, then it contains at least two edges from that cycle.

It follows from Claim 2.2 and the construction of G that there exists a minimum-weight edge multiway cut for (G, T) that neither contains the edges incident on the terminals nor does it contain the links. Among the minimum-weight edge multiway cuts that satisfy Claim 2.2, we shall select one that contains the maximum number of connector edges and from the ones that satisfy both the aforementioned properties, we shall pick one that contains the maximum number of triangle-bases from clause gadgets of size 2. Let S be a minimum edge multiway cut that fulfills all these requirements.

We say a link e incident on a gadget *reaches* a terminal t if e is the first edge on a path P from the gadget to t and no edge on P is contained in S. A terminal t is *reachable* by a gadget if one of the links incident on the gadget reaches t. Note that, for any terminal t' in the gadget, if t is reached from some incident link by a path P, then P can be extended to a t'-t path in G using only edges inside the gadget. However, among the edges used by such an extension, at least one must belong to S, or else $t = t'$.

▷ Claim 2.4. S contains exactly one base of a triangle from each variable gadget.

Proof. Clearly, S must contain at least one base from each variable gadget, else by the fact that S contains no edges incident on terminals, a path between the terminals in such a gadget would remain in $G - S$.

Suppose that S contains two bases of some variable gadget, say that of x_i. By Claim 2.3, S must also contain at least three connector edges from this variable gadget: at least two connector edges (of the two triangles) of the diamond and at least one connector edge of the hat. We claim that, without loss of generality, at least all the outer connector edges must be in S. If for some triangle the outer connector edge next to terminal t is not in S, then the link incident on this triangle does not reach any terminal $t' \neq t$; otherwise, a t-t' path would remain in $G - S$, a contradiction. Hence, we simultaneously replace all inner connector edges for which the corresponding outer connector edge is not in S by their corresponding outer connector edge. For the resulting set S', the variable terminals of the gadget and their neighbors in G form a connected component of $G - S'$. Since the link incident on a triangle for which the outer connector edge (next to terminal t) was not in S does not reach any terminal $t' \neq t$, S' is feasible. Moreover, it has the same properties we demanded of S. Thus, henceforth, we may assume that all the outer connector edges of the x_i-gadget are in S.

We now distinguish six cases based on how many links of the gadget reach a terminal:

Case 1. *No link of the x_i gadget reaches a terminal.*
We can remove one of the two bases from S without connecting any terminal pairs. This is so because in order to disconnect x_i from $\overline{x_i}$, it suffices for S to contain either the base of the diamond along with the two outer connector edges or the base and outer connector edge of the hat. No other terminal pairs are connected via the gadget by the assumption of this case. Hence, we contradict the minimality of S.

Case 2. *A link of the x_i-gadget reaches at least two distinct terminals.*
By the definition of reaches, this implies that there is a path in $G - S$ between any two of the reached terminals. This contradicts that S is an edge multiway cut for (G, T).

Case 3. *Exactly one link e of the x_i-gadget reaches some terminal t.*
We remove from S the base of a triangle that is not attached to e and add the remaining connector edge of the triangle that is attached to e (if it is not already in S). Consequently, although e reaches t, both connector edges incident on e are in S. Since no other link reached any terminals and x_i remains disconnected from $\overline{x_i}$ in $G - S$, we can obtain an edge multiway cut for (G, T) satisfying Claim 2.2 that has the same or less weight as S, but has strictly more connector edges than S. This is a contradiction to our choice of S.

Case 4. *Exactly two links e, e' of the x_i-gadget reach two distinct terminals t and t', respectively.*
Recall that all three outer connector edges are in S. Now at least one of the inner connector edges of the gadget must be in S, or else t would be connected to t' via this gadget. In particular, both the connector edges of at least one of the two triangles attached to e, e' must be in S. We can remove from S one of the two bases and add instead the remaining connector edge of the other triangle (if it is not already in S). Consequently, although e reaches t and e' reaches t', all connector edges incident on e and e' are in S. Moreover, x_i and $\overline{x_i}$ are not connected to each other in $G - S$, as one base and its corresponding outer connector(s) are still in S. The transformation results in an edge multiway cut for (G, T) satisfying Claim 2.2 that has the same or less weight than S, but has strictly more connector edges than S. This is a contradiction to our choice of S.

Case 5. *All the three links of the x_i-gadget reach distinct terminals t, t', t'', respectively.*
Recall that all three outer connected edges are in S. Now at most one (inner) connector edge of the x_i-gadget is not in S, or else at least one pair of terminals among $\{(t, t'), (t', t''), (t'', t)\}$ would remain connected via the gadget. We replace one of the bases in S with this connector edge (if it is not already in S). The resulting edge multiway cut is no heavier. To see that it is also feasible, note that while t, t', t'' are still reached from the links of the gadget, all the connector edges of this gadget are in the edge multiway cut. The terminals x_i and $\overline{x_i}$

are disconnected from each other in $G - S'$ because one triangle-base and its connectors are still in the edge multiway cut. Hence, we obtain an edge multiway cut for (G, T) satisfying Claim 2.2 that has the same or less weight than S, but with strictly more connector edges than S, a contradiction to our choice of S.

Case 6. *At least two links of the x_i-gadget reach exactly one terminal t outside the gadget.* Recall that every variable occurs in at least two clauses of size 2. Hence, t is reachable via a link from the x_i-gadget to at least one directly linked clause gadget of a clause of size 2. Also recall that S is a minimum-weight edge multiway cut containing the maximum number of bases from clauses of size 2.

Suppose that there exists a size-2 clause gadget c, directly linked to the x_i-gadget, that does not contain t and via which t is reachable from the x_i-gadget. That is, some link reaches t via a path P that contains edges of c, but t is not in c. Then S must contain two base-connector pairs from c; else, some terminal of c would not be disconnected from t in $G - S$. Now remove from S the base of one of the two triangles of c and add the remaining two connector edges of c. This does not increase the weight, as the base of the clause-triangle has weight 2 and the connectors have weight 1 each. The only terminal pair that could get connected by the transformation is the pair of terminals on c itself. However, one of the bases is still in the transformed cut. This new cut contradicts our choice of S, as it has strictly more connector edges and satisfies the other conditions.

Suppose t is contained in one of the size-2 clause gadgets, c', directly linked to the x_i-gadget. If the link between the x_i-gadget and c' is not one of the links meant in the assumption of this case, then the situation of the previous paragraph holds and we obtain a contradiction. Thus, t is reachable from the x_i-gadget via both links of c'. Hence, a base-connector pair of the triangle of c' that t is not attached to must be in S. Consider the link of the x_i-gadget that is not attached to c' but reaches t and let P be a corresponding path, starting at this link and going to t. Note that P passes through a clause gadget c'' directly linked to the x_i-gadget. If c'' is a size-2 clause gadget, then we obtain a contradiction as before. Hence, c'' corresponds to a size-3 clause (as in Figure 5). Since P must either enter or leave c'' through one of its outer triangles, a base-connector pair of at least one outer triangle of c'' must be in S, or the attached terminal would reach t in $G - S$, contradicting that S is an edge multiway cut for (G, T). Let Λ be such an outer triangle (see Figure 5).

We argue that, without loss of generality, S contains a base-connector pair of the other outer triangle, Δ. Suppose not. Then, in particular, the base of Δ is not in S. If P passes through the link attached to Δ, then one of the endpoints of the base of Δ must be on P. Since the base of Δ is not in S, the terminal t'' next to Δ remains connected to t in $G - S$, a contradiction. Hence, P must either enter or exit c'' via the link attached to its middle triangle μ. Moreover, S must contain a base-connector pair of μ (see Figure 5), or t'' would still reach t in $G - S$. We now modify S to obtain a set S'. If both connector edges of Δ are in S, then replace the base of μ by the base of Δ to obtain S'. Then all edges of Δ are in S'. Otherwise, no edge of Δ is in S and thus no terminal is reachable via the link attached to Δ (or it would be connected to t'' in $G - S$). So, we replace the base-connector pair of μ by a base-connector pair of Δ to obtain S'. Then S' is an edge multiway cut for (G, T) of the same weight at S that has the same properties as S. Hence, we may assume $S = S'$. Then S contains a base-connector pair of Δ.

Now remove from S the base and connector edge of Λ. Then t and t' become connected to each other in $G - S$, but not to any other terminal, or that terminal would already be connected to t in $G - S$. Now add the base and outer connector edge of the triangle in c' that t is attached to. This restores that S is an edge multiway cut for (G, T). The edge

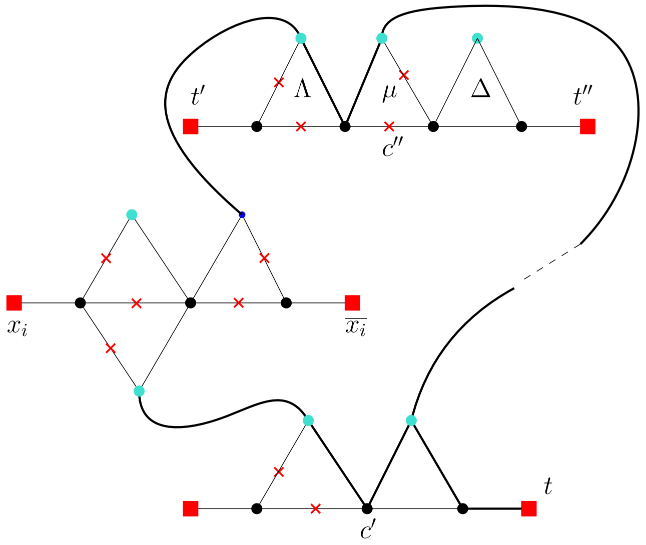


Figure 5 A variable gadget for x_i for which two of its bases are in S. There is a terminal t reachable via (at least) two links of the x_i-gadget. Moreover, t appears in a clause gadget c' corresponding to a clause of size 2 that is directly linked to the x_i-gadget.

multiway cut we obtain has the same weight as S and satisfies Claim 2.2. Moreover, it has no less connectors than S but contains at least one more base of a clause gadget of size 2. Hence, we obtain a contradiction to our choice of S. ◁

We now focus on the link structures.

▷ Claim 2.5 (proof omitted). There cannot exist a link structure in G that contributes less than two edges to S and for which the clause-triangle of the link structure contributes no connector edges to S.

▷ Claim 2.6. S contains at least two edges from each link structure.

Proof. Suppose that there exists a link structure ℓ that contributes less than two edges to S. Suppose that ℓ connects the clause gadget c and the variable gadget x_i. By Claim 2.5, we know that the clause-triangle of ℓ must contribute an edge e to S. Therefore, none of the connectors of the variable-triangle attached to ℓ are in S. As a result, the variable-terminal of the x_i-gadget attached to ℓ, say we call it t, is reachable from c via ℓ.

By Claim 2.3 and the fact that only e is in S, the base of the clause-triangle must also be in S. We do the following replacement: remove from S the base-connector pair of the clause-triangle and add the base and (possibly two) connectors of the variable-triangle of ℓ, as follows. If the variable-triangle of ℓ is part of a diamond, then we add to S the base and two outer connectors, thereby getting an edge multiway cut of equal weight but strictly more connectors. If the variable-triangle is a hat, then we add to S the base and outer connector of the hat, obtaining an edge multiway cut for (G, T) of strictly smaller weight than S. If we can show that the resultant edge multiway cut is feasible, we obtain a contradiction in either scenario. We claim that such a replacement does not compromise the feasibility of S.

Let a, b be the endpoints of the base of the clause-triangle of ℓ, where a is the endpoint on which e is incident (see Figure 6). Note that no terminal other than t should be reachable in $G - S$ from b; else, there would be a path from t to that terminal via ℓ. In particular, the

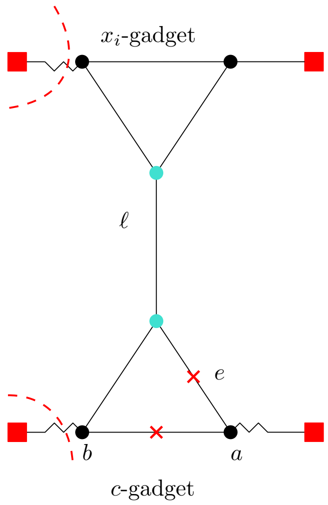


Figure 6 A link structure with the variable gadget of x_i at the top and its clause gadget for c at the bottom. The crossed-out edges are assumed to be in the minimum edge multiway cut S. The dashed red lines depict that the terminals cannot be reached from the vertices a or b.

terminal of the clause gadget for c on the side of b can not be reached in $G - S$ from the vertex b. By removing the base-connector pair of the clause-triangle of ℓ, we may expose the clause-terminal on the side of the vertex a (or another terminal outside c) to t. However, by adding the base and (possibly two) connectors closest to t, we disconnect any path between this terminal and t. Since we did not modify the cut in any other way, no new connections would have been made. This shows the feasibility of the resultant edge multiway cut and thus proves our claim. ◁

▷ Claim 2.7. If there exists an edge multiway cut of weight at most $7n + 2m$ for (G, T), then there exists a satisfying truth assignment for Φ.

Proof. Let S be the edge multiway cut defined before. The immediate consequence of Claims 2.4 and 2.6 is that the weight of S is at least $n + 2 \cdot (3n) = 7n$. S must also contain at least one base per clause gadget lest the two terminals on a clause gadget remain connected. Therefore, its weight is at least $7n + 2m$. Since it is an edge multiway cut of weight at most $7n + 2m$, it has exactly one base per clause gadget.

We also claim that for each link structure, if one of the triangles attached to it has its base in S, then the other one cannot: note that if both the triangles had their bases in S, then each of them would also have a connector edge in S by Claim 2.3. By Claim 2.6 and the assumption that the weight of S is at most $7n + 2m$, the other two connector edges of the link structure are not in S. Since at most one base per variable/clause gadget can be in S, there would be a path between one of the variable-terminals and one of the clause-terminals in the linked gadgets through the link structure, a contradiction to S being an edge multiway cut for (G, T). Figure 7 shows one such case.

We now define the truth assignment $\mathcal{A}$. For each variable-terminal, if the diamond has its base in S, we make it "false", otherwise if the hat has its base in S we make it "true". Each clause gadget has exactly one triangle contributing its base to S. From the above argument, we know that the variable-triangle linked to this clause-triangle must not contribute its base to S. Hence, every clause gadget is attached to one literal triangle such that its base is not in S, and is therefore "true". Hence, every clause is satisfied by the truth assignment $\mathcal{A}$ and Φ is a YES instance of PLANAR 2P1N-3SAT. ◁

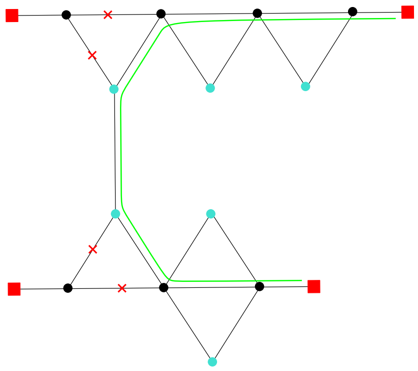


Figure 7 The figure shows a link structure with the variable gadget at the bottom and its connected clause gadget at the top. The crossed-out red edges are the ones contained in the minimum edge multiway cut S. The green curve shows the existence of a path between a variable-terminal and a clause-terminal.

The above implies that $\{1, 2, 3, 6\}$-Edge Multiway Cut is NP-complete on planar subcubic graphs. We now proceed to prove that (unweighted) Edge Multiway Cut is NP-complete on planar subcubic graphs. The proof follows from the claim below, which states that the honeycombs of $\tilde{G}$ (defined before) do not contribute any edge to any minimum edge multiway cut for $(\tilde{G}, T)$.

▷ Claim 2.8 (proof omitted). Any minimum edge multiway cut for $(\tilde{G}, T)$ does not contain any of the honeycomb edges.

By the construction of $\tilde{G}$ and Claims 2.1, 2.7, and 2.8, we conclude that Edge Multiway Cut is NP-complete on planar subcubic graphs. ◀

3 Proofs of Theorems 1.2 and 1.3

We first prove Theorem 1.2.

▶ **Theorem 1.2** (Restated). Node Multiway Cut with Deletable Terminals *is* NP*-complete for planar subcubic graphs.*

Proof. It is readily seen that Node Multiway Cut with Deletable Terminals belongs to NP. We now reduce from Vertex Cover on planar subcubic graphs, which is known to be NP-complete [21]. Let G be the graph of an instance of this problem. We keep the same graph, but set $T = V(G)$. Since any two adjacent vertices are now adjacent terminals, any vertex cover in G corresponds to a node multiway cut for (G, T). The result follows. ◀

To prove Theorem 1.3, we first make the following observation (proof omitted).

▶ **Proposition 3.1.** Node Multiway Cut *is* NP*-complete for planar graphs of maximum degree* 4.

We also need the following lemma from Johnson et al. [18].

▶ **Lemma 3.2.** *If* Edge Multiway Cut *is* NP*-complete for a class $\mathcal{H}$ of graphs, then it is also* NP*-complete for the class of graphs consisting of the* 1*-subdivisions of the graphs of $\mathcal{H}$.*

We are now ready to prove Theorem 1.3.

▶ **Theorem 1.3** (Restated). Node Multiway Cut *is* NP*-complete for planar subcubic graphs.*

Proof. It is readily seen that Node Multiway Cut belongs to NP. In Theorem 1.1, we showed that Edge Multiway Cut is NP-complete on the class of planar subcubic graphs. We will now reduce Node Multiway Cut from Edge Multiway Cut restricted to the class of planar subcubic graphs. Let G be a planar subcubic graph with a set of terminals T.

From G, we create an instance of Node Multiway Cut by the following operations; here, the *line graph* of a graph $G = (V, E)$ has E as vertex set and for every pair of edges e and f in G, there is an edge between e and f in the line graph of G if and only if e and f share an end-vertex.

- We construct the 2-subdivision of G, which we denote by G'.
- Next, we construct the line graph of G', which we denote by L.
- Finally, we create the terminal set of L as follows: for each terminal t in G', consider the edges incident on it. In the line graph L, these edges must form a clique, K_i for $i \in \{1, 2, 3\} : i = \deg(t)$. In this clique, we pick one vertex and make it a terminal. We denote the terminal set in L by T_L.

Note that L is planar, as G' is planar and every vertex in G' has degree at most 3 [25]. Note also that L is subcubic, as every edge in G' has one end-vertex of degree 2 and the other end-vertex of degree at most 3. Moreover, L and T_L can be constructed in polynomial time.

▷ Claim 3.3 (proof omitted). There exists an edge multiway cut of (G, T) of size at most k if and only if there exists a node multiway cut of (L, T_L) of size at most k.

By our construction and Claim 3.3, Node Multiway Cut is NP-complete on the class of planar subcubic graphs. ◀

4 Conclusions

We proved that Edge Multiway Cut and both versions of Node Multiway Cut are NP-complete for planar subcubic graphs. We also showed that these results filled complexity gaps in the literature related to maximum degree, $\mathcal{H}$-topological-minor-free graphs and $\mathcal{H}$-subgraph-free graphs. The last dichotomy result assumes that $\mathcal{H}$ is a finite set of graphs. We therefore pose the following challenging question.

▶ **Open Problem 1.** *Classify the complexity of* Edge Multiway Cut *and both versions of* Node Multiway Cut *for $\mathcal{H}$-subgraph-free graphs when $\mathcal{H}$ is infinite.*

An answer to Open Probem 1 will require novel insights into the structure of $\mathcal{H}$-subgraph-free graphs.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

2 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Algorithmica*, 84:1385–1417, 2022.

3 Stan Birchfield and Carlo Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Proc. ICCV 1999*, pages 489–495. IEEE Computer Society, 1999.

4 Hans L. Bodlaender, Édouard Bonnet, Lars Jaffke, Dusan Knop, Paloma T. Lima, Martin Milanic, Sebastian Ordyniak, Sukanya Pandey, and Ondrej Suchý. Treewidth is NP-complete on cubic graphs. In *Proc. IPEC 2023*, volume 285 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

5 Édouard Bonnet, Dibyayan Chakraborty, and Julien Duron. Cutting Barnette graphs perfectly is hard. In *Proc. WG 2023*, volume 14093 of *LNCS*, pages 116–129. Springer, 2023.

6 Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. In *Proc. CVPR 1998*, pages 648–655. IEEE Computer Society, 1998.

7 Gruia Călinescu and Cristina G. Fernandes. Multicuts in unweighted digraphs with bounded degree and bounded tree-width. *Electronic Notes in Discrete Mathematics*, 7:194–197, 2001.

8 Gruia Călinescu, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for Multiway cut. *Journal of Computer and System Sciences*, 60:564–574, 2000.

9 Yixin Cao, Jianer Chen, and Jia-Hao Fan. An $O^*(1.84^k)$ parameterized algorithm for the Multiterminal Cut problem. *Information Processing Letters*, 114:167–173, 2014.

10 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the Minimum Node Multiway Cut problem. *Algorithmica*, 55:1–13, 2009.

11 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of Directed Multiway Cut parameterized by the size of the cutset. *SIAM Journal on Computing*, 42:1674–1696, 2013.

12 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On Multiway Cut parameterized above lower bounds. *ACM Transactions on Computation Theory*, 5:3:1–3:11, 2013.

13 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23:864–894, 1994.

14 Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

15 Esther Galby, Dániel Marx, Philipp Schepper, Roohani Sharma, and Prafullkumar Tale. Domination and cut problems on chordal graphs with bounded leafage. In *Proc. IPEC 2022*, volume 249 of *LIPIcs*, pages 14:1–14:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

16 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms*, 50:49–61, 2004.

17 David Hartvigsen. The Planar Multiterminal Cut problem. *Discrete Applied Mathematics*, 85:203–222, 1998.

18 Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs I: The framework. *arXiv*, 2022. `arXiv:2211.12887`.

19 Philip N. Klein and Dániel Marx. Solving Planar k-Terminal Cut in $O(n^{c\sqrt{k}})$ time. In *Proc. ICALP 2012*, volume 7391 of *LNCS*, pages 569–580. Springer, 2012.

20 Dániel Marx. A tight lower bound for Planar Multiway Cut with fixed number of terminals. In *Proc. ICALP 2012*, volume 7391 of *LNCS*, pages 677–688. Springer, 2012.

21 Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82:102–117, 2001.

22 Sukanya Pandey and Erik Jan van Leeuwen. Planar Multiway Cut with terminals on few faces. In *Proc. SODA 2022*, pages 2032–2063. SIAM, 2022.

23 Charis Papadopoulos and Spyridon Tzimas. Subset feedback vertex set on graphs of bounded independent set size. *Theoretical Computer Science*, 814:177–188, 2020.

24 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41:92–114, 1986.

25 J. Sedlaáček. Some properties of interchange graphs. In *Theory of Graphs and Its Applications*, pages 145–150. Academic Press, 1964.

26 H.S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, 1977.

27 Martin Škoviera and Peter Varša. NP-completeness of perfect matching index of cubic graphs. In *Proc. STACS 2022*, volume 219 of *LIPIcs*, pages 56:1–56:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

28 Radoslaw Ziemann and Pawel Zylinski. Vertex-edge domination in cubic graphs. *Discrete Mathematics*, 343:112075, 2020.

Parameterized Complexity of Submodular Minimization Under Uncertainty

Naonori Kakimura
Department of Mathematics, Keio University, Yokohama, Japan

Ildikó Schlotter
HUN-REN Centre for Economic and Regional Studies, Budapest, Hungary
Budapest University of Technology and Economics, Hungary

Abstract

This paper studies the computational complexity of a robust variant of a two-stage submodular minimization problem that we call Robust Submodular Minimizer. In this problem, we are given k submodular functions $f_1, \dots, f_k$ over a set family 2^V, which represent k possible scenarios in the future when we will need to find an optimal solution for one of these scenarios, i.e., a minimizer for one of the functions. The present task is to find a set $X \subseteq V$ that is close to *some* optimal solution for each f_i in the sense that some minimizer of f_i can be obtained from X by adding/removing at most d elements for a given integer $d \in \mathbb{N}$. The main contribution of this paper is to provide a complete computational map of this problem with respect to parameters k and d, which reveals a tight complexity threshold for both parameters:

- Robust Submodular Minimizer can be solved in polynomial time when $k \leq 2$, but is NP-hard if k is a constant with $k \geq 3$.
- Robust Submodular Minimizer can be solved in polynomial time when $d = 0$, but is NP-hard if d is a constant with $d \geq 1$.
- Robust Submodular Minimizer is fixed-parameter tractable when parameterized by (k, d).

We also show that if some submodular function f_i has a polynomial number of minimizers, then the problem becomes fixed-parameter tractable when parameterized by d. We remark that all our hardness results hold even if each submodular function is given by a cut function of a directed graph.

2012 ACM Subject Classification Theory of computation → Submodular optimization and polymatroids; Theory of computation → Fixed parameter tractability; Mathematics of computing → Combinatorial optimization

Keywords and phrases Submodular minimization, optimization under uncertainty, parameterized complexity, cut function

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.30

Related Version *Full Version*: https://arxiv.org/abs/2404.07516 [17]

Funding *Naonori Kakimura*: Supported by JSPS KAKENHI Grant Numbers JP22H05001, JP20H05795, and JP21H03397, Japan and JST ERATO Grant Number JPMJER2301, Japan.
Ildikó Schlotter: Supported by the Hungarian Academy of Sciences under its János Bolyai Research Scholarship and its Momentum Programme (LP2021-2).

Acknowledgements The authors are grateful to the organizers of the 14th Emléktábla Workshop which took place in Vác, Hungary in 2023, and provided a great opportunity for our initial discussions.

1 Introduction

This paper proposes a two-stage robust optimization problem under uncertainty. Suppose that we want to find a minimum cut on a directed graph under uncertainty. The uncertainty here is represented by k directed graphs $G_1, \dots, G_k$ on the same vertex set $V \cup \{s, t\}$. That is, we have k possible scenarios of graph realizations in the future. At the moment, we want

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 30; pp. 30:1–30:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to choose an (s,t)-cut in advance, so that after the graph is revealed, we will be able to obtain a minimum (s,t)-cut in the graph with small modification. Therefore, our aim is to find an (s,t)-cut that is close to some minimum (s,t)-cut in each graph G_i for $i = 1,\dots,k$.

Let us formalize this problem. For a vertex set X in a directed graph $G = (V \cup \{s,t\}, E)$, the *cut function* $f : 2^V \to \mathbb{Z}$ is the number of out-going edges from X. Let us denote the family of minimum (s,t)-cuts in G by $\mathcal{C}_{s,t}(G)$, that is, $\mathcal{C}_{s,t}(G) = \{Y \subseteq V : f(Y) \leq f(Y') \ \forall Y' \subseteq V\}$. Given directed graphs $G_1,\dots,G_k$ over a common vertex set $V \cup \{s,t\}$, we want to find a subset $X \subseteq V$ and sets $Y_i \in \mathcal{C}_{s,t}(G_i)$ for each $i \in [k]$ that minimizes $\max_{i\in[k]} |X \triangle Y_i|$ where $\triangle$ stands for symmetric difference and $[k]$ denotes $\{1,\dots,k\}$ for any positive integer k.

We study a natural generalization of this problem where, instead of the cut functions of directed graphs which are known to be submodular [28], we consider arbitrary submodular set functions over some non-empty finite set V. Let $f_1,\dots,f_k : 2^V \to \mathbb{R}$ be k submodular functions. Let $\arg\min f_i = \{Y \subseteq V : f_i(Y) \leq f_i(Y') \ \forall Y' \subseteq V\}$ refer to the set of minimizers of f_i. We want to find a subset $X \subseteq V$ and sets $Y_i \in \arg\min f_i$ for all $i \in [k]$ that

$$\text{minimize} \qquad \max_{i\in[k]} |X \triangle Y_i|.$$

We call the decision version of this problem Robust Submodular Minimizer.

Robust Submodular Minimizer:	
Input:	A finite set V, submodular functions $f_1,\dots,f_k : 2^V \to \mathbb{R}$, and an integer $d \in \mathbb{N}$.
Task:	Find a set $X \subseteq V$ such that for each $i \in [k]$ there exists $Y_i \in \arg\min f_i$ with $|X \triangle Y_i| \leq d$, or detect if no such set exists.

We remark that the min-sum variant of the problem, that is, the problem obtained by replacing the condition $\max_{i\in[k]} |X \triangle Y_i| \leq d$ with $\sum_{i\in[k]} |X \triangle Y_i| \leq d$, was introduced by Kakimura et al. [16], who showed that it can be solved in polynomial time.

1.1 Our Contributions and Techniques

Our contribution is to reveal the complete computational complexity of Robust Submodular Minimizer with respect to the parameters k and d. We also provide an algorithm for the case when one of the submodular functions has only polynomially many minimizers. Our results are as follows:

1. Robust Submodular Minimizer can be solved in polynomial time when $k \leq 2$ (Theorem 6), but is NP-hard if k is a constant with $k \geq 3$ (Corollary 24).
2. Robust Submodular Minimizer can be solved in polynomial time when $d = 0$ (Observation 4), but is NP-hard if d is a constant with $d \geq 1$ (Theorem 20).
3. Robust Submodular Minimizer is fixed-parameter tractable when parameterized by (k,d).
4. Robust Submodular Minimizer is fixed-parameter tractable when parameterized by d, if the size of $\arg\min f_i$ for some $i \in [k]$ is polynomially bounded.

When $k = 1$, Robust Submodular Minimizer is equivalent to the efficiently solvable submodular function minimization problem [20], in which we are given a single submodular function $f : 2^V \to \mathbb{R}$ and want to find a set $X \subseteq V$ in $\arg\min f$. It is not difficult to observe that Robust Submodular Minimizer for $d = 0$ can also be solved in polynomial time by computing a minimizer of the submodular function $\sum_{i=1}^{k} f_i$; see Section 3.1.

The rest of our positive results are based on Birkhoff's representation theorem on distributive lattices [1] that allows us to maintain the family of minimizers of a submodular function in a compact way. Specifically, even though the number of minimizers may be exponential in the input size, we can represent all minimizers as a family of cuts in a directed acyclic graph with polynomial size. As we show in Section 3.1, we can use this representation to solve an instance I of Robust Submodular Minimizer with $k = 2$ by constructing a directed graph with two distinct vertices, s and t, in which a minimum (s, t)-cut yields a solution for I. More generally, Birkhoff's compact representation allows us to reduce Robust Submodular Minimizer for arbitrary k to the so-called Multi-Budgeted Directed Cut problem, solvable by an algorithm due to Kratsch et al. [18], leading to a fixed-parameter tractable algorithm for the parameter (k, d). We note that a similar construction was used to show that the min-sum variant of the problem is polynomial-time solvable [16].

In Section 3.3, we consider the case when one of the k submodular functions has only polynomially many minimizers. As mentioned in [16], Robust Submodular Minimizer is NP-hard even when each submodular function f_i has a unique minimizer. In fact, the problem is equivalent to the Closest String problem over a binary alphabet, shown to be NP-hard under the name Minimum Radius by Frances and Litman [11]. For the case when $|\arg\min f_i|$ is polynomially bounded for some $i \in [k]$, we present a fixed-parameter tractable algorithm parameterized only by d. Our algorithm guesses a set in $\arg\min f_i$ and uses it as an "anchor," then solves the problem recursively by the bounded search-tree technique.

Section 4 contains our NP-hardness results for the cases when either d is a constant at least 1, or k is a constant at least 3. We present reductions from an intermediate problem that may be of independent interest: in this problem, we are given k set families $\mathcal{F}_1, \dots, \mathcal{F}_k$ over a universe V containing two distinguished elements, s and t, with each $\mathcal{F}_i$ containing pairwise disjoint subsets of V; the task is to find a set $X \subseteq V$ containing s but not t that has a bounded distance from each family $\mathcal{F}_i$ for a specific distance measure.

The symbol $\star$ marks statements whose proofs we defer to the full version of our paper [17].

1.2 Related Work

Robust Submodular Minimizer is related to the *robust recoverable* combinatorial optimization problem, introduced by Liebchen et al. [22]. It is a framework of mathematical optimization that allows recourse in decision-making to deal with uncertainty. In this framework, we are given a problem instance with some scenarios and a recovery bound d, and the task is to find a feasible solution X (the first-stage solution) to the instance that can be transformed to a feasible solution Y_i (the second-stage solutions) in each scenario i respecting the recovery bound (e.g., $|X \triangle Y_i| \leq d$ for each i). The cost of the solution is usually evaluated by the sum of the cost of X and the sum of the costs of Y_i's. Robust recoverable versions have been studied for a variety of standard combinatorial optimization problems. Examples include the shortest path problem [5], the assignment problem [10], the travelling salesman problem [12], and others [14, 19, 21]. The setting was originally motivated from the situation where the source of uncertainty was the cost function which changes in the second stage. We consider another situation dealing with *structural uncertainty*, where some unknown set of input elements can be interdicted [8, 15]. Recently, a variant of robust recoverable problems has been studied where certain operations are allowed in the second stage [13].

Reoptimization is another concept related to Robust Submodular Minimizer. In general reoptimization, we are given an instance I of a combinatorial optimization problem and an optimal solution X for I. Then, for a slightly modified instance I' of the problem, we need to make a small change to X so that the resulting solution X' is an optimal (or a good approximate) solution to the modified instance I'. Reoptimization has been studied for several combinatorial optimization problems such as the minimum spanning tree problem [4], the traveling salesman problem [23], and the Steiner tree problem [2].

2 Preliminaries

Graphs and Cuts

Given a directed graph $G = (V, E)$, we write uv for an edge pointing from u to v. For a subset $X \subseteq V$ of vertices in G, let $\delta_G(X)$ denote the set of edges leaving X. If G is an undirected graph, then $\delta_G(X)$ for some set X of vertices denotes the set of edges with exactly one endpoint in X. We may simply write $\delta(X)$ if the graph is clear from the context.

For two vertices s and t in a directed or undirected graph $G = (V, E)$, an *(s,t)-cut* is a set X of vertices such that $s \in X$ but $t \notin X$. A *minimum (s,t)-cut* in G is an (s,t)-cut X that minimizes $|\delta(X)|$. Given a cost function $c\colon E \to \mathbb{R}_+ \cup \{+\infty\}$ on the edges of G where $\mathbb{R}_+$ is the set of all non-negative real numbers, the *(weighted) cut function $\kappa_G : 2^V \to \mathbb{R}_+ \cup \{+\infty\}$* is defined by

$$\kappa_G(X) = \sum_{e \in \delta(X)} c(e). \tag{1}$$

A *minimum-cost (s,t)-cut* is an (s,t)-cut X that minimizes $\kappa_G(X)$.

Distributive Lattices

In this paper, we will make use of properties of finite distributive lattices on a ground set V.

A *distributive lattice* is a set family $\mathcal{L} \subseteq 2^V$ that is closed under union and intersection, that is, $X, Y \in \mathcal{L}$ implies $X \cup Y \in \mathcal{L}$ and $X \cap Y \in \mathcal{L}$. Then $\mathcal{L}$ is a partially ordered set with respect to set inclusion $\subseteq$, and has a unique minimal element and a unique maximal element.

Birkhoff's representation theorem is a useful tool for studying distributive lattices.

▶ **Theorem 1** (Birkhoff's representation theorem [1]). *Let $\mathcal{L} \subseteq 2^V$ be a distributive lattice. Then there exists a partition of V into $U_0, U_1, \dots, U_b, U_\infty$, where U_0 and U_∞ can possibly be empty, such that the following hold:*

(1) *Every set in $\mathcal{L}$ contains U_0.*

(2) *Every set in $\mathcal{L}$ is disjoint from U_∞.*

(3) *For every set $X \in \mathcal{L}$, there exists a set $J \subseteq [b]$ of indices such that $X = U_0 \cup \bigcup_{j \in J} U_j$.*

(4) *There exists a directed acyclic graph $G(\mathcal{L})$ that has the following properties.*

  (a) *The vertex set is $\{U_0, U_1, \dots, U_b\}$.*

  (b) *U_0 is a unique sink[1] of $G(\mathcal{L})$.*

  (c) *For a non-empty set Z of vertices in $G(\mathcal{L})$, Z has no out-going edge if and only if $\bigcup_{U_j \in Z} U_j \in \mathcal{L}$.*

For a distributive lattice $\mathcal{L} \subseteq 2^V$, we call the directed acyclic graph $G(\mathcal{L})$ above a *compact representation* of $\mathcal{L}$. Note that the size of $G(\mathcal{L})$ is $O(|V|^2)$ while $|\mathcal{L}|$ can be as large as $2^{|V|}$.

Submodular Function Minimization

Let V be a non-empty finite set. A function $f\colon 2^V \to \mathbb{R}$ is *submodular* if $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$ for all $X, Y \subseteq V$. A typical example of submodular functions is the cut function κ_G of a directed (or undirected) edge-weighted graph G as defined in (1). If the graph $G = (V \cup \{s,t\}, E)$ contains two distinct vertices, s and t, then we can restrict the cut function to the domain of (s,t)-cuts in the following sense: each $X \subseteq V$ corresponds to an (s,t)-cut $X \cup \{s\}$ in G; then the function $\lambda_G : 2^V \to \mathbb{R}_+ \cup \{+\infty\}$ defined by $\lambda_G(X) = \kappa_G(X \cup \{s\})$ is submodular.

[1] A *sink* is a vertex of out-degree zero.

When we discuss computations on a submodular function $f\colon 2^V \to \mathbb{R}$, we assume that we are given a *value oracle* of f. A value oracle takes $X \subseteq V$ as an input, and returns the value $f(X)$. Assuming that we are given a value oracle, we can minimize a submodular function in polynomial time. The currently fastest algorithm for submodular function minimization was given by Lee et al. [20] and runs in $\tilde{O}(n^3\mathrm{EO} + n^4)$ time, where $n = |V|$ and EO is the query time of a value oracle.

Let $f\colon 2^V \to \mathbb{R}$ be a submodular function. A subset $Y \subseteq V$ is a *minimizer* of the function f if $f(Y) \le f(Y')$ for all $Y' \subseteq V$. The set of minimizers of f is denoted by $\arg\min f$. The following is a well-known fact on submodular functions.

▶ **Lemma 2** (See e.g., [28]). *Let $f\colon 2^V \to \mathbb{R}$ be a submodular function. Then* $\arg\min f$ *forms a distributive lattice.*

A compact representation of the distributive lattice $\arg\min f$ can be constructed in $\tilde{O}(n^5\mathrm{EO} + n^6)$ time via Orlin's submodular function minimization algorithm [25]. See [24, Notes 10.11–10.12]. We will assume that the submodular functions given in our problem instances are given via their compact representation.

As a special case, consider minimum (s,t)-cuts in a directed graph $G = (V \cup \{s,t\}, E)$ with a positive cost function c on its edges. By Lemma 2, the family of minimum (s,t)-cuts forms a distributive lattice. A compact representation for this lattice can be constructed from a maximum flow in the (s,t)-network in linear time [26]. Thus the running time is dominated by the maximum flow computation, and this can be done in $|E|^{1+o(1)}$ time [6].

Parameterized Complexity

In parameterized complexity, each input instance I of a *parameterized problem* Q is associated with a *parameter* k, usually an integer or a tuple of integers, and we consider the running time of any algorithm solving Q as not only a function of the input length $|I|$, but also as a function of the parameter k. An algorithm for Q is *fixed-parameter tractable* or FPT, if it runs in time $g(k)|I|^{O(1)}$ for some computable function g. Such an algorithm can be efficient in practice if the parameter is small. See the books [7, 9] for more background.

3 Algorithms for Robust Submodular Minimizer

In this section, we present algorithms for Robust Submodular Minimizer. We start with a construction that we will use in most of our algorithms. Let $I_{\mathrm{RSM}} = (V, f_1, \dots, f_k, d)$ be our input instance.

For each $i \in [k]$, let $\mathcal{L}_i = \arg\min f_i$ denote the set of minimizers. By Lemma 2, using Birkhoff's representation theorem we may assume that f_i is given through a compact representation $G(\mathcal{L}_i)$ of $\mathcal{L}_i$, whose vertex set is $\{U_0^i, U_1^i, \dots, U_{b_i}^i\}$ with $U_\infty^i = V \setminus \bigcup_{j=0}^{b_i} U_j^i$.

We then construct a directed graph G^i from $G(\mathcal{L}_i)$ by expanding each vertex in $G(\mathcal{L}_i)$ to a complete graph. More precisely, G^i has vertex set $V^i \cup \{s,t\}$ where $V^i = \{v^i : v \in V\}$ is a copy of V, and its edge set E^i is defined as follows.

- $u^iv^i \in E^i$ if $u, v \in U_j^i$ for some $j \in \{0, 1, \dots, b_i, \infty\}$.
- $u^iv^i \in E^i$ for any $u \in U_j^i$ and $v \in U_{j'}^i$ if $G(\mathcal{L}_i)$ has an edge from U_j^i to $U_{j'}^i$.
- $u^is \in E^i$ and $su^i \in E^i$ if $u \in U_0^i$.
- $u^it \in E^i$ and $tu^i \in E^i$ if $u \in U_\infty^i$.

We define the function $\lambda_i : 2^{V^i} \to \mathbb{Z}_+$ so that $\lambda_i(X) = |\delta_{G^i}(X \cup \{s\})|$ for a subset $X \subseteq V^i$. Then it is observed below that each subset $X \subseteq V^i$ with $\lambda_i(X) = 0$ corresponds to a minimizer of f_i.

▶ **Lemma 3** ([16, Lemma 3.2]). *Let X be a subset in V, and $X^i = \{v^i \in V^i : v \in X\}$ its copy in G^i. Then $\lambda_i(X^i) = 0$ if and only if $X \in \mathcal{L}_i$.*

The rest of the section is organized as follows. In Section 3.1 we present polynomial-time algorithms for the cases $d = 0$ and $k = 2$. In Section 3.2 we give an FPT algorithm for the combined parameter (k, d). Section 3.3 deals with the case when some function f_i has only polynomially many minimizers, allowing for an FPT algorithm with parameter d.

3.1 Polynomial-time algorithms

We start by observing that the case $d = 0$ is efficiently solvable by computing a minimizer for the function $\sum_{i \in [k]} f_i$ which is also submodular.

▶ **Observation 4** ($\star$). *Robust Submodular Minimizer can be solved in polynomial time if $d = 0$.*

Next, we show that the problem is polynomial-time solvable when $k = 2$. We will need the following intuitive fact.

▶ **Proposition 5** ($\star$). *Let Y_1, Y_2 be two subsets of a set V. Then $|Y_1 \triangle Y_2| \leq 2d$ if and only if there exists a set $X \subseteq V$ such that $|X \triangle Y_i| \leq d$ for each $i \in \{1, 2\}$.*

▶ **Theorem 6.** *Robust Submodular Minimizer for $k = 2$ can be solved in polynomial time via a maximum flow computation.*

Proof. Let our instance be $I_{\text{RSM}} = (V, f_1, f_2, d)$. Using the method explained at the beginning of Section 3, we construct the directed graphs $G^1 = (V^1 \cup \{s, t\}, E^2)$ and $G^2 = (V^2 \cup \{s, t\}, E^2)$ for f_1 and f_2. We then construct a directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ by identifying s, as well as t, in G^1 and G^2, and then connecting the corresponding copies of each vertex with a bidirected edge. That is, $\tilde{V} = V^1 \cup V^2 \cup \{s, t\}$ and $\tilde{E} = E^1 \cup E^2 \cup E'$ where $E' = \big\{v^1 v^2, v^2 v^1 : v \in V\big\}$. We set $c(e) = +\infty$ for all edges $e \in E^1 \cup E^2$, and we set $c(e) = 1$ for all edges $e \in E'$.

We next compute a minimum-cost (s, t)-cut Z in the graph $\tilde{G}$ with cost function c using standard flow techniques. Let $\kappa_{\tilde{G}}$ denote the cut function in this graph. We will show below that $\kappa_{\tilde{G}}(Z) \leq 2d$ if and only if the answer is "yes".

First suppose that $\kappa_{\tilde{G}}(Z) \leq 2d$. Let $Y_1 = \{v \in V : v^1 \in Z\}$ and $Y_2 = \{v \in V : v^2 \in Z\}$. Since $\delta_{\tilde{G}}(Z)$ has no edges in $E^1 \cup E^2$, we see that $\lambda_i(\{v^i \in V^i : v \in Y_i\}) = 0$ for both $i = 1, 2$, and therefore the set Y_i is in $\mathcal{L}_i$ by Lemma 3. Since $|Y_1 \triangle Y_2| = \kappa_{\tilde{G}}(Z) \leq 2d$, we can compute a set X such that $|X \triangle Y_i| \leq d$ for both $i = 1, 2$ by Proposition 5.

Conversely, let $X \subseteq V$ and $Y_i \in \mathcal{L}_i$ for each $i = 1, 2$ such that $|X \triangle Y_i| \leq d$. Define $Z = \{s\} \cup \{v^1 \in V^1 : v \in Y_1\} \cup \{v^2 \in V^2 : v \in Y_2\}$. Due to Lemma 3 we know that $\lambda_i(\{v^i \in V^i : v \in Y_i\}) = 0$ for both $i = 1, 2$. This implies $\kappa_{\tilde{G}}(Z) = |Y_1 \triangle Y_2| \leq 2d$ where the inequality follows from Proposition 5. ◀

3.2 FPT algorithm for parameter (k, d)

We propose a fixed-parameter tractable algorithm for Robust Submodular Minimizer parameterized by k and d; let $I_{\text{RSM}} = (V, f_1, \ldots, f_k, d)$ denote our instance.

▶ **Theorem 7.** *Robust Submodular Minimizer can be solved in FPT time when parameterized by (k, d).*

To this end, we reduce our problem to the Multi-Budgeted Directed Cut problem [18], defined as follows. We are given a directed graph $D = (V, E)$ with distinct vertices s and t, together with pairwise disjoint edge sets $A_1, \dots, A_k$, and positive integers $d_1, \dots, d_k$. The task is to decide whether D has an (s,t)-cut X such that $|\delta(X) \cap A_i| \le d_i$ for each $i \in [k]$.

▶ **Proposition 8** (Kratsch et al. [18]). *The Multi-Budgeted Directed Cut problem can be solved in FPT time when the parameter is $\sum_{i=1}^{k} d_i$.*

In fact, we will need to use *forbidden* edges, so let us define the Multi-Budgeted Directed Cut with Forbidden Edges problem as follows. Given an instance I_{MBC} of Multi-Budgeted Directed Cut and a set F of forbidden edges, find a solution X for I_{MBC} such that $\delta(X)$ is disjoint from F. It is straightforward to solve this problem using the results by Kratsch et al. [18], after replacing each forbidden edge with an appropriate number of parallel edges. Hence, we get the following.

▶ **Proposition 9** ($\star$). *The Multi-Budgeted Directed Cut with Forbidden Edges problem can be solved in FPT time when the parameter is $\sum_{i=1}^{k} d_i$.*

Reduction to Multi-Budgeted Directed Cut with Forbidden Edges

Compute the graph G^i for each $i \in [k]$, as described at the beginning of Section 3. We construct a large directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ as follows. We identify all vertices s (and t, respectively) in the graphs G^i into a single vertex s (and t, respectively). We further prepare another copy of V, which is denoted by $V^* = \{v^* : v \in V\}$. Thus the vertex set of $\tilde{G}$ is defined by $\tilde{V} = \bigcup_{i=1}^{k} V^i \cup V^* \cup \{s, t\}$. The edge set of $\tilde{G}$ consists of E^i and bidirected edges connecting v^* and the copy v^i of v in G^i, for each $i \in [k]$. That is,

$$\tilde{E} = \bigcup_{i=1}^{k} \left(E^i \cup A^i\right) \quad \text{where} \quad A^i = \left\{v^* v^i, v^i v^* : v \in V\right\}.$$

We also set $d_i = d$ for each $i \in [k]$. Consider the instance $I_{\text{MBC}} = (\tilde{G}, s, t, \{A^i\}_{i=1}^{k}, \{d_i\}_{i=1}^{k})$ of multi-budgeted directed cut with $F = \bigcup_{i=1}^{k} E^i$ as forbidden edges; note that its parameter is $k \cdot d$. Theorem 7 immediately follows from Proposition 9 and Lemma 10 below.

▶ **Lemma 10.** *There exists a solution for I_{RSM} if and only if there exists a solution for the instance (I_{MBC}, F) of Multi-Budgeted Directed Cut with Forbidden Edges.*

Proof. Suppose that (I_{MBC}, F) admits a solution. That is, there exists a subset X of $\tilde{V}$ containing s but not t such that $\delta_{\tilde{G}}(X)$ is disjoint from F and satisfies $|\delta_{\tilde{G}}(X) \cap A^i| \le d_i$ for each $i \in [k]$. Define $Y^i = X \cap V^i$ for $i = 1, \dots, k$. Observe that all edges within G^i leaving $Y^i \cup \{s\}$ also leave X in $\tilde{G}$, since $s \in X$ but $t \notin X$. Since all edges in E^i are forbidden edges, we see that $\lambda_i(Y^i) = 0$. Let $Y_i = \{v \in V : v^i \in Y^i\}$, so that Y^i contains the copy of each vertex of Y_i in G^i. Then Y_i is in $\mathcal{L}_i$ by Lemma 3.

Define the subset $X^* = \{v : v^* \in X\}$ of V. Observe that

$$\delta_{\tilde{G}}(X) \cap A^i = \{v^* v^i : v \in X^*, v \notin Y_i\} \cup \{v^i v^* : v \notin X^*, v \in Y_i\}.$$

Therefore, we get that $|X^* \triangle Y_i| = |\delta_{\tilde{G}}(X) \cap A^i| \le d_i = d$ for each $i \in [k]$ as required, so X^* is a solution to our instance I_{RSM} of Robust Submodular Minimizer.

Conversely, let $X \subseteq V$ and $Y_i \in \mathcal{L}_i$ for each $i \in [k]$ such that $|X \triangle Y_i| \leq d$. Define $X^* = \{v^* \in V^* : v \in X\}$ and $Y^i = \{v^i \in V^i : v \in Y_i\}$. Then the set $\tilde{X} = \{s\} \cup X^* \cup \bigcup_{i=1}^k Y^i$ is an (s,t)-cut of $\tilde{G}$ such that

$$\begin{aligned}\delta_{\tilde{G}}(\tilde{X}) \cap A^i &= \{v^*v^i : v^* \in X^*, v \notin Y^i\} \cup \{v^iv^* : v^* \notin X^*, v^i \in Y^i\} \\ &= \{v^*v^i : v \in X, v \notin Y_i\} \cup \{v^iv^* : v \notin X, v \in Y_i\} = X \triangle Y_i.\end{aligned}$$

Hence we obtain $|\delta_{\tilde{G}}(\tilde{X}) \cap A^i| = |X \triangle Y_i| \leq d = d_i$ for each $i \in [k]$. Since Y_i is in $\mathcal{L}_i$, by Lemma 3 we know $\lambda_i(Y^i) = 0$ for each $i \in [k]$. Thus $\delta_{\tilde{G}}(\tilde{X})$ is disjoint from the set F of forbidden edges, and therefore $\tilde{X}$ is indeed a solution to our instance (I_{MBC}, F) of Multi-Budgeted Directed Cut with Forbidden Edges. ◀

3.3 Polynomially many minimizers: FPT algorithm parameterized by d

In this section, we present a fixed-parameter tractable algorithm for the case when our threshold d is small, assuming that $|\mathcal{L}_1|$ can bounded by a polynomial of the input size. Note that even with a much stronger assumption, Robust Submodular Minimizer remains intractable (see also [16]):

▶ **Observation 11.** *Robust Submodular Minimizer is* NP*-hard even if* $|\mathcal{L}_i| = 1$ *for each* $i \in [k]$.

Proof. If $|\mathcal{L}_i| = 1$ for each $i \in [k]$, then there is a unique minimizer $Y_i \subseteq V$ for each f_i, and the problem is equivalent with finding a set $X \subseteq V$ whose symmetric difference is at most d from each of the sets Y_i, $i \in [k]$. This is the Closest String problem over a binary alphabet, shown to be NP-hard under the name Minimum Radius by Frances and Litman [11]. ◀

▶ **Theorem 12.** *Robust Submodular Minimizer can be solved in* $|\mathcal{L}_1|g(d)n^c$ *time where* c *is a constant and* g *is a computable function.*

Let us consider a slightly more general version of Robust Submodular Minimizer which we call Anchored Submodular Minimizer. In this problem, in addition to an instance $I_{\text{RSM}} = (V, f_1, \ldots, f_k, d)$ of Robust Submodular Minimizer, we are given a set $Y_0 \subseteq V$ and integer $d_0 \leq d$, and we aim to find a subset X such that

$$|X \triangle Y_0| \leq d_0 \quad \text{and} \tag{2}$$

$$|X \triangle Y_i| \leq d \quad \text{for some } Y_i \in \mathcal{L}_i, \text{ for each } i \in [k]. \tag{3}$$

Observe that we can solve our instance $I_{\text{RSM}} = (V, f_1, \ldots, f_k, d)$ by solving the instance $(V, f_2, \ldots, f_k, d, Y_0, d_0)$ of Anchored Submodular Minimizer for each $Y_0 \in \mathcal{L}_1$ and $d_0 = d$. Hence, Theorem 12 follows from Theorem 13 below.

▶ **Theorem 13.** *Anchored Submodular Minimizer can be solved in FPT time when parameterized by* d.

To prove Theorem 13, we will use the technique of bounded search-trees. Given an instance $I = (V, f_1, \ldots, f_k, d, Y_0, d_0)$, after checking whether Y_0 itself is a solution, we search for a minimizer $Y_i \in \mathcal{L}_i$ for which $d < |Y_0 \triangle Y_i| \leq d + d_0$. It is not hard to see the following.

▶ **Observation 14.** *If* X *is a solution for an instance* $I = (V, f_1, \ldots, f_k, d, Y_0, d_0)$ *of Anchored Submodular Minimizer, and* $Y_i \in \mathcal{L}_i$ *fulfills* $|X \triangle Y_i| \leq d$*, then for all* $T \subseteq Y_0 \triangle Y_i$ *with* $|T| > d$ *it holds that there exists some* $v \in T$ *with* $v \in X \triangle Y_0$.

Proof. Indeed, assuming that the claim does not hold, we have that $T \cap (Y_0 \setminus Y_i) \subseteq X$ and that $(T \cap (Y_i \setminus Y_0)) \cap X = \emptyset$. From the former, $T \cap (Y_0 \setminus Y_i) \subseteq X \setminus Y_i$ follows, while the latter implies $T \cap (Y_i \setminus Y_0) \subseteq Y_i \setminus X$. Thus,

$$X \triangle Y_i = (X \setminus Y_i) \cup (Y_i \setminus X) \supseteq (T \cap (Y_0 \setminus Y_i)) \cup (T \cap (Y_i \setminus Y_0)) = T \cap (Y_0 \triangle Y_i) = T.$$

Hence, $|X \triangle Y_i| \geq |T| > d$, contradicting our assumption that X is a solution for I. ◀

Our algorithm will compute in $O^*(2^d)$ time[2] a set $T \subseteq Y_0 \setminus Y_i$ of size $d < |T| \leq d + d_0$ that contains some element v fulfilling the above conditions. Then, by setting $Y_0 \leftarrow Y_0 \triangle \{v\}$ and reducing the value of d_0 by one, we obtain an equivalent instance I' of Anchored Submodular Minimizer which we solve by applying recursion.

Description of our algorithm

Our algorithm will make "guesses"; nevertheless, it is a deterministic one, where guessing a value from a given set U is interpreted as branching into $|U|$ branches. We continue the computations in each branch, and whenever a branch returns a solution for the given instance, we return it; if all branches reject the instance (by outputting "No"), we also reject it. See Algorithm ASM for a pseudo-code description.

We start by checking whether Y_0 is a solution for our instance $I = (V, f_1, \ldots, f_k, d, Y_0, d_0)$, that is, whether it satisfies (3). This can be done in polynomial time, since the set function $\gamma_i(Z) = \min\{|Z \triangle Y_i| : Y_i \in \mathcal{L}_i\}$ is known to be submodular and can be computed via a maximum flow computation [16]. If Y_0 satisfies (3), i.e., $\gamma_i(Y_0) \leq d$ for each $i \in [k]$, then we output Y_0; note that (2) is obviously satisfied by Y_0, so Y_0 is a solution for I.

Otherwise, if $d_0 = 0$, then we output "No" as in this case the only possible solution could be Y_0. We proceed by fixing an index $i \in [k]$ such that $\gamma_i(Y_0) > d$, that is, $|Y_0 \triangle Y| > d$ for all minimizers $Y \in \mathcal{L}_i$.

▶ **Observation 15.** *If X is a solution for I that satisfies $|X \triangle Y_i| \leq d$ for some $Y_i \in \mathcal{L}_i$, then $|Y_i \triangle Y_0| \leq d + d_0$.*

Proof. Since X is a solution for I, we have $|X \triangle Y_0| \leq d_0$, and thus the triangle inequality implies $|Y_i \triangle Y_0| \leq |X \triangle Y_i| + |X \triangle Y_0| \leq d + d_0$. ◀

By our choice of i and Observation 15, we know that $d < |Y_0 \triangle Y_i| \leq d + d_0$. We are going to compute a set $T \subseteq Y_0 \triangle Y_i$ with the same bounds on its cardinality, i.e., $d < |T| \leq d + d_0$.

To this end, we compute a compact representation $G(\mathcal{L}_i)$ of the distributive lattice $\mathcal{L}_i$; let $\mathcal{P} = \{U_0, U_1, \ldots, U_b, U_\infty\}$ be the partition of V in this representation.

Next, we proceed with an iterative procedure which also involves a set of guesses. We start by setting $Y = Y_0$ and $T = \emptyset$. We will maintain a family of *fixed sets* from $\mathcal{P}$ for which we already know whether they are in Y_i or not (according to our guesses); initially, no set from $\mathcal{P}$ is fixed.

After this initialization, we start an iteration where at each step we check whether $Y \in \mathcal{L}_i$ or $|T| > d$. If yes, then we stop the iteration. If not, then it can be shown that one of the following conditions holds:

Condition 1: there exists a set $S \in \mathcal{P}$ such that $S \cap Y \neq \emptyset$ and $S \setminus Y \neq \emptyset$;

Condition 2: there exists an edge (S, S') in $G(\mathcal{L}_i)$ for which $S \subseteq Y$ but $S' \cap Y = \emptyset$.

[2] The $O^*()$ notation hides polynomial factors.

If Condition 1 holds for some set $S \in \mathcal{P}$, then we guess whether S is contained in Y_i. If $S \subseteq Y_i$ according to our guesses, then we add $S \setminus Y$ to T; otherwise, we add $S \cap Y$ to T. In either case, we declare S as fixed, and proceed with the next iteration.

By contrast, if Condition 1 fails, but Condition 2 holds for some edge (S, S') in $G(\mathcal{L}_i)$ with endpoints $S, S' \in \mathcal{P}$, then we proceed as follows. If both S and S' are fixed, then we stop and reject the current set of guesses. If S is fixed but S' is not, then we add all elements of S' to T. If S' is fixed but S is not, then we add S to T. If neither S nor S' is fixed, then we guess whether S is contained in Y_i or not, and in the former case we add S' to T, while in the latter case we add S to T. In all cases except for the last one, we declare both S and S' as fixed; in the last case declare only S as fixed.

Next, we modify Y to reflect the current value of T by updating Y to $Y_0 \triangle T$. If $|T| > d + d_0$, then we reject the current branch. If $d < |T| \leq d + d_0$, then we finish the iteration; otherwise, we proceed with the next iteration.

Finally, when the iteration stops, we guess a vertex $v \in T$, define $Y'_{0,v} = Y_0 \triangle \{v\}$ and call the algorithm recursively on the instance $I'_v := (V, f_1, \ldots, f_k, d, Y'_{0,v}, d_0 - 1)$.

Algorithm ASM Solving Anchored Submodular Minimizer on $I = (V, f_1, \ldots, f_k, d, Y_0, d_0)$.

```
for all $j \in [k]$ do compute the value $\gamma_j = \min\{|Y_0 \triangle Y| : Y \in \arg\min f_j\}$.
if $\gamma_j \leq d$ for each $j \in [k]$ then return $Y_0$.
if $d_0 = 0$ then return "No".
Fix an index $i \in [k]$ such that $\gamma_i > d$.
Compute the graph $G(\mathcal{L}_i)$, and let $\mathcal{P}$ be its vertex set.
Set $T := \emptyset$ and $Y := Y_0$, and fixed$(S)$ := false for each $S \in \mathcal{P}$.
while $Y \notin \mathcal{L}_i$ and $|T| \leq d$ do
    if $\exists S \in \mathcal{P} : S \cap Y_0 \neq \emptyset, S \setminus Y_0 \neq \emptyset$ then
        Guess contained$(S)$ from {false, true}.
        if contained$(S)$ = true then set $T := T \cup (S \setminus Y)$.
        else set $T := T \cup (S \cap Y)$.
        Set fixed$(S)$ := true.
    else Find an edge $(S, S') \in G(\mathcal{L}_i)$ such that $S \subseteq Y$ and $S' \cap Y = \emptyset$.
        if fixed$(S)$ = true then
            if fixed$(S')$ = true then return "No".
            else set $T := T \cup S'$ and fixed$(S')$ := true.
        else                                                        ▷ fixed$(S)$ = false.
            if fixed$(S')$ = true then set $T := T \cup S$ and fixed$(S)$ := true.
            else guess contained$(S)$ from {false, true}.
                if contained$(S)$ = true then set $T := T \cup S'$, fixed$(S)$ := fixed$(S')$ := true.
                else set $T := T \cup S$ and fixed$(S)$ := true.
    Set $Y := Y_0 \triangle T$.
    if $|T| > d + d_0$ then return "No".
Guess a vertex $v$ from $T$.
Set $Y'_{0,v} = Y_0 \triangle \{v\}$ and $I'_v = (V, f_1, \ldots, f_k, d, Y'_{0,v}, d_0 - 1)$.
return ASM$(I'_v)$.
```

Proof of Theorem 13. We first prove the correctness of the algorithm. Clearly, for $d_0 = 0$, the algorithm either correctly outputs the solution Y_0, or rejects the instance. Hence, we can apply induction on d_0, and assume that the algorithm works correctly when called for an instance with a smaller value for d_0.

We show that any set X returned by the algorithm is a solution for I. First, this is clear if $X = Y_0$, as the algorithm explicitly checks whether $\gamma_i(Y_0) \le d$ holds for each $i \in [k]$; second, if X was returned by a recursive call on some instance I'_v, then by our induction hypothesis we know that X is a solution for $I'_v = (V, f_1, \dots, f_k, d, Y'_{0,v}, d_0 - 1)$. Hence, X satisfies (3); moreover, by $|X \triangle Y'_{0,v}| \le d_0 - 1$, it also satisfies $|X \triangle Y_0| \le d_0$, because $|Y_0 \triangle Y'_{0,v}| = 1$.

Let us now prove that if I admits a solution X, then the algorithm correctly returns a solution for I. Let $Y_i \in \mathcal{L}_i$ be a minimizer such that $|X \triangle Y_i| \le d$ where i is the index fixed for which $\gamma_i(Y_0) > d$.

▷ Claim 16 (⋆). Assuming that all guesses made by the algorithm are correct, in the iterative process of modifying T and Y it will always hold that

(i) $T \subseteq Y_i \triangle Y_0$, and

(ii) for each $S \in \mathcal{P}$:

  (a) if S is fixed, then $S \subseteq Y \iff S \subseteq Y_i$, and $S \cap Y = \emptyset \iff S \cap Y_i = \emptyset$, and

  (b) if $v \in S$ and S is not fixed, then $v \in Y \iff v \in Y_0$.

Next, we show that in each run of the iteration, Condition 1 or Condition 2 holds. Indeed, if neither holds, then (1) $Y = \bigcup_{U \in \mathcal{P}'} U$ for some $\mathcal{P}' \subseteq \mathcal{P}$, and (2) no edge leaves $\mathcal{P}'$ in $G(\mathcal{L}_i)$. Hence, $Y \in \mathcal{L}_i$ by Birkhoff's representation theorem. However, since $|T| \le d$ holds at the beginning of each iteration, $|Y \triangle Y_0| = |T| \le d$ follows, contradicting our choice of i.

Therefore, in each run of the iteration, at least one element of V is put into T. Thus, the iteration stops after at most $d + 1$ runs, at which point the obtained set T has size greater than d. Using now statement (i) of Claim 16, Observation 14 yields that T contains at least one vertex from $X \triangle Y_0$. Assuming that the algorithm guesses such a vertex v correctly, it is clear that our solution X for I will also be a solution for the instance I'_v. Using our inductive hypothesis, we obtain that the recursive call returns a correct solution for I'_v which, as discussed already, will be a solution for I as well. Hence, our algorithm is correct.

Finally, let us bound the running time. Consider the search tree $\mathcal{T}$ where each node corresponds to a call of Algorithm ASM. Note that the value of d_0 decreases by one in each recursive call, and the algorithm stops when $d_0 = 0$. Hence $\mathcal{T}$ has depth at most d_0. Consider the guesses made during the execution of a single call of the algorithm (without taking into account the guesses in the recursive calls): we make at most one guess in each iteration on line 9 or on line 19, leading to at most 2^{d+1} possibilities. Then the algorithm further guesses a vertex from T, leading to a total of at most $2^{d+1}|T| \le 2^{d+1}(d + d_0) = 2^{O(d)}$ possibilities; recall that $d_0 \le d$. We get that the number of nodes in our search tree is $2^{d_0 O(d)}$. Since all computations for a fixed series of guesses take polynomial time, we obtain that the running time is indeed fixed-parameter tractable with parameter d. ◀

4 Hardness Results

We first introduce a separation problem that we will use as an intermediary problem in our hardness proofs. Given a subset $X \subseteq V$ of some universe V that contains two distinguished elements, s and t, and a family Π of pairwise disjoint subsets of V, we define the *distance* of the set X from Π as $\sum_{S \in \Pi} \mathsf{dist}_{s,t}(X, S)$ where

$$\mathsf{dist}_{s,t}(X,S) = \begin{cases} \min\{|S \setminus X|, |S \cap X|\} & \text{if } s \notin S, t \notin S; \\ |S \setminus X| & \text{if } s \in S, t \notin S; \\ |S \cap X| & \text{if } s \notin S, t \in S; \\ +\infty & \text{if } s \in S, t \in S. \end{cases}$$

Given a collection of set families $\Pi_1, \dots, \Pi_k$, the goal is to find a set $X \subseteq V$ that separates s from t in the sense that $s \in X$ but $t \notin X$, and subject to this constraint, minimizes the maximum distance of X from the given set families. Formally, the problem is:

Robust Separation:

Input: A finite set V with two elements $s, t \in V$, set families $\Pi_1, \dots, \Pi_k$ where each Π_i is a collection of pairwise disjoint subsets of V, and an integer $d \in \mathbb{N}$.

Task: Find a set $X \subseteq V$ containing s but not t such that for each $i \in [k]$

$$\sum_{S \in \Pi_i} \mathsf{dist}_{s,t}(X, S) \leq d, \tag{4}$$

or output "No" if no such set X exists.

Given an instance $(V, s, t, \Pi_1 \dots, \Pi_k, d)$ of Robust Separation, the reduction proving Lemma 17 below constructs a graph G_i over V for each $i \in [k]$ in which each set in Π_i forms a clique, and defines a submodular function f_i based on the cut function of G_i.

▶ **Lemma 17** (⋆). *Robust Separation can be reduced to Robust Submodular Minimizer in polynomial time via a reduction that preserves the values of both k and d.*

4.1 NP-hardness for a constant $d \geq 1$

In this section, we prove that Robust Submodular Minimizer is NP-hard for each constant $d \geq 1$. To this end, we first prove the NP-hardness of Robust Separation in the case $d = 1$, and then extend this result to hold for any constant $d \geq 1$.

For the case $d = 1$, we present a reduction from the 1-in-3 SAT problem. In this problem, we are given a set V of variables and a set $\mathcal{C}$ of clauses, with each clause $C \in \mathcal{C}$ containing exactly three distinct literals; here, a *literal* is either a variable $v \in V$ or its negation $\overline{v}$. Given a truth assignment $\phi : V \to \{\texttt{true}, \texttt{false}\}$, we automatically extend it to the set $\overline{V} = \{\overline{v} : v \in V\}$ of negative literals by setting $\phi(\overline{v}) = \texttt{true}$ if and only if $\phi(v) = \texttt{false}$. We say that a truth assignment is *valid*, if it maps *exactly* one literal in each clause to $\texttt{true}$. The task in the 1-in-3 SAT problem is to decide whether a valid truth assignment exists. This problem is NP-complete [27].

▶ **Theorem 18** (⋆). *Robust Separation is* NP*-hard even when $d = 1$.*

Proof. Suppose that we are given an instance of the 1-in-3 SAT problem with variable set V and clause set $\mathcal{C} = \{C_1 \dots, C_m\}$. We construct an instance I_{RS} of Robust Separation as follows. In addition to the set V of variables and the set $\overline{V} = \{\overline{v} : v \in V\}$ of negative literals, we introduce our two distinguished elements, s and t. We further introduce a set $R_j = \{r_{j,1}, r_{j,2}, r_{j,3}\}$ together with an extra element z_j for each clause $C_j \in \mathcal{C}$ to form our universe U. We let $R = R_1 \cup \dots \cup R_m$ and $Z = \{z_1, \dots, z_m\}$, so that

$$U = V \cup \overline{V} \cup \{s, t\} \cup \bigcup_{j \in [m]} (R_j \cup \{z_j\}) = V \cup \overline{V} \cup \{s, t\} \cup R \cup Z.$$

Next, for each variable, we introduce two set families, Π_v and $\Pi_{\overline{v}}$, where

$$\Pi_v = \{\{s, v, \overline{v}\} \cup R\} \qquad \text{and} \qquad \Pi_{\overline{v}} = \{\{v, \overline{v}, t\}\}.$$

For simplicity, we write $\Pi(V) = \langle \Pi_v, \Pi_{\overline{v}} : v \in V \rangle$ to denote the $2|V|$-tuple formed by these set families. For each clause $C_j \in \mathcal{C}$, we fix an arbitrary ordering of its literals, and we denote the first, second, and third literals in C_j as $\ell_{j,1}, \ell_{j,2}$ and $\ell_{j,3}$. We define three set families:

$$\begin{aligned}
\Pi_{C_j} &= \{S_j\} && \text{where} \quad S_j \;\, = C_j \cup \{t\} = \{\ell_{j,1}, \ell_{j,2}, \ell_{j,3}, t\},\\
\Pi^{\alpha}_{C_j} &= \{S_j^{\alpha,1}, S_j^{\alpha,2}\} && \text{where} \quad S_j^{\alpha,1} = \{\ell_{j,1}, z_j\},\\
& && \phantom{\text{where}} \quad S_j^{\alpha,2} = \{\ell_{j,2}, r_{j,2}\};\\
\Pi^{\beta}_{C_j} &= \{S_j^{\beta,1}, S_j^{\beta,2}\} && \text{where} \quad S_j^{\beta,1} = \{r_{j,1}, z_j\},\\
& && \phantom{\text{where}} \quad S_j^{\beta,2} = \{\ell_{j,3}, r_{j,3}\}.
\end{aligned}$$

We also write $\Pi(\mathcal{C}) = \langle \Pi_C, \Pi^{\alpha}_C, \Pi^{\beta}_C : C \in \mathcal{C}\rangle$ to denote the $3|\mathcal{C}|$-tuple formed by these set families in an arbitrarily fixed ordering. We set our threshold as $d = 1$. Thus, our instance of Robust Separation is $I_{\mathrm{RS}} = (U, s, t, \Pi(V), \Pi(\mathcal{C}), 1)$.

We will show that the constructed instance I_{RS} has a solution if and only if our instance $(V, \mathcal{C})$ of the 1-in-3 SAT problem is solvable.

First suppose that there is a valid truth assignment ϕ for $(V, \mathcal{C})$. Consider the set

$$X = \{s\} \cup R \cup \{\ell : \ell \in V \cup \overline{V}, \phi(\ell) = \texttt{true}\} \cup \{z_j : z_j \in Z, \phi(\ell_{j,3}) = \texttt{false}\}.$$

Note that X contains s, but not t; we are going to show that it is a solution for I_{RS}. Since ϕ maps exactly one literal in $\{v, \overline{v}\}$ to $\texttt{true}$ for each $v \in V$, by $R \cup \{s\} \subseteq X$ we get that

$$\sum_{S \in \Pi_v} \mathsf{dist}_{s,t}(X, S) = |(\{s, v, \overline{v}\} \cup R) \setminus X| = |\{v, \overline{v}\} \setminus X| = 1 \qquad \text{and}$$
$$\sum_{S \in \Pi_{\overline{v}}} \mathsf{dist}_{s,t}(X, S) = |(\{v, \overline{v}, t\}) \cap X| = |\{v, \overline{v}\} \cap X| = 1.$$

For the distance of X from the set families associated with some clause $C_j \in \mathcal{C}$, by the validity of ϕ we obtain

$$\sum_{S \in \Pi_{C_j}} \mathsf{dist}_{s,t}(X, S) = |(C_j \cup \{t\}) \cap X| = 1;$$

$$\begin{aligned}
\sum_{S \in \Pi^{\alpha}_{C_j}} \mathsf{dist}_{s,t}(X, S) &= \min\{|S_j^{\alpha,1} \setminus X|, |S_j^{\alpha,1} \cap X|\} + \min\{|S_j^{\alpha,2} \setminus X|, |S_j^{\alpha,2} \cap X|\}\\
&= \min\{|\{\ell_{j,1}, z_j\} \setminus X|, |\{\ell_{j,1}, z_j\} \cap X|\}\\
&\qquad\qquad + \min\{|\{\ell_{j,2}, r_{j,2}\} \setminus X|, |\{\ell_{j,2}, r_{j,2}\} \cap X|\}\\
&= \left\{ \begin{array}{ll} \min\{0,2\} + \min\{1,1\} = 1 & \text{if } \phi(\ell_{j,1}) = \texttt{true}\\ \min\{1,1\} + \min\{0,2\} = 1 & \text{if } \phi(\ell_{j,2}) = \texttt{true}\\ \min\{2,0\} + \min\{1,1\} = 1 & \text{if } \phi(\ell_{j,3}) = \texttt{true} \end{array} \right\} = 1;
\end{aligned}$$

$$\begin{aligned}
\sum_{S \in \Pi^{\beta}_{C_j}} \mathsf{dist}_{s,t}(X, S) &= \min\{|S_j^{\beta,1} \setminus X|, |S_j^{\beta,1} \cap X|\} + \min\{|S_j^{\beta,2} \setminus X|, |S_j^{\beta,2} \cap X|\}\\
&= \min\{|\{r_{j,1}, z_j\} \setminus X|, |\{r_{j,1}, z_j\} \cap X|\}\\
&\qquad\qquad + \min\{|\{\ell_{j,3}, r_{j,3}\} \setminus X|, |\{\ell_{j,3}, r_{j,3}\} \cap X|\}\\
&= \left\{ \begin{array}{ll} \min\{0,2\} + \min\{1,1\} = 1 & \text{if } \phi(\ell_{j,1}) = \texttt{true}\\ \min\{0,2\} + \min\{1,1\} = 1 & \text{if } \phi(\ell_{j,2}) = \texttt{true}\\ \min\{1,1\} + \min\{0,2\} = 1 & \text{if } \phi(\ell_{j,3}) = \texttt{true} \end{array} \right\} = 1.
\end{aligned}$$

Hence, X satisfies constraint (4) for each set family, and thus is a solution for I_{RS}.

We prove the other direction of the claim in the full version of our paper [17]. ◀

Using Theorem 18, it is not hard to show that Robust Separation remains NP-hard for any constant $d \geq 1$.

▶ **Lemma 19** ($\star$). *Robust Separation is* NP*-hard for each constant* $d \geq 1$.

▶ **Corollary 20.** *Robust Submodular Minimizer is* NP*-hard for each constant* $d \geq 1$.

4.2 NP-hardness for a constant $k \geq 3$

In this section we prove that Robust Separation, and hence, Robust Submodular Minimizer is NP-hard even for $k = 3$. To this end, we are going to define another intermediary problem. First consider the Most Balanced Minimum Cut problem, proved to be NP-complete by Bonsma [3]. The input of this problem is an undirected graph $G = (V, E)$ with two distinguished vertices, s and t, and a parameter ℓ. The task is to decide whether there exists a minimum (s, t)-cut $X \subseteq V$ in G such that $\min\{|X|, |V \setminus X|\} \geq \ell$; recall that a set of vertices $X \subseteq V$ is a minimum (s, t)-cut in the *undirected* graph G if $s \in X, t \notin X$ and subject to this, the value $|\delta(X)|$, i.e., the number of edges between X and $V \setminus X$, is minimized.

Instead of the Most Balanced Minimum Cut problem, it will be more convenient to use a variant that we call Perfectly Balanced Minimum Cut where we seek a minimum (s, t)-cut that contains exactly half of the vertices. Formally, its input is an undirected graph $G = (V, E)$ with two distinguished vertices, s and t, and its task is to find a minimum (s, t)-cut X with $|X| = |V|/2$. Since Most Balanced Minimum Cut can be reduced to Perfectly Balanced Minimum Cut by simply adding a sufficient number of isolated vertices, we obtain the following.

▶ **Lemma 21** ($\star$). *Perfectly Balanced Minimum Cut is* NP*-complete.*

▶ **Theorem 22** ($\star$). *Robust Separation is NP-hard even when* $k = 3$.

Proof. We present a reduction from the Perfectly Balanced Minimum Cut problem. Let $I = (G, s, t)$ be our input instance where $G = (V, E)$. Clearly, we may assume that $|V|$ is even, as otherwise I is trivially a "no"-instance. First we compute the number of edges in a minimum (s, t)-cut using standard flow techniques; let δ^* denote this value, that is, $\delta^* = \min_{Y: s \in Y \subseteq V \setminus \{t\}} |\delta(Y)|$.

Second, we modify G in order to ensure that there are at least $2\delta^* + 2$ vertices in the graph; if this holds already for G, then we set $G' = G$. Otherwise, we construct a new graph $G' = (V', E')$ by adding two sets of vertices, A_s and A_t, to the graph with $|A_s| = |A_t| = \lceil (2\delta^* + 2 - |V|)/2 \rceil$, and connecting each vertex in A_s to s, as well as each vertex in A_t to t, with an edge. Observe that all minimum (s, t)-cuts in G' contain A_s and are disjoint from A_t. Moreover, any minimum (s, t)-cut X in G corresponds to a minimum (s, t)-cut $X \cup A_S$ in G' and vice versa. Thus, $I' = (G', s, t)$ is an instance of Perfectly Balanced Minimum Cut equivalent with I. Let $2n + 2$ denote the number of vertices in G', so that $\tilde{V} := V' \setminus \{s, t\}$ has $2n$ vertices. By our choice of $|A_s| = |A_t|$, we know that the number of vertices in G' is $|V'| = 2n + 2 \geq |V| + (2\delta^* + 2 - |V|) = 2\delta^* + 2$, as promised.

Let us construct an instance J of Robust Separation. We define our universe U as follows. For each $v \in V'$ we introduce a set $P(v) = \{\hat{v}\} \cup \{v^u : uv \in E\}$, and we additionally define a copy $V^* = \{v^* : v \in V\}$ of V, a set R of size $|R| = n - \delta^*$, and a copy $R' = \{r' : r \in R\}$ of R. Thus, we have

$$U = \bigcup_{v \in V'} P(v) \cup V^* \cup R \cup R'.$$

We set s^* and t^*, both in V^*, as our two distinguished vertices.

We define our three families for J as follows:

$$\begin{array}{ll}
\Pi_1 = \{S_1\} & \text{where } S_1 = V^* \setminus \{t^*\} \cup R \cup P(s); \\
\Pi_2 = \{S_2\} \cup \{S_2^v : v \in \tilde{V}\} & \text{where } S_2 = V^* \setminus \{s^*\} \cup R' \cup P(t), \\
 & \qquad S_2^v = P(v) \quad \forall v \in \tilde{V}; \\
\Pi_3 = \{S_3^v : v \in \tilde{V}\} \cup \{S_3^e : e \in E'\} \cup \{S_3^r : r \in R\} & \text{where } S_3^v = \{\hat{v}, v^*\} \quad \forall v \in \tilde{V}, \\
 & \qquad S_3^e = \{u^v, v^u\} \ \forall e = uv \in E', \\
 & \qquad S_3^r = \{r, r'\} \quad \forall r \in R.
\end{array}$$

Thus, Π_1 contains only a single set, Π_2 contains $|\tilde{V}|+1$ pairwise disjoint sets, and Π_3 contains $|\tilde{V}| + |E'| + |R|$ pairwise disjoint sets. We finish the definition of our instance J by setting $d = n$ as our threshold, so that $J = (U, s^*, t^*, \Pi_1, \Pi_2, \Pi_3, n)$.

We claim that G' admits a minimum (s,t)-cut containing exactly $n+1$ vertices if and only if J is a "yes"-instance of Robust Separation. The proof of this claim can be found in the full version of our paper [17]. ◀

Clearly, we can increase the value of parameter k without changing the solution set of our instance of Robust Separation by repeatedly adding a copy of, say, the first set family Π_1. Using also Lemma 17, we have the following easy consequences of Theorem 22:

▶ **Corollary 23.** *Robust Separation is NP-hard for each constant $k \geq 3$.*

▶ **Corollary 24.** *Robust Submodular Minimizer is NP-hard for each constant $k \geq 3$.*

5 Conclusion

In this paper, we studied the computational complexity of Robust Submodular Minimizer, and provided a complete computational map of the problem with respect to the parameters k and d, offering dichotomies for the case when one of these parameters is a constant, and giving an FPT algorithm for the combined parameter (k, d). Regarding the case when one of the functions f_i has only polynomially bounded minimizers, there are a few questions left open: First, what is the computational complexity of this variant when parameterized by k? Second, is there an algorithm for this case with running time $2^{O(d)}|I|^{O(1)}$ on some instance I instead of the running time $2^{O(d^2)}|I|^{O(1)}$ we obtained based on the algorithm for Theorem 13?

We remark that our algorithmic results can be adapted in a straightforward way to a slightly generalized problem: given k submodular functions $f_1, \ldots, f_k$ with non-negative integers $d_1, \ldots, d_k$, we aim to find a set X such that, for each $i \in [k]$, there exists some set $Y_i \in \arg\min f_i$ with $|X \triangle Y_i| \leq d_i$ for each $i \in [k]$. As mentioned in Section 1.2, Robust Submodular Minimizer is related to recoverable robustness. We can consider the robust recoverable variant of submodular minimization: given submodular functions $f_0, f_1, \ldots, f_k$, we aim to find a set X that minimizes

$$f_0(X) + \max_{i \in [k]} \min_{Y_i : |Y_i \triangle X| \leq d} f_i(X_i).$$

The optimal value is lower-bounded by $f_0(Y_0) + \max_{i \in [k]} f_i(Y_i)$ where $Y_i \in \arg\min f_i$ for each $i \in \{0, 1, \ldots, k\}$. Our results imply that we can decide efficiently whether the optimal value attains this lower bound or not, when d and k are parameters, or when f_0 has polynomially many minimizers.

References

1 Garrett Birkhoff. Rings of sets. *Duke Math. J.*, 3(3):443–454, 1937. doi:10.1215/S0012-7094-37-00334-X.

2 Hans-Joachim Böckenhauer, Karin Freiermuth, Juraj Hromkovic, Tobias Mömke, Andreas Sprock, and Björn Steffen. Steiner tree reoptimization in graphs with sharpened triangle inequality. *J. Discrete Algorithms*, 11:73–86, 2012. doi:10.1016/J.JDA.2011.03.014.

3 Paul Bonsma. Most balanced minimum cuts. *Discrete Applied Mathematics*, 158:261–276, 2010. doi:10.1016/j.dam.2009.09.010.

4 Nicolas Boria and Vangelis Th. Paschos. Fast reoptimization for the minimum spanning tree problem. *J. Discrete Algorithms*, 8(3):296–310, 2010. doi:10.1016/J.JDA.2009.07.002.

5 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012. doi:10.1002/NET.20487.

6 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear-time algorithms for maximum flow and minimum-cost flow. *Commun. ACM*, 66(12):85–92, 2023. doi:10.1145/3610940.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, Cham, 2015. doi:10.1007/978-3-319-21275-3.

8 Mitre Costa Dourado, Dirk Meierling, Lucia Draque Penso, Dieter Rautenbach, Fábio Protti, and Aline Ribeiro de Almeida. Robust recoverable perfect matchings. *Networks*, 66(3):210–213, 2015. doi:10.1002/NET.21624.

9 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, London, 2013. doi:10.1007/978-1-4471-5559-1.

10 Dennis Fischer, Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. An investigation of the recoverable robust assignment problem. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.IPEC.2021.19.

11 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/s002240000044.

12 Marc Goerigk, Stefan Lendl, and Lasse Wulf. On the recoverable traveling salesman problem. *CoRR*, abs/2111.09691, 2021. arXiv:2111.09691.

13 Felix Hommelsheim, Nicole Megow, Komal Muluk, and Britta Peis. Recoverable robust optimization with commitment. *CoRR*, abs/2306.08546, 2023. arXiv:2306.08546.

14 Mikita Hradovich, Adam Kasperski, and Pawel Zielinski. Recoverable robust spanning tree problem under interval uncertainty representations. *J. Comb. Optim.*, 34(2):554–573, 2017. doi:10.1007/S10878-016-0089-6.

15 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. A parameterized view to the robust recoverable base problem of matroids under structural uncertainty. *Oper. Res. Lett.*, 50(3):370–375, 2022. doi:10.1016/J.ORL.2022.05.001.

16 Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Submodular reassignment problem for reallocating agents to tasks with synergy effects. *Discret. Optim.*, 44(Part):100631, 2022. doi:10.1016/j.disopt.2021.100631.

17 Naonori Kakimura and Ildikó Schlotter. Parameterized complexity of submodular minimization under uncertainty. *CoRR*, abs/2404.07516, 2024. arXiv:2404.07516.

18 Stefan Kratsch, Shaohua Li, Dániel Marx, Marcin Pilipczuk, and Magnus Wahlström. Multi-budgeted directed cuts. *Algorithmica*, 82(8):2135–2155, 2020. doi:10.1007/S00453-019-00609-1.

19 Thomas Lachmann, Stefan Lendl, and Gerhard J. Woeginger. A linear time algorithm for the robust recoverable selection problem. *Discret. Appl. Math.*, 303:94–107, 2021. doi:10.1016/J.DAM.2020.08.012.

20 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17–20 October, 2015*, pages 1049–1065, 2015. doi:10.1109/FOCS.2015.68.

21 Stefan Lendl, Britta Peis, and Veerle Timmermans. Matroid bases with cardinality constraints on the intersection. *Math. Program.*, 194(1):661–684, 2022. doi:10.1007/S10107-021-01642-1.

22 Christian Liebchen, Marco E. Lübbecke, Rolf H. Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, volume 5868 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2009. doi:10.1007/978-3-642-05465-5_1.

23 Jérôme Monnot. A note on the traveling salesman reoptimization problem under vertex insertion. *Inf. Process. Lett.*, 115(3):435–438, 2015. doi:10.1016/J.IPL.2014.11.003.

24 Kazuo Murota. *Discrete Convex Analysis*. SIAM, 2003. doi:10.1137/1.9780898718508.

25 James B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Math. Program.*, 118(2):237–251, 2009. doi:10.1007/s10107-007-0189-2.

26 Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. *Mathematical Programming Studies*, 13:8–16, 1980. doi:10.1007/BFb0120902.

27 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth ACM Symposium on Theory of Computing (STOC '78)*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

28 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

Optimal In-Place Compaction of Sliding Cubes

Irina Kostitsyna
TU Eindhoven, The Netherlands

Tim Ophelders
Utrecht University, The Netherlands
TU Eindhoven, The Netherlands

Irene Parada
Universitat Politècnica de Catalunya, Barcelona, Spain

Tom Peters
TU Eindhoven, The Netherlands

Willem Sonke
TU Eindhoven, The Netherlands

Bettina Speckmann
TU Eindhoven, The Netherlands

Abstract

The sliding cubes model is a well-established theoretical framework that supports the analysis of reconfiguration algorithms for modular robots consisting of face-connected cubes. As is common in the literature, we focus on reconfiguration via an intermediate canonical shape. Specifically, we present an in-place algorithm that reconfigures any n-cube configuration into a compact canonical shape using a number of moves proportional to the sum of coordinates of the input cubes. This result is asymptotically optimal and strictly improves on all prior work. Furthermore, our algorithm directly extends to dimensions higher than three.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Sliding cubes, Reconfiguration algorithm, Modular robots

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.31

Related Version *Full Version*: `https://arxiv.org/abs/2312.15096`

Supplementary Material *Audiovisual*: `https://www.computational-geometry.org/SoCG-videos/socg24video/optimal-in-place-compaction-of-sliding-cubes.mp4`

Funding *Tim Ophelders*: partially supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.

1 Introduction

Modular robots consist of a large number of comparatively simple robotic units. These units can attach and detach to and from each other, move relative to each other, and in this way form different shapes or configurations. This shape-shifting ability allows modular robots to robustly adapt to previously unknown environments and tasks. One of the major questions regarding modular robots is *universal reconfiguration*: is there a sequence of moves which transforms any two given configurations into each other, and if so, how many moves are necessary? There are a variety of real-world mechatronics or theoretical computational models for modular robots and the answer to the universal reconfiguration question differs substantially between systems [3].

In this paper, we study the *sliding cube model*, which is a well-established theoretical framework that supports the analysis of reconfiguration algorithms for modular robots consisting of face-connected cubes. In this model, a module (cube) can perform two types of moves: straight-line moves called *slides* and moves around a corner called *convex transitions*

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 31; pp. 31:1–31:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

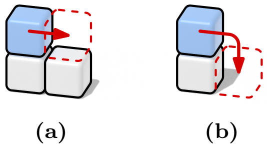


Figure 1 Moves in the sliding cube model: slide (a) and convex transition (b). Solid cubes are part of the configuration.

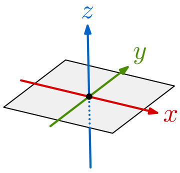
(see Figure 1). Maintaining connectivity during a sequence of moves is the main challenge when developing algorithms in the sliding cube model. During a move, the configuration (excluding the moving cube) must stay connected. Furthermore, there have to be sufficient empty cells to perform the move. This connectivity is crucial for most actual modular robotic systems since it allows them to retain their structure, communicate, and share other resources such as energy.

Almost 20 years ago, Dumitrescu and Pach [8] showed that the sliding cube model in 2D (or *sliding square model*) is universally reconfigurable. More precisely, they presented an algorithm that transforms any two given configurations with n squares into each other in $O(n^2)$ moves. This algorithm transforms any given configuration into a canonical shape (a horizontal line) and then reverts the procedure to reach the final configuration. It was afterwards adapted to be *in-place* using flooded bounding boxes as canonical intermediate configurations [15]. Recently, Akitaya et al. [4] presented Gather&Compact: an input-sensitive in-place algorithm which uses $O(Pn)$ moves, where P is the maximum among the perimeters of the bounding boxes of the initial and final configurations. The authors also show that minimizing the number of moves required to reconfigure is NP-hard.

These algorithms in 2D do not directly transfer to 3D: they fundamentally rely on the fact that a connected cycle of squares encloses a well-defined part of the configuration. One could ask whether the fact that enclosing space in 3D is more difficult has positive or negative impact on universal reconfiguration in 3D. Miltzow et al. [14] showed that there exist 3D configurations in which no module on the external boundary is able to move without disconnecting the configuration. Hence, simple reconfiguration strategies [11, 13] can generally not guarantee reconfiguration for all instances.

Until very recently, the most efficient algorithm for the reconfiguration problem in 3D was the algorithm by Abel and Kominers [1], which uses $O(n^3)$ moves to transform any n-cube configuration into any other n-cube configuration. As is common in the literature, this algorithm reconfigures the input into an intermediate canonical shape. Stock et al. [17] just announced a worst-case bound of $O(n^2)$ moves for the Abel and Kominers algorithm. Furthermore, their paper presents an in-place reconfiguration algorithm, which runs in time proportional to a measure of the size of the bounding box times the number of cubes. Specifically, their algorithm requires $O(n(wd + h))$ moves in the worst-case, where w, d, and h are the width, depth, and height of the bounding box, respectively.

Results. In this paper we present an in-place algorithm that reconfigures any n-cube configuration into a compact canonical shape using a number of moves proportional to the sum of coordinates of the input cubes. This result is asymptotically optimal and strictly stronger than the bounds obtained by Stock et al. [17]. Furthermore, our algorithm directly extends to hypercube reconfiguration in dimensions higher than three. Last but not least, the restriction of our algorithm to two dimensions improves upon the best bound for compacting sliding squares [4].



Additional related work. For more restricted sliding models, for example, only allowing one of the two moves in the sliding cube model, reconfiguration is not always possible. Michail et al. [13] explore universal reconfiguration using *helpers* or *seeds* (dedicated cubes that help other cubes move). They show that the problem of deciding how many seeds are needed is in PSPACE.

Another popular model for modular robots is the pivoting cube model, in which the modules move by rotating around an edge shared with a neighboring module. In this model the extra free-space requirements for the moves that come from pivoting instead of sliding mean that there are configurations in which no move is possible. Akitaya et al. [3] show that the reconfiguration problem in this setting is PSPACE-complete. In contrast, adding five additional modules to the outer boundary guarantees universal reconfigurability in 2D using $O(n^2)$ moves [2]. Other algorithms for pivoting modules require the absence of narrow corridors in both the initial and final configurations [10, 18]. A more powerful move is to allow the modules to *tunnel* through the configuration. With it, $O(n)$ parallel steps suffice to reconfigure 2D and 3D cubes [5, 6, 12]. However, for most real-world prototype systems, tunnelling requires the use of *metamodules* [16] which are sets of modules which act as a single unit with enhanced capabilities, increasing the granularity of the configurations.

We require that the configuration stays connected at all times. In a slightly different model that relaxes the connectivity requirement (referred to as the *backbone property*), Fekete et al. [9] show that scaled configurations of labeled squares can be efficiently reconfigured using parallel coordinated moves with a schedule that is a constant factor away from optimal.

2 Preliminaries

In this paper, we study cubical modules moving in the 3-dimensional grid $\mathbb{Z}^3$. The handedness of the coordinate system does not have any impact on the correctness of our algorithm.

A *configuration* $\mathcal{C}$ is a subset of coordinates in the grid. The elements of $\mathcal{C}$ are called *cubes*. We call two cubes *adjacent* if they lie at unit distance. For a configuration $\mathcal{C}$, denote by $G_{\mathcal{C}}$ the graph with vertex set $\mathcal{C}$, whose edges connect all adjacent cubes. We say a *cell* is a vertex of $G_{\mathbb{Z}^3}$ which is not occupied by a cube in $\mathcal{C}$. We always require a configuration to remain *connected*, that is, $G_{\mathcal{C}}$ must be connected. For ease of exposition we assume $\mathcal{C}$ consists of at least two cubes. Let $B_{\mathcal{C}}$ be the bounding box of a configuration $\mathcal{C}$. W.l.o.g. we assume that the vertex in $B_{\mathcal{C}}$ with minimum x-, y-, and z-coordinate is the origin of $G_{\mathbb{Z}^3}$.

In the sliding cubes model, a configuration can rearrange itself by letting cubes perform moves. A move replaces a single cube $c \in \mathcal{C}$ by another cube $c' \notin \mathcal{C}$. Moves come in two types: *slides* and *convex transitions* (see Figure 1). In both cases, we consider a 4-cycle γ in $G_{\mathbb{Z}^3}$. For slides, exactly three cubes of γ are in $\mathcal{C}$; c' is the cell of γ not in $\mathcal{C}$, and c is adjacent to c'. For convex transitions, γ has exactly two adjacent cubes in $\mathcal{C}$; c is one of these two cubes, and c' is the vertex of γ not adjacent to c. The slide or convex transition is a *move* if and only if $\mathcal{C} \setminus \{c\}$ is connected.

Call a cube $c = (x, y, z)$ *finished* if the cuboid spanned by the origin and c is completely in $\mathcal{C}$, that is, if $\{0, \ldots, x\} \times \{0, \ldots, y\} \times \{0, \ldots, z\} \subseteq \mathcal{C}$. We call $\mathcal{C}$ *finished* if all cubes in $\mathcal{C}$ are finished. The *compaction problem* starts with an arbitrary connected configuration $\mathcal{C}$ with bounding box $B_{\mathcal{C}}$ and is solved when all cubes are finished. An algorithm for this problem is *in-place* if at most a single cube simultaneously moves through cells face-adjacent to $B_{\mathcal{C}}$.

Most of the algorithm works on vertical contiguous strips of cubes in $\mathcal{C}$ called subpillars. More precisely, a *subpillar* of $\mathcal{C}$ is a subset of $\mathcal{C}$ of the form $\{x\} \times \{y\} \times \{z_b, \ldots, z_t\}$. In the remainder of this paper, we denote this subpillar by $\langle x, y, z_b \mathinner{..} z_t \rangle$. The cube (x, y, z_t) is called the *head*, and the remainder $\langle x, y, z_b \mathinner{..} z_t - 1 \rangle$ is called the *support* of the subpillar. A *pillar* is a maximal subpillar, that is, a subpillar that is not contained in any other subpillar. Note that there can be multiple pillars above each other with the same x- and y-coordinates, as long as there is a gap between them. Two sets S and S' of cubes are *adjacent* if S contains a cube adjacent to a cube in S'. The *pillar graph* $\mathcal{P}_S$ of a set S of cubes is the graph whose vertices are the pillars of S and whose edges connect adjacent pillars.

3 Algorithm

For a set of cubes $S \subseteq \mathcal{C}$, let its *coordinate vector sum* be $(X_S, Y_S, Z_S) = \sum_{(x,y,z) \in S} (x, y, z)$. Let $\mathcal{C}_{>0}$ be the subset of cubes $(x, y, z) \in \mathcal{C}$ for which $z > 0$, and $\mathcal{C}_0$ be the subset of cubes for which $z = 0$. Let the *potential* of a cube $c = (c_x, c_y, c_z)$ be $\Pi_c = w_c(c_x + 2c_y + 4c_z)$, where the *weight* w_c depends on the coordinates of c in the following way. If $c_z > 1$, then $w_c = 5$; if $c_z = 1$, then $w_c = 4$. If $c_z = 0$, then w_c depends on c_y. If $c_y > 1$, then $w_c = 3$; if $c_y = 1$, then $w_c = 2$ and lastly, if $c_z = c_y = 0$, then $w_c = 1$. We aim to minimize the *potential function* $\Pi_{\mathcal{C}} = \sum_{c \in \mathcal{C}} \Pi_c$. From now on, let $\mathcal{C}$ be an unfinished configuration. We call a sequence of m moves applied to $\mathcal{C}$ *safe* if the result is a configuration $\mathcal{C}'$, such that $\Pi_{\mathcal{C}'} < \Pi_{\mathcal{C}}$ and $m = O(\Pi_{\mathcal{C}} - \Pi_{\mathcal{C}'})$. This means that the sequence of moves reduces the potential by at least some constant fraction of m by going from $\mathcal{C}$ to $\mathcal{C}'$. We show that if $\mathcal{C}$ is unfinished, it always admits a safe move sequence.

The main idea is as follows. For a configuration $\mathcal{C}$, whenever possible, we try to reduce $Z_{\mathcal{C}}$ by some sequence of moves. If that is not possible, then the configuration must admit another sequence of moves, where a complete pillar is moved to a different x- or y-coordinate. In this way, by reducing either the z-coordinate of cubes, or the x- or y-coordinate, we guarantee that eventually every cube becomes finished.

In this paper, we describe the algorithm in three dimensions. However, it naturally extends to higher dimensions, and also works for squares in two dimensions. In fact, we will use the algorithm in two dimensions as a subroutine for the three-dimensional case.

Local z reduction. Let $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ be a subpillar of $\mathcal{C}$. We refer to the four coordinates $\{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\}$ as the *sides* of P. On each side, P has zero or more adjacent pillars. We order these by their z-coordinates; as such, we may refer to the top- or bottommost adjacent pillar on a side of P. We say that a set of cubes $S \subseteq \mathcal{C}$ is *non-cut* if $G_{\mathcal{C} \setminus S}$ is connected or empty. A pillar of $\mathcal{C}$ is non-cut if and only if it is a non-cut vertex of the pillar graph $\mathcal{P}_{\mathcal{C}}$.

Let $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ be a non-cut subpillar, and let $P' = \langle x', y', z'_b \mathinner{..} z'_t \rangle$ be a pillar adjacent to P. We define a set of operations of at most three moves within P which locally reduce $Z_{\mathcal{C}}$ (see Figure 2). Because P is non-cut, $\mathcal{C} \setminus P$ is connected. Therefore, if cubes of P move in such a way that each component (of cubes originally in P) remains adjacent to a cube of $\mathcal{C} \setminus P$, then the result of that operation is a valid configuration.

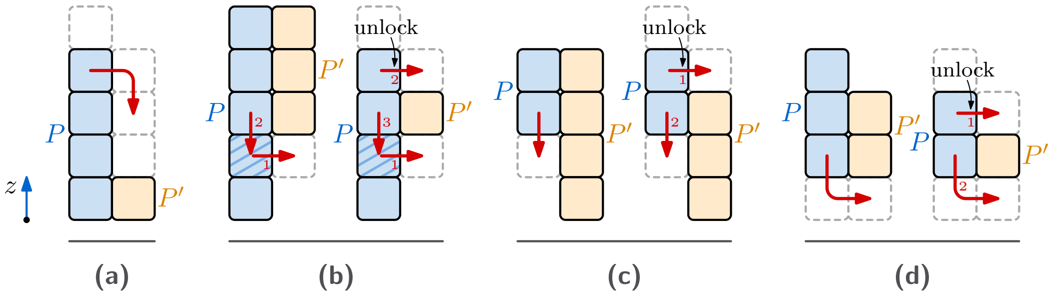


Figure 2 Examples of operations **(a–d)**; hatched cubes are non-cut and dashed outlines indicate cells that must be empty. Each case admits a move sequence that reduces $Z_{\mathcal{C}}$.

(a) If P' is a topmost adjacent pillar of P and $z'_t \leq z_t - 2$, then the topmost cube of P admits a convex transition that decreases $Z_{\mathcal{C}}$.

(b) If $z'_b > z_b$ and $(x, y, z'_b - 1)$ is a non-cut cube, then there is a move sequence that decreases $Z_{\mathcal{C}}$: first slide $(x, y, z'_b - 1)$ to $(x', y', z'_b - 1)$ and then slide (x, y, z'_b) to $(x, y, z'_b - 1)$. There is one special case. We say that P is *locked* if the head of P has no adjacent cubes except for P's support. If P is locked and $z'_b = z_t - 1$, then the second slide would disconnect P's head from the rest of the configuration. To avoid this, before performing the second slide, we *unlock* P by sliding the head of P from (x, y, z_t) to (x', y', z_t), as shown in the right part of Figure 2b.

(c) If $(x, y, z_b - 1) \notin \mathcal{C}$ and $z'_b < z_b$, then (after unlocking P, if necessary) (x, y, z_b) admits a slide to $(x, y, z_b - 1)$ that decreases $Z_{\mathcal{C}}$.

(d) If $(x, y, z_b - 1) \notin \mathcal{C}$, P' is a bottommost adjacent pillar of P, and $z_b = z'_b > 0$, then (after unlocking P, if necessary) (x, y, z_b) admits a convex transition to $(x', y', z'_b - 1)$ that decreases $Z_{\mathcal{C}}$.

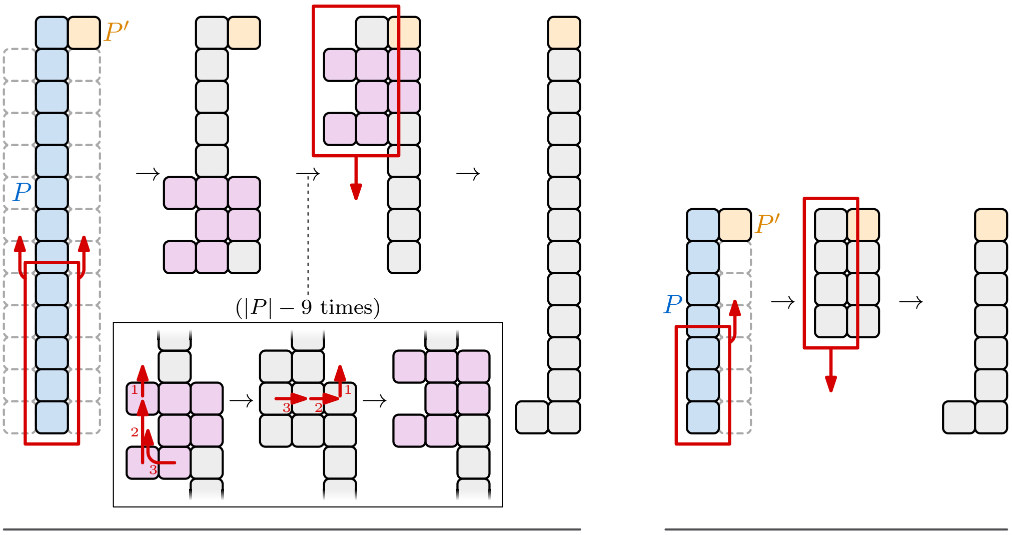
▶ **Lemma 1.** *Let $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ be a non-cut pillar. Assume P does not admit any operation of type* **(a–d)**. *Then, on each side, P has at most one adjacent pillar $P' = \langle x', y', z'_b \mathinner{..} z'_t \rangle$. For these pillars P', we have $z_t \leq z'_t + 1$, and either $z_b < z'_b$ or $z_b = z'_b = 0$.*

Proof. Consider one side s of P. Because **(a)** does not apply to P, for any adjacent pillar $P' = \langle x', y', z'_b \mathinner{..} z'_t \rangle$, we know that $z_t \leq z'_t + 1$. Consider the case that $(x, y, z'_b - 1)$ is a non-cut cube. Because **(b)** does not apply, $z_b \geq z'_b$, and because **(c)** does not apply either, $z_b = z'_b$, and finally because **(d)** does not apply, we have $z_b = z'_b = 0$. Now consider the case that $(x, y, z'_b - 1)$ is a cut cube. Then because **(c)** does not apply, we have $z_b \leq z'_b$, and finally because **(d)** does not apply, we have $z_b < z'_b$ or $z_b = z'_b = 0$. If $z_b = z'_b = 0$ for each adjacent pillar P', then each side of P can have at most one pillar. If $z_b < z'_b$, then, because **(b)** does not apply, $(x, y, z'_b - 1)$ is a cut cube. Therefore, also in this case, there can be no cube adjacent to the subpillar $\langle x, y, z_b \mathinner{..} z'_b - 2 \rangle$, and therefore also in this case there can be at most one adjacent pillar on each side. ◀

Pillar shoves. Next, we consider longer move sequences that still involve a single subpillar. A central operation of our algorithm is a *pillar shove*, which takes as parameters a subpillar $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ and a side (x', y') of P. The result of the pillar shove is the set of cubes

$$\operatorname{shove}(\mathcal{C}, P, (x', y')) \;:=\; (\mathcal{C} \setminus P) \;\cup\; \langle x', y', z_b \mathinner{..} z_t - 1 \rangle \;\cup\; \{(x, y, z_b)\},$$

in which the support is effectively shifted to the side (x', y'), and the head is effectively moved from (x, y, z_t) to (x, y, z_b). Although $\operatorname{shove}(\mathcal{C}, \langle x, y, z_b \mathinner{..} z_t \rangle, (x', y'))$ is well-defined, it is not necessarily a connected configuration, let alone safely reachable from $\mathcal{C}$.



(e) **(e′)**

Figure 3 Examples of pillar shoves for a long pillar **(e)** and a short pillar **(e′)**. The zipper operation on the left is executed $|P| - 9$ times.

Let $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ be a non-cut subpillar, and assume that on at least two sides (x', y') and (x'', y'') of P, no cube except possibly the head (x, y, z_t) has an adjacent cube. Moreover, assume that $(x', y', z_t) \in \mathcal{C}$. We define two types of pillar shoves, each of which transforms $\mathcal{C}$ into shove$(\mathcal{C}, \langle x, y, z_b \mathinner{..} z_t \rangle, (x', y'))$: a *long pillar shove* (for $|P| \geq 9$; see Figure 3**(e)**) and a *short pillar shove* (for $|P| < 9$; see Figure 3**(e')**). Note that the short pillar shove could be applied to the $|P| \geq 9$ case as well. However, a short pillar shove takes a number of moves quadratic in $|P|$ and hence would not be safe. A long pillar shove, on the other hand, takes a number of moves linear in $|P|$ as for each cube, we take a constant number of moves to move it to its new location (the "zipper" operation shown in the framed part of Figure 3**(e)**). As such, both pillar shoves reduce $Z_{\mathcal{C}}$ by $z_t - z_b$ and take $O(z_t - z_b)$ moves, so they are safe.

(e) Assume that no operations of type **(a–d)** are possible. Then, by Lemma 1, $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ has at most one adjacent pillar on each side, and there exists an adjacent pillar $P' = \langle x', y', z'_b \mathinner{..} z'_t \rangle$ with $z'_b > z_b$ (assume that P' is such a pillar with the lowest z'_b), and there is a side $(x'', y'') \neq (x', y')$ of P such that $(x'', y'', z_b) \notin \mathcal{C}$. Together, this implies that both sides (x'', y'') and (x', y') are empty up to at least $z'_b - 1$ (otherwise P' would not be the pillar with lowest $z'_b > z_b$). Then the subpillar $\langle x, y, z_b \mathinner{..} z'_b \rangle$ (after unlocking P, if necessary) admits a pillar shove.

▶ **Lemma 2.** *Let $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ be a non-cut subpillar. Assume P does not admit any operation of type* **(a–e)**. *Then P has no adjacent pillar $\langle x', y', z'_b \mathinner{..} z'_t \rangle$ with $z'_b > z_b$.*

Proof. Assume that P has at least two adjacent pillars, say $P' = \langle x', y', z'_b \mathinner{..} z'_t \rangle$ and $P'' = \langle x'', y'', z''_b \mathinner{..} z''_t \rangle$, such that $z_b < z'_b$ and $z_b < z''_b$; let P' denote the lowest one, such that $z_b < z'_b \leq z''_b$. Then $(x'', y'', z_b) \notin \mathcal{C}$, which contradicts that **(e)** does not apply. Therefore, there can be at most one such pillar. However, this, together with Lemma 1, implies that on all other sides $(x'', y'') \neq (x', y')$, $(x'', y'', z_b) \in \mathcal{C}$. This means that $(x, y, z'_b - 1)$ is a non-cut cube, contradicting that **(b)** does not apply. Therefore, there can be no such adjacent pillars. ◀

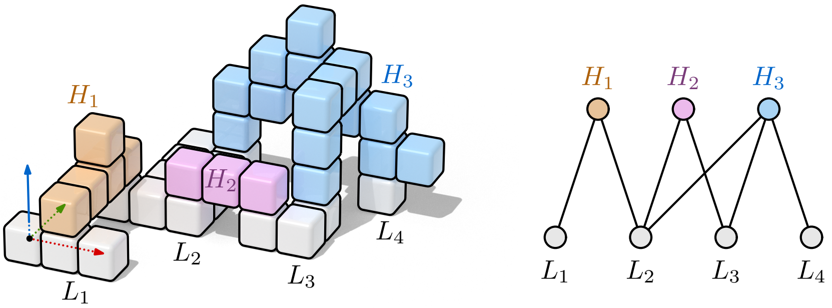


Figure 4 An example configuration $\mathcal{C}$ and its low-high graph $\mathcal{LH}_\mathcal{C}$. This configuration does still admit operations of type **(a–g)**.

In summary, if **(a–e)** do not apply to any non-cut subpillar, then for any non-cut pillar $P = \langle x, y, z_b \mathinner{..} z_t \rangle$ and any adjacent pillar $\langle x', y', z'_b \mathinner{..} z'_t \rangle$, we have $z'_b = z_b = 0$.

Local potential reduction. Let $C_{>0}$ be the subconfiguration consisting of cubes with $z > 0$. We may greedily reduce the potential by moving individual cubes in $C_{>0}$.

(f) Perform any move of $\mathcal{C}$ that moves a cube c of $\mathcal{C}_{>0}$, reduces the potential, and keeps c inside the bounding box $B_\mathcal{C}$ of $\mathcal{C}$.

▶ **Lemma 3.** *If an unfinished configuration $\mathcal{C}$ does not admit any operation of type* **(a–f)**, *and some maximal connected component of $\mathcal{C}_{>0}$ consists of a single pillar $P = \langle x, y, z_b \mathinner{..} z_t \rangle$, then $P = \{(0, 0, 1)\}$ and $(0, 0, 0) \in \mathcal{C}$.*

Proof. Because $\mathcal{C}_{>0}$ does not contain cubes with $z = 0$, we have $z_b = 1$ or $z_b > 1$. If $z_b > 1$, then $\langle x, y, z_b \mathinner{..} z_t \rangle$ would be disconnected from the rest of $\mathcal{C}$, so this cannot be the case. Likewise, if $z_b = 1$ then $(x, y, 0) \notin \mathcal{C}$ would mean that $\langle x, y, z_b \mathinner{..} z_t \rangle$ is disconnected from $\mathcal{C}$, so this cannot be the case either. Therefore $z_b = 1$ and $(x, y, 0) \in \mathcal{C}$. If $z_t > 1$, then the topmost cube of P can do a convex transition to $(x + 1, y, z_t - 1)$, reducing the potential. Therefore $z_t = 1$ and $P = \{(x, y, 1)\}$. If $x > 0$ or $y > 0$, then we can move the single cube of P: using the cube at $(x, y, 0)$, the cube of P can slide or convex transition closer to the origin $(0, 0, 0)$. Therefore, $z_b = z_t = 1$ and $x = y = 0$. ◀

▶ **Corollary 4.** *If a configuration $\mathcal{C}$ does not admit any operation of type* **(a–f)** *then of the connected components of $\mathcal{C}_{>0}$, at most one consists of a single pillar.*

Low and high components. Let $\mathcal{LH}_\mathcal{C}$ be the bipartite graph obtained from $G_\mathcal{C}$ by contracting the components of $G_{\mathcal{C}_0}$ and $G_{\mathcal{C}_{>0}}$ to a single vertex (see Figure 4). We call $\mathcal{LH}_\mathcal{C}$ the *low-high graph* of $\mathcal{C}$, and we call the vertices of $\mathcal{LH}_\mathcal{C}$ that correspond to components of $G_{\mathcal{C}_0}$ and $G_{\mathcal{C}_{>0}}$ *low* and *high* components, respectively. For brevity, we may refer to a low or high component by its corresponding vertex in $\mathcal{LH}_\mathcal{C}$ and vice versa.

We will use the following lemma several times.

▶ **Lemma 5.** *Let H be a high component and P be a pillar of H. For every component H' of $H \setminus P$, there exists a non-cut pillar of H' that is also a non-cut pillar of H.*

Proof. Any component with at least two pillars contains at least two non-cut pillars and every component contains at least one non-cut pillar, so let P' be an arbitrary non-cut pillar of H'. $H \setminus P'$ has at most two components, namely $H' \setminus P'$ and $H \setminus H'$. If P' is a non-cut pillar of H, the lemma holds. Else, if P' is a cut pillar of H, then it has exactly these components, so $H' \setminus P'$ is nonempty and P is adjacent to P'. Therefore, H' consists of multiple pillars and hence contains at least two non-cut pillars. Let $P'' \neq P'$ be a second non-cut pillar of H'. We claim that P'' is also a non-cut pillar of H. Indeed, because P and P' are adjacent, the sets $H' \setminus P'' \supseteq P'$ and $H \setminus H' \supseteq P$ are adjacent. Hence, $H \setminus P'' = (H' \setminus P'') \cup (H \setminus H')$ has a single component, so P'' is a non-cut pillar of H. ◀

▶ **Lemma 6.** *Assume $\mathcal{C}$ does not admit any operation of type* **(a–f)**. *Suppose that H is a high component such that $\mathcal{C} \setminus H$ is connected. Then any pillar of H is a non-cut subpillar of $\mathcal{C}$.*

Proof. Suppose for a contradiction that a pillar P of H is a cut subpillar of $\mathcal{C}$. Then $\mathcal{C} \setminus P$ contains at least one component H' that does not intersect $\mathcal{C} \setminus H$. Therefore, H' is also a component of $H \setminus P$, so by Lemma 5, there exists a non-cut pillar $P' = \langle x', y', z'_b \,..\, z'_t \rangle$ of H' that is also a non-cut pillar of H. If $z'_b > 1$ or $(x', y', 0) \notin \mathcal{C}$, then P' would be a non-cut pillar of $\mathcal{C}$. If $\mathcal{C}$ does not admit any operation of type **(a–f)**, all non-cut pillars of $\mathcal{C}$ start at $z = 0$, which contradicts $z'_b > 1$ or $(x', y', 0) \notin \mathcal{C}$. Therefore, $z'_b = 1$ and $(x', y', 0) \in \mathcal{C}$, but then H' would not be a component of $\mathcal{C} \setminus P$, as H' is adjacent to $(x', y', 0) \in \mathcal{C} \setminus P$. Hence, H' cannot exist, completing the proof. ◀

▶ **Corollary 7.** *If H is a high component such that $\mathcal{C} \setminus H$ is connected, then every pillar of H is part of a pillar of $\mathcal{C}$ starting at $z = 0$.*

▶ **Lemma 8.** *Assume $\mathcal{C}$ does not admit any operation of type* **(a–f)**. *If H is a high component such that $\mathcal{C} \setminus H$ is connected, then H consists entirely of finished cubes.*

Proof. Assume for contradiction that H contains an unfinished cube. Because of Corollary 7, every pillar of H is part of a pillar of $\mathcal{C}$ starting at $z = 0$. Therefore, H contains an unfinished cube (x, y, z) with $x > 0$ or $y > 0$. Let c be such a cube that lexicographically maximizes $(z, -y, -x)$. If $x > 0$ and $(x - 1, y, z) \notin H$ (and thus $(x - 1, y, z) \notin \mathcal{C}$), then we can move c to either $(x - 1, y, z)$ or $(x - 1, y, z - 1)$, reducing the potential while keeping all cubes within the bounding box $B_{\mathcal{C}}$, so **(f)** would apply. If $y > 0$ and $(x, y - 1, z) \notin H$, then we can similarly move c to either $(x, y - 1, z)$ or $(x, y - 1, z - 1)$. On the other hand, if both (1) $x = 0$ or $(x - 1, y, z) \in H$ and (2) $y = 0$ or $(x - 1, y, z) \in H$, then because c is the unfinished cube of H that maximizes $(z, -y, -x)$, the cubes $(x - 1, y, z)$ (if $x > 0$) and $(x, y - 1, z)$ (if $y > 0$) are finished, but then (x, y, z) would also be finished. Contradiction. ◀

▶ **Corollary 9.** *There is at most one high component that contains a finished cube, as any high component that contains a finished cube also contains* $(0, 0, 1)$.

Handling low components. We pick a vertex R of $\mathcal{LH}_{\mathcal{C}}$ that we call the *root* of $\mathcal{LH}_{\mathcal{C}}$. If $(0, 0, 0) \in \mathcal{C}$, pick R to be the low component that contains $(0, 0, 0)$. Otherwise, pick R to be an arbitrary low component. Let d be the maximum distance in the graph $\mathcal{LH}_{\mathcal{C}}$ from R to any vertex. Let $\mathcal{U}$ be the set of vertices of $\mathcal{LH}_{\mathcal{C}}$ that are locally furthest away (in $\mathcal{LH}_{\mathcal{C}}$) from R. That is, all neighbors v of a vertex $u \in \mathcal{U}$ lie closer to R. All vertices of $\mathcal{U}$ are non-cut subsets of $\mathcal{C}$. Therefore, if $\mathcal{U}$ contains a high component H, then H consists entirely of finished cubes (and H is adjacent to R), so $\mathcal{U}$ contains at most one high component.

If $d = 0$, then $\mathcal{C}$ consists of a single low component. If $d = 1$, then $\mathcal{C}$ consists of one high and one low component. Set $\mathcal{U}$ contains exactly one high component, and it consists entirely of finished cubes. If $d \geq 2$, then $\mathcal{C}$ consists of at least two low components and $\mathcal{U}$ consists of at least one low component, and at most one high component. We now give operations that can be executed when $d \geq 2$, such that we end up with a configuration where $d = 0$ or $d = 1$. We will show how to handle the case where $d = 0$ or $d = 1$ afterwards.

We call a low component L *clear* if $\mathcal{C} \setminus L$ is connected, $L \neq R$, and L is connected to a non-cut pillar P in $\mathcal{C} \setminus L$. We call such a pillar P a *clearing pillar.* We show in Lemma 10 that if $d \geq 2$, there is at least one clear low component. For this, consider a low component that is furthest from R (that is, at distance d), and let H be an adjacent high component. Let $\mathcal{L}_H$ be the set of low components in $\mathcal{U}$ that are adjacent to H (and hence also lie at distance d from R).

▶ **Lemma 10.** *At least one low component $L \in \mathcal{L}_H$ is connected to H via a non-cut pillar of $\mathcal{C} \setminus L$.*

Proof. Let $\mathcal{C}' = \mathcal{C} \setminus \bigcup_{L \in \mathcal{L}_H} L$. Let $H_{\mathcal{L}_H}$ be the set of cubes of H that are adjacent to a low component in $\mathcal{L}_H$. Fix an arbitrary cube c_s of R, and in the graph $G_{\mathcal{C}'}$, consider a cube c of $H_{\mathcal{L}_H}$ that is farthest from c_s. Let P be the pillar of H that contains c. We will show that P is a non-cut pillar of $\mathcal{C}'$. Suppose for a contradiction that P is a cut pillar, then $\mathcal{C}' \setminus P$ contains at least two components, at most one of which contains R. Let H' be a component of $\mathcal{C}' \setminus P$ not containing R. If H' contains a cube not in H, then that cube lies in a low or high component of $\mathcal{C}'$ that lies closer to R, which therefore remains connected to R after removing P. Therefore, H' is a subset of H.

All cubes $(x, y, z) \in H'$ with $z = 1$ lie farther from c_s than c, and therefore H' does not contain any cubes of $H_{\mathcal{L}_H}$, so H' is not adjacent to any cubes of $\mathcal{L}_H$. Therefore, H' is not adjacent to any cubes of $(\mathcal{C} \setminus P) \setminus H'$. By Lemma 5, H' contains at least one non-cut pillar P' that is also a non-cut pillar of H. P' is also a non-cut pillar of $\mathcal{C}$, but all non-cut pillars of $\mathcal{C}$ start at $z = 0$ (Corollary 7), which is a contradiction. ◀

We will now present an algorithm that repeatedly selects a (non-root) clear low component L, and performs the following operation on it:

(g) Perform any move of $\mathcal{C}$ that moves a cube c of L, reduces the potential, and keeps c inside the bounding box $B_{\mathcal{C}}$ of $\mathcal{C}$.

Note that **(g)** is essentially the same as **(f)**, but now executed on a low rather than a high component. When operations of type **(g)** are executed, one of three special events can occur:

(1) L connects to a different low component, merging them.

(2) L connects to the root, and becomes part of the root.

(3) L reaches the origin (at which point it becomes the root).

When none of the operations **(a–g)** are available for a clear low component L with clearing pillar $\langle p_x, p_y, z_b \mathinner{..} z_t \rangle$, there are two cases. Either L contains enough cubes to reach the origin ($|L| \geq x + y$), or L does not contain enough cubes to reach the origin ($|L| < x + y$). We call these low components *big* and *small* respectively and we handle them differently. For small low components, we would like to shove the clearing pillar. However, this might not be valid and might disconnect the configuration in the process. Therefore, we will devise a special operation that is only safe on small low components.

Small low components. If a clear low component is too small to reach the origin, we want to move the clearing pillar and do a pillar shove. However, it could be that moving the clearing pillar would disconnect the low component, or that there are cubes around the clearing pillar obstructing the shove. For both of these situations, we devise a new operation. Let N_P be the set of cells c with $z = 1$ in $B_{\mathcal{C}}$ neighboring P.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a configuration that does not admit operations of type* **(a–g)**. *Let L be a clear component and $P = \langle x, y, z_b \mathinner{..} z_t\rangle$ be its clearing pillar. Assume $z_t \geq 2$. The cubes in N_P need to be either all present, or all absent from $\mathcal{C}$.*

Proof. Assume for a contradiction that at least one, but at most three of the cubes neighboring P with $z = 1$ are present. Denote these cubes by c_1, c_2, and c_3; not all need to be present. Since P is a clearing pillar, $(\mathcal{C}\backslash L)\backslash P$ is connected. If the cube $(x, y, 2)$ is completely surrounded by cubes, then it can do a potential reducing operation of type **(b)**. Otherwise, the cube $(x, y, 2)$ can do a potential reducing operation of type **(f)**. Both lead to a contradiction. ◀

Depending on if the cubes in N_P are present, we perform the following operations on L.

(h) Let L be a clear component and $P = \langle x, y, z_b \mathinner{..} z_t\rangle$ be its clearing pillar. Assume at least one of the cubes in N_P is present. By definition, $L \neq R$. Therefore, there exists an empty cell e, with coordinates $(e_x, e_y, 0)$, with $e_x < x$ or $e_y < y$. Let e be such an empty cell with highest y, and from those, the one with highest x. We now want to take the cube $c = (x, y, 0)$ and move it on a shortest path via $z = -1$ towards e, reducing its potential. This however, could disconnect parts of L if c is a cut cube of L and if $z_t = 1$. (If $z_t \geq 2$, then by Lemma 11, all cubes in N_P are present and c is not a cut cube of the configuration.) In this case, we first gather cubes from L to fill the 3×3 horizontal square centered around c, making c a non-cut cube, before moving c towards e. Now the configuration stays connected when moving c to e. If we gathered cubes because $z_t = 1$, the head of P at $(x, y, 1)$ is not a cut cube, and can subsequently move down.

If **(a–h)** do not apply, then all cubes from N_P are not in $\mathcal{C}$.

(i) Let L be a clear component and $P = \langle x, y, z_b \mathinner{..} z_t\rangle$ be its clearing pillar. All cubes in N_P are absent from $\mathcal{C}$. Gather cubes from L towards P according to Figure 5 and do a pillar shove on P. Then, move the extra cubes back to their original location.

The only reason that **(e)** is not possible, is that P is a cut pillar, since it is the only pillar connecting L to the other components. Therefore, gathering cubes from L to P makes the operations **(h)** and **(i)** viable. This is done in the following way. Let the clearing pillar of L be $P = \langle x, y, z_b \mathinner{..} z_t\rangle$. Let z_t be the highest z such that only (x, y, z_t) has a horizontal neighboring cube (x', y', z_t). Let $c = (x, y, 0) \in L$ be the cube below P. Assume that P has at least size 5. Then, repeatedly select the non-cut cube $c \in L$ that lexicographically maximizes (z, y, x) and move it towards P to fill the 3×3 square (for **(h)**), or create the configuration shown in Figure 5a (for **(i)**). The cubes that were gathered keep the configuration connected during the operation. For **(i)**, if P has fewer than 5 cubes, we gather cubes towards P in a different way. Because P is too small to gather enough cubes for a pillar shove, we simply fill the cells that P would want to go towards, see Figure 5b. Then, the original P can be deconstructed. Again, using a constant number of moves, we can decrease the potential vector, while maintaining connectivity.

▶ **Lemma 12.** *Operations of type* **(h)** *and* **(i)** *are safe.*

Proof. For an operation of type **(h)** or **(i)**, let $c = (c_x, c_y, c_z)$ be the head of P. We will show that any operation of type **(i)** strictly decreases $P_\mathcal{C}$ by $O(c_z + c_y + c_x)$ and uses $O(c_z + c_y + c_x)$ moves to do so. First we analyze the operations of type **(h)** or **(i)** with a pillar of size larger than one: the head $c = (c_x, c_y, c_z)$ of the pillar involved moves down from c_z to $z = 1$, so $P_\mathcal{C}$ reduces by $4(c_z - 1)$. The cubes beneath c from $z = 1$ up to c_z might increase their x- or y-coordinate by 1. Therefore, the potential also increases by $2(c_z - 1)$ at most. The cubes that are gathered and then returned do not move positions and therefore do not affect the

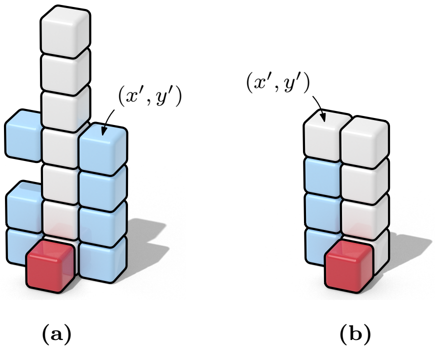


Figure 5 The start configuration for a pillar shove for a clearing pillar. The white pillar is the clearing pillar. The red cube is part of L. The blue cubes are required and need to be gathered. (a) Clearing pillar of height at least 5. (b) The configuration for a pillar shove of height at most 4.

potential. Moreover, w_c becomes one lower, because c_z decreases from $c_z > 1$ to $c_z = 1$. In total, the potential $P_{\mathcal{C}}$ decreases by $2(c_z - 1) + c_x + c_y$. For operations of this type with a pillar of size one, the potential decreases by $c_z + c_y + c_x$, since the head moves from $z = 1$ to $z = 0$.

Now we will show that executing one of these operations, which reduces $\Pi_{\mathcal{C}}$ by $O(c_z + c_y + c_x)$, takes $O(c_z + c_y + c_x)$ moves and is therefore safe. Moving via a shortest path over a component with x cubes takes $O(x)$ moves. Gathering the seven cubes from L to the clearing pillar and moving them back takes at most $O(c_x + c_y)$ moves, since L is a small low component and has therefore size at most $O(c_x + c_y)$. Then, the normal pillar shove takes $O(c_z)$ moves. Hence, the total operation takes $O(c_z + c_y + c_x)$ moves and is safe. ◀

Big low components. If a low component L is big, that is, if it contains enough cubes to reach the origin from its clearing pillar, we want L to actually contain the origin. Performing operations of types **(g)** is not sufficient to achieve this, and operations **(h)** and **(i)** are only safe on small low components. To make L contain the origin, note that all of our operations not only work in 3D, but also in 2D when instead of prioritizing reducing the z-coordinate, we prioritize reducing the y-coordinate. We run the algorithm on L in 2D, with an additional constraint. We fix an arbitrary clearing pillar of L, and call the cube p of L below that clearing pillar its *pinned cube*. When executing the algorithm in 2D, we never move p. With minor changes, all of the lemmas above still hold in the presence of at most one pinned cube.

We abstract from L being a clear low component, and instead consider a component $\mathcal{C}$ in which we disallow a single cube $p \in \mathcal{C}$ from moving. We again call this cube p the *pinned cube*. We adapt our algorithm for configurations without pinned cubes to one for configurations with pinned cubes as follows. Whenever we are looking for the next operation to perform, simply disregard any operation that would move the pinned cube. We cannot guarantee that this adapted algorithm results in a finished configuration, but for our purposes, it is sufficient to prove that it reaches the origin if it is big enough.

Recall that R is a vertex of $\mathcal{LH}_{\mathcal{C}}$ chosen as follows. If $\mathcal{C}$ contains a low component that contains the origin, then let R be that low component. Otherwise, we choose R to be an arbitrary low component. If $\mathcal{C}$ already contains the origin, the lemma trivially holds, so we would choose R to be an arbitrary low component. However, since there exists a pinned cube p, we need to be more careful with our choice and instead pick R as follows. If there exists a low component that either contains p or that neighbors a pillar containing p, let R be that low component. We are now ready to prove the following lemma.

▶ **Lemma 13.** *Let $\mathcal{C}$ be a configuration with a single pinned cube $p = (p_x, p_y, p_z)$ and assume $\mathcal{C}$ has at least $p_x + p_y + p_z$ cubes. If $\mathcal{C}$ does not admit operations of type* **(a–i)** *that do not move p, then $\mathcal{C}$ contains the origin.*

Proof. Assume there are at least two low components. Let L_1 and L_2 be the low components such that the distance between them in $\mathcal{LH}_\mathcal{C}$ is maximized. Hence, if there would be no pinned cube p, we could pick any of them and the other would be clear (Lemma 10). Because $\mathcal{C}$ only contains a single pinned cube, we pick R to be either L_1 or L_2 such that the other one is clear. Therefore, there is always a clear low component. While a clear low component exists, we can execute operations **(g–i)** if it is small, or perform operations one dimension lower if it is big and does not contain the origin. This is a contradiction and therefore there can be at most one low component L.

Because no operations of type **(h)** or **(i)** are possible on L, its clearing pillar must contain p. Furthermore, no operations are possible on L in one dimension lower, so L must contain its own origin by recursion. Because there are no possible operations of type **(g)** on L and because it is big enough, this means that L also contains the global origin. ◀

If none of the operations **(a–g)** are possible, every clear low component either contains the origin, or is too small to do so.

The algorithm terminates when no clear low component (and hence only the root low component) remains. Recall that d is the maximum distance from the root R of $\mathcal{LH}_\mathcal{C}$ over all vertices. We are left with two cases. Either no high component remains ($d = 0$), or there is at most one high component ($d = 1$), which consists of entirely finished cubes.

As stated before, all operations **(a–i)** not only work in 3 dimensions, they also work in 2 dimensions when instead of prioritizing reducing the z-coordinate, we prioritize reducing the y-coordinate. Moreover, these operations never move the origin. Therefore, we can now run the exact same operations on the bottom layer in 2D, until the root component is finished. If there is still a high component, it stays connected via the origin. We end up with a finished configuration.

Running time. Recall that the *potential* of a cube $c = (c_x, c_y, c_z)$ is $\Pi_c = w_c(c_x + 2c_y + 4c_z)$, where the *weight* w_c depends on the coordinates of c in the following way. If $c_z > 1$, then $w_c = 5$, if $c_z = 1$, then $w_c = 4$. If $c_z = 0$, then w_c depends on c_y. If $c_y > 1$, then $w_c = 3$, if $c_y = 1$, then $w_c = 2$ and lastly, if both $c_z = c_y = 0$, then $w_c = 1$. The potential of the complete configuration is the sum of potential of the individual cubes. Moreover, a sequence of m moves is *safe* if the result is a configuration $\mathcal{C}'$ inside $B_\mathcal{C}$, such that $\Pi_{\mathcal{C}'} < \Pi_\mathcal{C}$ and $m = O(\Pi_\mathcal{C} - \Pi_{\mathcal{C}'})$. Each operation of type **(a–i)** strictly reduces the potential function. Moreover, each of the operations **(a–g)** is trivially safe. We have shown that operations **(h)** and **(i)** are also safe (see Lemma 12).

Because all operations are safe and reduce the potential, the algorithm performs at most $O(\Pi_\mathcal{C}) = O(X_\mathcal{C} + Y_\mathcal{C} + Z_\mathcal{C})$ moves. For the problem of reconfiguring the cubes into a finished configuration, this is worst-case optimal. An example achieving this bound is a configuration consisting of a path of cubes in a bounding box of equal side lengths w tracing from the origin to the opposite corner of the bounding box. To see that, note that any finished cube at position (x, y, z) requires there to exist $n \geq x \cdot y \cdot z$ cubes, so at least one of x, y, and z is at most $n^{1/3}$ for any candidate finished position. There are $\Omega(n)$ cubes (x', y', z') that are initially $\Omega(w - n^{1/3}) = \Theta(n) = \Theta(x' + y' + z')$ away from any such potential finished position.

4 Conclusion

We presented an in-place algorithm that reconfigures any configuration of cubes into a compact canonical shape using a number of moves proportional to the sum of coordinates of the input cubes. This result is asymptotically optimal. However, just as many other algorithms in the literature, our bounds are amortized in the sense that we make use of a number of dedicated cubes which help other cubes move by establishing the necessary connectivity in their neighborhood. This is in particular the case with our pillar shoves, that need some additional cubes to gather at the pillar, to then move up and down the pillar to facilitate moves. These extra moves are charged to one cube in the pillar reducing its coordinates. In the literature such cubes are referred to as *helpers*, *seeds*, or even *musketeers* [2, 7, 13, 17].

Such helping cubes are in many ways in conflict with the spirit of modular robot reconfiguration: ideally each module should be able to run the same program more or less independently, without some central control system sending helpers to those places where they are needed. The input-sensitive Gather&Compact algorithm in 2D by Akitaya et al. [4] does not require amortized analysis and gives a bound on the number of moves for each square in terms of the perimeters of the input and output configurations. The question hence arises whether it is possible to arrive at sum-of-coordinates bounds either in 2D or 3D without amortization? For example, is there a compaction algorithm in which each cube in the configuration that starts at position (x, y, z) performs at most $O(x + y + z + a)$ moves, where a is the average L_1-distance that cubes lie from the origin?

References

1 Zachary Abel and Scott Duke Kominers. Universal reconfiguration of (hyper-)cubic robots. *arXiv e-Prints*, 2011. arXiv:0802.3414v3.

2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The O(1) musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/S00453-020-00784-6.

3 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Dylan H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Proc. 37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *LIPIcs*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.

4 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Proc. 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 227 of *LIPIcs*, pages 4:1–4:19, 2022. doi:10.4230/LIPICS.SWAT.2022.4.

5 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Val Pinciu, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Efficient constant-velocity reconfiguration of crystalline robots. *Robotica*, 29(1):59–71, 2011. doi:10.1017/S026357471000072X.

6 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry*, 42(6):652–663, 2009. doi:10.1016/j.comgeo.2008.11.003.

7 Matthew Connor and Othon Michail. Centralised connectivity-preserving transformations by rotation: 3 musketeers for all orthogonal convex shapes. In *Proc. 18th International Symposium on Algorithmics of Wireless Networks (ALGOSENSORS 2022)*, volume 13707 of *LNCS*, pages 60–76. Springer, 2022. doi:10.1007/978-3-031-22050-0_5.

8 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006. doi:10.1007/s00373-005-0640-1.

9 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. In *Proc. 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *LIPIcs*, pages 9:1–9:16, 2021. doi:10.4230/LIPIcs.ISAAC.2021.9.

10 Daniel Feshbach and Cynthia Sung. Reconfiguring non-convex holes in pivoting modular cube robots. *IEEE Robotics and Automation Letters*, 6(4):6701–6708, 2021. doi:10.1109/LRA.2021.3095030.

11 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 2003)*, volume 3, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

12 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán. Distributed reconfiguration of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38:383–413, 2015. doi:10.1007/s10514-015-9421-8.

13 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *LIPIcs*, pages 136:1–136:15, 2017. doi:10.4230/LIPICS.ICALP.2017.136.

14 Tillmann Miltzow, Irene Parada, Willem Sonke, Bettina Speckmann, and Jules Wulms. Hiding sliding cubes: Why reconfiguring modular robots is not easy. In *Proc. 36th International Symposium on Computational Geometry, (SoCG 2020, Media Exposition)*, volume 164 of *LIPIcs*, pages 78:1–78:5, 2020. doi:10.4230/LIPICS.SOCG.2020.78.

15 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proc. 36th European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020.

16 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. doi:10.1007/s10514-021-09977-6.

17 Frederick Stock, Hugo Akitaya, Matias Korman, Scott Kominers, and Zachary Abel. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in quadratic time. In *Proc. 40th International Symposium on Computational Geometry (SoCG)*, 2024. To appear.

18 Cynthia R. Sung, James M. Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 2015)*, pages 1933–1940, 2015. doi:10.1109/ICRA.2015.7139451.

Canonizing Graphs of Bounded Rank-Width in Parallel via Weisfeiler–Leman

Michael Levet[1] ✉
Department of Computer Science, College of Charleston, SC, USA

Puck Rombach ✉
Department of Mathematics and Statistics, University of Vermont, Burlington, VT, USA

Nicholas Sieger ✉
Department of Mathematics, University of California San Diego, La Jolla, CA, USA

Abstract

In this paper, we show that computing canonical labelings of graphs of bounded rank-width is in TC^2. Our approach builds on the framework of Köbler & Verbitsky (CSR 2008), who established the analogous result for graphs of bounded treewidth. Here, we use the framework of Grohe & Neuen (*ACM Trans. Comput. Log.*, 2023) to enumerate separators via *split-pairs* and *flip functions*. In order to control the depth of our circuit, we leverage the fact that any graph of rank-width k admits a rank decomposition of width $\leq 2k$ and height $O(\log n)$ (Courcelle & Kanté, WG 2007). This allows us to utilize an idea from Wagner (CSR 2011) of tracking the depth of the recursion in our computation.

Furthermore, after splitting the graph into connected components, it is necessary to decide isomorphism of said components in TC^1. To this end, we extend the work of Grohe & Neuen (*ibid.*) to show that the $(6k+3)$-dimensional Weisfeiler–Leman (WL) algorithm can identify graphs of rank-width k using only $O(\log n)$ rounds. As a consequence, we obtain that graphs of bounded rank-width are identified by $\mathsf{FO} + \mathsf{C}$ formulas with $6k+4$ variables and quantifier depth $O(\log n)$. Prior to this paper, isomorphism testing for graphs of bounded rank-width was not known to be in NC.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Circuit complexity; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Graph Isomorphism, Weisfeiler–Leman, Rank-Width, Canonization, Descriptive Complexity, Circuit Complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.32

Related Version *Full Version*: https://arxiv.org/abs/2306.17777

Funding This work was completed in part at the 2022 Graduate Research Workshop in Combinatorics, which was supported in part by NSF grant #1953985 and a generous award from the Combinatorics Foundation.
Michael Levet: Partially supported by J. A. Grochow's NSF award CISE-2047756 and the University of Colorado Boulder, Department of Computer Science Summer Research Fellowship.

1 Introduction

The Graph Isomorphism problem (GI) takes as input two graphs G and H, and asks if there exists an isomorphism $\varphi : V(G) \to V(H)$. GI is in particular conjectured to be NP-intermediate; that is, belonging to NP but neither in P nor NP-complete [36]. Algorithmically, the best known upper-bound is $n^{\Theta(\log^2 n)}$, due to Babai [3]. It remains open as to whether

[1] Corresponding author

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

GI belongs to P. There is considerable evidence suggesting that GI is not NP-complete [42, 6, 30, 3, 33, 1, 39]. In a precise sense, GI sits between linear and multilinear algebra. For any field $\mathbb{F}$, GI belongs to $\mathbb{F}$-Tensor Isomorphism ($\mathsf{TI}_{\mathbb{F}}$). When $\mathbb{F}$ is finite, $\mathsf{TI}_{\mathbb{F}} \subseteq \mathsf{NP} \cap \mathsf{coAM}$ [16, 17]. In contrast, the best known lower-bound for GI is DET [44], which contains NL and is a subclass of TC^1. It is thus natural to inquire as to families of graphs where isomorphism is decidable in sub-classes of DET.

There has been considerable work on efficient algorithms for special families of graphs. Sparse graphs, in particular, have received considerable attention from the perspective of polynomial-time computation, though less is known for dense graphs– see [24] for discussion. The story is similar in the setting of NC isomorphism tests, with considerable work on planar graphs (see the references in [12]) and graphs of bounded treewidth [25, 46, 11], culminating in L-completeness results for both families (see [12, 43] for planar graphs, and and [15] for graphs of bounded treewidth). Isomorphism testing for graphs of bounded genus is also L-complete [14]. We now turn to dense graphs. An NC isomorphism test is known for graphs of bounded eigenvalue multiplicity [2]. The isomorphism problems for interval graphs [32] and Helly ciruclar-arc graphs [35] are both L-complete.

The k-dimensional Weisfeiler–Leman algorithm (k-WL) serves as a key combinatorial tool in GI. It works by iteratively coloring k-tuples of vertices in an isomorphism-invariant manner. On its own, Weisfeiler–Leman serves as an efficient polynomial-time isomorphism test for several families of graphs, including planar graphs [31, 21], graphs of bounded genus [18, 20], and graphs for which a specified minor H is forbidden [19]. WL even serves as an NC isomorphism test for graphs of bounded treewidth [25] and planar graphs [25, 45, 21]. Despite the success of WL, it is insufficient to place GI into P [7, 40]. Nonetheless, WL remains an active area of research. For instance, Babai's quasipolynomial-time algorithm [3] combines $O(\log n)$-WL with group-theoretic techniques.

Graphs of bounded rank-width have only recently received attention from the perspective of isomorphism testing. Grohe & Schweitzer [24] gave the first polynomial-time isomorphism test for graphs of bounded rank-width. In particular, their isomorphism test ran in time $n^{f(k)}$, where $f(k)$ was a non-elementary function of the rank-width k. Subsequently, Grohe & Neuen [22] showed that graphs of rank-width k have Weisfeiler–Leman dimension $\leq 3k+5$, which yields an $O(n^{3k+6}\log n)$-time isomorphism test and also the first polynomial-time canonical labeling procedure for this family. In particular, it is open as to whether graphs of bounded rank-width admit NC or FPT isomorphism tests. This is in contrast to graphs of bounded treewidth, where NC [25, 34, 46, 11, 15] and FPT [38, 23] isomorphism tests are well-known.

Closely related to Graph Isomorphism is Graph Canonization, which for a class $\mathcal{C}$ of graphs, asks for a function $F : \mathcal{C} \to \mathcal{C}$ such that for all $X, Y \in \mathcal{C}$, $X \cong F(X)$ and $X \cong Y \iff F(X) = F(Y)$. Graph Isomorphism reduces to Graph Canonization, and the converse remains open. Nonetheless, efficient canonization procedures have often followed efficient isomorphism tests, usually with non-trivial work– see e.g., [29, 22, 34, 46, 15, 4].

Main Results. In this paper, we investigate the parallel and descriptive complexities of identifying and canonizing graphs of bounded rank-width, using the Weisfeiler–Leman algorithm.

▶ **Theorem 1.** *Let G be a graph on n vertices, of rank-width k. We can compute a canonical labeling for G using a TC circuit of depth $O(\log^2 n)$ and size $n^{O(16^k)}$.*

Our approach in proving Thm. 1 was inspired by the previous work of Köbler & Verbitsky [34], who established the analogous result for graphs of bounded treewidth. Köbler & Verbitsky crucially utilized the fact that graphs of treewidth k admit *balanced* separators of size $k+1$, where removing such a separator leaves connected components each of size $\leq n/2$. This ensures that the height of their recursion tree is $O(\log n)$. For graphs of bounded rank-width, we are unable to identify such separators. Instead, we leverage the framework of Grohe & Neuen to descend along a rank decomposition, producing a canonical labeling along the way. To ensure that our choices are canonical, we utilize the Weisfeiler–Leman algorithm. As a first step, we will establish the following:

▶ **Theorem 2.** *The* $(6k+3)$*-dimensional Weisfeiler–Leman algorithm identifies graphs of rank-width* k *in* $O(\log n)$ *rounds.*

Combining Thm. 2 with the parallel WL implementation of Grohe & Verbitsky [25], we obtain the first NC bound for isomorphism testing of graphs of bounded rank-width. This is a crucial ingredient in obtaining the TC^2 bound for Thm. 1.

▶ **Corollary 3.** *Let* G *be a graph of rank-width* $O(1)$*, and let* H *be arbitrary. We can decide isomorphism between* G *and* H *in* TC^1*.*

Furthermore, in light of the close connections between Weisfeiler–Leman and $\mathsf{FO}+\mathsf{C}$ [29, 7], we obtain the following corollary. Let $\mathcal{C}_{m,r}$ denote the m-variable fragment of $\mathsf{FO}+\mathsf{C}$ where the formulas have quantifier depth at most r (see Sec. 2.3).

▶ **Corollary 4.** *For every graph* G *of rank-width at most* k*, there is a sentence* $\varphi_G \in \mathcal{C}_{6k+4,O(\log n)}$ *that characterizes* G *up to isomorphism. That is, whenever* $H \not\cong G$*, we have that* $G \models \varphi_G$ *and* $H \not\models \varphi_G$*.*

We will discuss shortly the proof technique for Thm. 2. We first discuss how we will utilize Thm. 2 to establish Thm. 1. As $(6k+3)$-WL identifies all graphs of rank-width $\leq k$ in $O(\log n)$ rounds, $(10k+3)$-WL identifies the orbits of sequences of vertices of length $\leq 4k$ (see Lem. 21). By applying $(10k+3)$-WL for $O(\log n)$ rounds at each recursive call to our canonization procedure, we give canonical labelings to the various parallel choices considered by the algorithm. While there exists a suitable rank decomposition of height $O(\log n)$ [8], it is open whether such a decomposition can be computed in NC [10]. Instead of explicitly constructing a rank decomposition, we instead track the depth of our recursion tree. By leveraging the framework of Grohe & Neuen [22], we show that one of our parallel computations witnesses the balanced rank decomposition of Courcelle & Kanté [8].

We will now outline the proof strategy for Thm. 2. Our work follows closely the strategy of Grohe & Neuen [22]. We again combine the balanced rank decomposition from [8] with a careful analysis of the pebbling strategy of Grohe & Neuen [22]. In parts of their argument, Grohe & Neuen utilize (an analysis of) the stable coloring of 1-WL. For a graph G, Grohe & Neuen [22, Sec. 3] construct an auxiliary graph that they call the *flipped graph*, whose construction depends on a specified set of vertices called a *split pair* and a coloring of the vertices. While the flipped graph is compatible with any vertex coloring, Grohe & Neuen [22, Lem. 3.6] crucially utilize the *stable coloring* of 1-WL to show that WL can detect which edges are present in the flipped graph. Even though we allow for higher-dimensional WL, the restriction of $O(\log n)$ rounds creates a technical difficulty in adapting [22, Lem. 3.6].

To resolve this issue, we consider a *different* notion of flipped graph– namely, a vertex colored variant introduced in [22, Sec. 5]. In this second definition, edges of the flipped graph depend only on the split pair and not the coloring. Grohe & Neuen established [22, Lem. 5.6], which is analogous to their Lem. 3.6. The proof of their Lem. 5.6 depends only on the structure of the graph and *not* the vertex colorings. In particular, Weisfeiler–Leman can take advantage of [22, Lem. 5.6] within $O(\log n)$ iterations.

The second such place where Grohe & Neuen rely on the stable coloring of 1-WL to detect the connected components of the flipped graph. We will show that 2-WL can in fact identify these components in $O(\log n)$ rounds. We further reduce the round complexity via a simple observation. In the pebble game characterization, if Duplicator fails to respect connected components of the flipped graph, Spoiler can win in $O(\log n)$ rounds *without descending down the rank decomposition*. Otherwise, Spoiler only needs a constant number of rounds to descend to a child node in the rank decomposition. In either case, we only need $O(\log n)$ rounds total, which yields a TC^1 isomorphism test.

In the process of our work, we came across a result of Bodlaender [5], who showed that any graph of treewidth k admits a *binary* tree decomposition of width $\leq 3k + 2$ and height $O(\log n)$. Using Bodlaender's result, we were able to modestly improve the descriptive complexity for graphs of bounded treewidth.

▶ **Theorem 5.** *The $(3k + 6)$-dimensional Weisfeiler–Leman algorithm identifies graphs of treewidth k in $O(\log n)$ rounds.*

In light of the above theorem, we obtain the following improvement in the descriptive complexity for graphs of bounded treewidth.

▶ **Corollary 6.** *Let G be a graph of treewidth k. Then there exists a formula $\varphi_G \in \mathcal{C}_{3k+7,O(\log n)}$ that identifies G up to isomorphism. That is, for any $H \ncong G$, $G \models \varphi_G$ and $H \not\models \varphi_G$.*

2 Preliminaries

2.1 Weisfeiler–Leman

We begin by recalling the Weisfeiler–Leman algorithm for graphs, which computes an isomorphism-invariant coloring. Let G be a graph on n vertices, let $\chi : V(G) \to [n]$ be a coloring of the vertices, and let $k \geq 2$ be an integer. The k-dimensional Weisfeiler–Leman, or k-WL, algorithm begins by constructing an initial coloring $\chi_0 : V(G)^k \to \mathcal{K}$, where $\mathcal{K}$ is our set of colors, by assigning each k-tuple a color based on its isomorphism type under the coloring χ.[2] Two k-tuples $(v_1, \ldots, v_k) \in V(G)^k$ and $(u_1, \ldots, u_k) \in V(G)^k$ receive the same color under χ_0 if and only if the following conditions all hold

- For all i, j, $v_i = v_j \Leftrightarrow u_i = u_j$.
- The map $v_i \mapsto u_i$ (for all $i \in [k]$) is an isomorphism of the induced subgraphs $G[\{v_1, \ldots, v_k\}]$ and $G[\{u_1, \ldots, u_k\}]$
- $\chi(u_i) = \chi(v_i)$ for all $i \in [k]$.

For $r \geq 0$, the coloring computed at the rth iteration of Weisfeiler–Leman is refined as follows. For a k-tuple $\overline{v} = (v_1, \ldots, v_k)$ and a vertex $x \in V(G)$, define

$$\overline{v}(v_i/x) = (v_1, \ldots, v_{i-1}, x, v_{i+1}, \ldots, v_k).$$

The coloring computed at the $(r + 1)$st iteration, denoted χ_{r+1}, stores the color of the given k-tuple $\overline{v}$ at the rth iteration, as well as the colors under χ_r of the k-tuples obtained by substituting a single vertex in $\overline{v}$ for another vertex x. We examine this multiset of colors over all such vertices x. This is formalized as follows:

$$\chi_{r+1}(\overline{v}) =(\chi_r(\overline{v}), \{\!\{(\chi_r(\overline{v}(v_1/x)), \ldots, \chi_r(\overline{v}(v_k/x))\big| x \in V(G)\}\!\}),$$

[2] Note that for k-WL applied to two graphs G and H, each of order n, there are at most $2n^k$ color classes. So without loss of generality, we may take $\mathcal{K} = [2n^k]$.

where $\{\!\{\cdot\}\!\}$ denotes a multiset. Note that the coloring χ_r computed at iteration r induces a partition of $V(G)^k$ into color classes. The Weisfeiler–Leman algorithm terminates when this partition is not refined, that is, when the partition induced by χ_{r+1} is identical to that induced by χ_r. The final coloring is referred to as the *stable coloring*, which we denote $\chi_\infty := \chi_r$.

The 1-dimensional Weisfeiler–Leman algorithm, sometimes referred to as *Color Refinement*, works nearly identically. The initial coloring is that provided by the vertex coloring for the input graph. For the refinement step, we have that: $\chi_{r+1}(u) = (\chi_r(u), \{\!\{\chi_r(v) : v \in N(u)\}\!\})$. We have that 1-WL terminates when the partition on the vertices is not refined.

As we are interested in both the Weisfeiler–Leman dimension and the number of rounds, we will use the following notation.

▶ **Definition 7.** *Let $k \geq 1$ and $r \geq 1$ be integers. The (k, r)-WL algorithm is obtained by running k-WL for r rounds. Here, the initial coloring counts as the first round.*

Let S be a sequence of vertices. The *individualize-and-refine* paradigm works first by assigning each vertex in S a unique color. We then run (k, r)-WL starting from this choice of initial coloring. We denote the coloring computed by (k, r)-WL after individualizing S as $\chi^S_{k,r}$. When there is ambiguity about the graph G in question, we will for clarity write $\chi^{S,G}_{k,r}$.

For two graphs G and H, we say that (k, r)-WL *distinguishes* G and H if there is some color c such that: $|\{v \in V(G)^k : \chi_{G,k,r}(v) = c\}| \neq |\{w \in V(H)^k : \chi_{H,k,r}(w) = c\}|$. Additionally, (k, r)-WL *identifies* a graph G if (k, r)-WL distinguishes G from every graph H such that $G \ncong H$.

▶ Remark 8. Grohe & Verbitsky [25] previously showed that for fixed k, the classical k-dimensional Weisfeiler–Leman algorithm for graphs can be effectively parallelized. Precisely, each iteration (including the initial coloring) can be implemented using a logspace uniform TC^0 circuit.

2.2 Pebbling Game

We recall the bijective pebble game introduced by [26, 27] for WL on graphs. This game is often used to show that two graphs X and Y cannot be distinguished by k-WL. The game is an Ehrenfeucht–Fraïssé game (c.f., [13, 37]), with two players: Spoiler and Duplicator. We begin with $k + 1$ pairs of pebbles. Prior to the start of the game, each pebble pair (p_i, p'_i) is initially placed either beside the graphs or on a given pair of vertices $v_i \mapsto v'_i$ (where $v_i \in V(X), v'_i \in V(Y)$). We refer to this initial configuration for X as $\overline{v}$, and this initial configuration for Y as $\overline{v'}$. Each round r proceeds as follows.

1. Spoiler picks up a pair of pebbles (p_i, p'_i).
2. Duplicator chooses a bijection $f_r : V(X) \to V(Y)$ (we emphasize that the bijection chosen depends on the round and, implicitly, the pebbling configuration at the start of said round).
3. Spoiler places p_i on some vertex $v \in V(X)$. Then p'_i is placed on $f(v)$.

Let $v_1, \ldots, v_m$ be the vertices of X pebbled at the end of round r of the game, and let $v'_1, \ldots, v'_m$ be the corresponding pebbled vertices of Y. Spoiler wins precisely if the map $v_\ell \mapsto v'_\ell$ is not an isomorphism of the induced subgraphs $X[\{v_1, \ldots, v_m\}]$ and $Y[\{v'_1, \ldots, v'_m\}]$. Duplicator wins otherwise. Spoiler wins, by definition, at round 0 if X and Y do not have the same number of vertices. We note that $\overline{v}$ and $\overline{v'}$ are not distinguished by the first r rounds of k-WL if and only if Duplicator wins the first r rounds of the $(k + 1)$-pebble game [26, 27, 7].

We establish a helper lemma, which effectively states that Duplicator must respect connected components of pebbled vertices.

▶ **Lemma 9.** *Let G, H be graphs on n vertices. Suppose that $(u, v) \mapsto (u', v')$ have been pebbled. Furthermore, suppose that u, v belong to the same connected component of G, while u', v' belong to different connected components of H. Then Spoiler can win using 1 additional pebble and $O(\log n)$ rounds.*

2.3 Logics

We recall key notions of first-order logic. We have a countable set of variables $\{x_1, x_2, \ldots\}$. Formulas are defined inductively. For the basis, we have that $x_i = x_j$ is a formula for all pairs of variables. Now if φ_1, φ_2 are formulas, then so are the following: $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \neg\varphi_1, \exists x_i\, \varphi_1$, and $\forall x_i\, \varphi_1$. In order to define logics on graphs, we add a relation $E(x, y)$, where $E(x, y) = 1$ if and only if $\{x, y\}$ is an edge of our graph, and 0 otherwise. In keeping with the conventions of [7], we refer to the first-order logic with relation E as $\mathcal{L}$ and its k-variable fragment as $\mathcal{L}_k$. We refer to the logic $\mathcal{C}$ as the logic obtained by adding counting quantifiers $\exists^{\geq n} x\, \varphi$ (there exist at least n elements x that satisfy φ) and $\exists! n\, x\, \varphi$ (there exist exactly n elements x that satisfy φ) and its k-variable fragment as $\mathcal{C}_k$.

The *quantifier depth* of a formula φ (belonging to either $\mathcal{L}$ or $\mathcal{C}$) is the depth of its quantifier nesting. We denote the quantifier depth of φ as $\text{qd}(\varphi)$ This is defined inductively as follows.

- If φ is atomic, then $\text{qd}(\varphi) = 0$.
- $\text{qd}(\neg\varphi) = \text{qd}(\varphi)$.
- $\text{qd}(\varphi_1 \vee \varphi_2) = \text{qd}(\varphi_1 \wedge \varphi_2) = \max\{\text{qd}(\varphi_1), \text{qd}(\varphi_2)\}$.
- $\text{qd}(Qx\, \varphi) = \text{qd}(\varphi) + 1$, where Q is a quantifier in the logic.

We denote the fragment of $\mathcal{L}_k$ (respectively, $\mathcal{C}_k$) where the formulas have quantifier depth at most r as $\mathcal{L}_{k,r}$ (respectively, $\mathcal{C}_{k,r}$). Let $\overline{v} \in V(X)^k, \overline{v'} \in V(Y)^k$. We note that $\overline{v}, \overline{v'}$ are distinguished by (k, r)-WL if and only if there exists a formula $\varphi \in \mathcal{C}_{k+1,r}$ such that $(X, \overline{v}) \models \varphi$ and $(Y, \overline{v'}) \not\models \varphi$ [29, 7].

2.4 Rank-Width

Oum & Seymour [28] introduced the rank-width parameter to measure the width of a certain hierarchical decomposition of graphs. The goal is to intuitively split the vertices of a graph along cuts of low complexity in a hierarchical fashion. Here, the complexity is the $\mathbb{F}_2$-rank of the matrix capturing the adjacencies crossing the cut.

Precisely, let G be a graph, and let $X, Y \subseteq V(G)$. Define $M(X, Y) \in \mathbb{F}_2^{X \times Y}$ to be the matrix where $(M(X, Y))_{uv} = 1$ if and only if $uv \in E(G)$. That is, $M(X, Y)$ is the submatrix of the adjacency matrix whose rows are indexed by X and whose columns are indexed by Y. Denote $\rho(X) := \text{rk}_{\mathbb{F}_2}(M(X, \overline{X}))$.

A *rank decomposition* of G is a tuple (T, γ), where T is a rooted binary tree and $\gamma : V(T) \to 2^{V(G)}$ satisfying the following:

- For the root r of T, $\gamma(r) = V(G)$,
- For an internal node $t \in V(T)$, denote the children of t as s_1, s_2. For every internal node t, we have that $\gamma(t) = \gamma(s_1) \cup \gamma(s_2)$, and $\gamma(s_1) \cap \gamma(s_2) = \emptyset$.
- For any leaf $t \in V(T)$, $|\gamma(t)| = 1$.

▶ Remark 10. Let $L(T)$ be the set of leaves of T. Instead of providing γ, we can equivalently define a bijection $f : V(G) \to L(T)$. By the second condition of a rank decomposition, f completely determines γ.

The *width* of a rank decomposition (T, γ) is: $\text{wd}(T, \gamma) := \max\{\rho_G(\gamma(t)) : t \in V(T)\}$. The *rank-width* of a graph G is: $\text{rw}(G) := \min\{\text{wd}(T, \gamma) : (T, \gamma) \text{ is a rank decomposition of } G\}$.

The parameter rank-width is closely related to the parameter clique width, introduced by Courcelle & Olariu [9]. Oum & Seymour [28] showed that: $\text{rw}(G) \leq \text{cw}(G) \leq 2^{\text{rw}(G)+1} - 1$. Denote $\text{tw}(G)$ to be the treewidth of G. Oum [41] showed that $\text{rw}(G) \leq \text{tw}(G) + 1$. Note that $\text{tw}(G)$ cannot be bounded in terms of $\text{rw}(G)$; for instance, the complete graph K_n has $\text{rw}(K_n) = 1$ but $\text{tw}(K_n) = n - 1$.

3 Weisfeiler–Leman for Graphs of Bounded Rank-Width

3.1 Split Pairs and Flip Functions

In designing a pebbling strategy for graphs of bounded rank-width, Grohe & Neuen [22] sought to pebble a set of vertices $X \subseteq V(G)$ such that $\rho(X) \leq k$ and pebbling X partitions the remaining vertices into sets $C_1, \ldots, C_\ell$ that can be treated independently. Furthermore, we want for each $i \in [\ell]$ that either $C_i \subseteq X$ or $C_i \subseteq \overline{X}$. As there can be many edges between X and $\overline{X}$, this is hard to accomplish in general. To this end, Grohe & Neuen [22] utilized split pairs and flip functions. We will now recall their framework.

Let $G(V, E, \chi)$ be a colored graph on n vertices, and suppose the rank-width of G is at most k. Let $X \subseteq V(G)$. For $v \in X$, define $\text{vec}_X(v) = (a_{v,w})_{w \in \overline{X}} \in \mathbb{F}_2^{\overline{X}}$, where $a_{v,w} = 1$ if and only if $vw \in E(G)$. For $S \subseteq X$, define $\text{vec}_X(S) = \{\text{vec}_X(v) : v \in S\}$. A *split pair* for X is a pair (A, B) such that:

(a) $A \subseteq X$, and $B \subseteq \overline{X}$,

(b) $\text{vec}_X(A)$ forms a linear basis for $\langle \text{vec}_X(X) \rangle$, and

(c) $\text{vec}_{\overline{X}}(B)$ forms a linear basis for $\langle \text{vec}_{\overline{X}}(\overline{X}) \rangle$.

An *ordered split pair* for X is a pair $((a_1, \ldots, a_q), (b_1, \ldots, b_p))$ such that $(\{a_1, \ldots, a_q\}, \{b_1, \ldots, b_p\})$ is a split pair for X.

Let $G(V, E, \chi)$ be a colored graph on n vertices, and suppose the rank-width of G is at most k. An *ordered split pair*, $(\overline{a}, \overline{b})$, of order at most $2k$ is a pair $(\overline{a}, \overline{b})$, where $\overline{a}, \overline{b} \in V(G)^{\leq 2k}$. For $v, w \in V(G)$, we say that $v \approx_{(\overline{a}, \overline{b})} w$ if $N(v) \cap (\overline{a}, \overline{b}) = N(w) \cap (\overline{a}, \overline{b})$ (here, we consider $N(v) \cap (\overline{a}, \overline{b})$ as a set). Observe that $\approx_{(\overline{a}, \overline{b})}$ forms an equivalence relation. For $(\overline{a}, \overline{b}) \in V(G)^{\leq 2k}$, let $2^{\overline{a} \cup \overline{b}}$ be the set of all subsets of $\overline{a} \cup \overline{b} \subseteq V(G)$, where we abuse notation by considering $\overline{a}, \overline{b}$ as subsets of $V(G)$. A *flip extension* of an ordered split pair $(\overline{a}, \overline{b})$ is a tuple:

$$\overline{s} := \left(\overline{a}, \overline{b}, f : \left(2^{\overline{a} \cup \overline{b}} \right)^2 \to [n] \cup \{\perp\} \right),$$

such that for all $M, N \in 2^{\overline{a} \cup \overline{b}}$ with $M \neq N$, either $f(M, N) = \perp$ or $f(N, M) = \perp$. There is no restriction on $f(M, N)$ if $M = N$. For $v, w \in V(G)$, we say that $v \approx_{\overline{s}} w$ if $v \approx_{(\overline{a}, \overline{b})} w$. Denote $[v]_{\approx_{\overline{s}}}$ to be the equivalence class of v with respect to $\approx_{\overline{s}}$. Define the *flipped graph* $G^{\overline{s}} = (V, E^{\overline{s}}, \chi, \overline{a}, \overline{b})$, where $V(G^{\overline{s}}) = V(G)$,

$$\begin{aligned} E^{\overline{s}} := &\{vw \in E(G) : f(N(v) \cap (\overline{a}, \overline{b}), N(w) \cap (\overline{a}, \overline{b})) = d \in [n] \wedge |N(v) \cap [w]_{\approx_{\overline{s}}}| < d\} \\ &\cup \{vw \notin E(G) : f(N(v) \cap (\overline{a}, \overline{b}), N(w) \cap (\overline{a}, \overline{b})) = d \in [n] \wedge |N(v) \cap [w]_{\approx_{\overline{s}}}| \geq d\}, \end{aligned}$$

and χ is the same coloring as in G. Denote $\text{Comp}(G, \overline{s}) \subseteq 2^{V(G)}$ be the set of vertex sets of the connected components of $G^{\overline{s}}$. Observe that $\text{Comp}(G, \overline{s})$ forms a partition of $V(G)$. Grohe & Neuen [22] established that for any choice $(\overline{a}, \overline{b})$ of split pair, there exists a suitable flip function; and thus, a suitable flip extension.

▶ **Lemma 11** ([22, Lem. 5.6]). *Let G be a (colored) graph, and let $X \subseteq V(G)$. Furthermore, let $(\overline{a}, \overline{b})$ be an ordered split pair for X. Then there exists a flip extension $\overline{s} := (\overline{a}, \overline{b}, f)$ such that $C \subseteq X$ or $C \subseteq \overline{X}$ for every $C \in Comp(G, \overline{s})$.*

Grohe & Neuen [22, Sec. 5] considered uncolored flipped graphs. As the conditions for determining the edges of the flipped graph do not depend on the vertex colors, [22, Lem. 5.6] holds in our setting.

We now turn to showing that the flip extensions preserve both isomorphism and the effects of Weisfeiler–Leman. To do so, we consider vertex colorings χ that refine the coloring $\chi_{1,3}$ computed by $(1,3)$-WL. The advantage of incorporating such a coloring on the vertices, is that it encodes some data about how the vertices of G interact with the specified split pair. Furthermore, the colorings computed by Weisfeiler–Leman are invariant under isomorphism. We take advantage of this to establish that the flipped graph preserves both the isomorphism problem (Lem. 12) and the effects of Weisfeiler–Leman (Lem. 13). For a graph G of rank-width k, we will be running $(6k+3, O(\log n))$, and so we may assume without loss of generality that the vertices of G have been colored according to $(1,3)$-WL.

▶ **Lemma 12.** *Let G, H be graphs, and let $\overline{s} = (\overline{a}, \overline{b}, f), \overline{s'} = (\overline{a'}, \overline{b'}, f)$ be flip extensions for G, H, respectively (we stress that the function f appearing in $\overline{s}$ is the same as that appearing in $\overline{s'}$). Let $k \geq 1, r \geq 3$. Consider the colorings $\chi_{k,r}^{(\overline{a},\overline{b}),G}, \chi_{k,r}^{(\overline{a'},\overline{b'}),H}$ obtained by individualizing $(\overline{a}, \overline{b}) \mapsto (\overline{a'}, \overline{b'})$ and applying (k,r)-WL.*

Let $\varphi : V(G) \to V(H)$ be a bijection. We have that φ is an isomorphism of the colored graphs $(G, \chi_{k,r}^{(\overline{a},\overline{b}),G}) \cong (H, \chi_{k,r}^{(\overline{a'},\overline{b'}),H})$ if and only if φ is an isomorphism of $G^{\overline{s}} \cong H^{\overline{s'}}$.

▶ **Lemma 13** (cf. [22, Lem. 3.10]). *Let $G(V, E, \chi), G'(V', E', \chi')$ be colored graphs, and let $\overline{s} = (\overline{a}, \overline{b}, f)$ and $\overline{s'} = (\overline{a'}, \overline{b'}, f)$ be flip extensions (we are using the same flip function f for both $\overline{s}, \overline{s'}$). Let $\chi_{1,3}$ be the coloring resulting from individualizing $(\overline{a}, \overline{b}) \mapsto (\overline{a'}, \overline{b'})$ and running $(1,3)$-WL. Suppose that χ, χ' both refine $\chi_{1,3}$. Let $((\overline{v}, \overline{w})) = ((v_1, \ldots, v_\ell), (w_1, \ldots, w_\ell))$ be a position in the ℓ-pebble bijective pebble game. We have that Spoiler wins from $((\overline{v}, \overline{w}))$ in the ℓ-pebble, r-round game on (G, G') if and only if Spoiler wins from from $((\overline{v}, \overline{w}))$ in the ℓ-pebble, r-round game on $(G^{\overline{s}}, (G')^{\overline{s}})$.*

▶ **Corollary 14** (Compare rounds cf. [22, Corollary 3.12]). *Let $G(V, E, \chi), G'(V', E', \chi')$ be colored graphs, and let $\overline{s} = (\overline{a}, \overline{b}, f)$ and $\overline{s'} = (\overline{a'}, \overline{b'}, f)$ be flip extensions (we are using the same flip function f for both $\overline{s}, \overline{s'}$). Let $\chi_{1,3}$ be the coloring resulting from individualizing $(\overline{a}, \overline{b}) \mapsto (\overline{a'}, \overline{b'})$ and running $(1,3)$-WL. Suppose that χ, χ' both refine $\chi_{1,3}$.*

Let $\overline{v} \in V^k, \overline{v'} \in (V')^k$. Let C be a connected component of $G^{\overline{s}}$ such that $\chi(u) \neq \chi(w)$ for all $u \in C$ and all $w \in V \setminus C$. Let C' be a connected component of $(G')^{\overline{s'}}$ such that $\chi'(u') \neq \chi'(w')$ for all $u' \in C'$ and $w' \in V' \setminus C'$. Let $r \geq 1$. Suppose that: $(G[C], \chi_{1,r}^{\overline{v},G}) \not\cong (G'[C'], \chi_{1,r}^{\overline{v'},G'})$. Let $\overline{w} := C \cap \overline{v}$ and $\overline{w'} := C' \cap \overline{v'}$. Then either: $(G[C], \chi_{1,r}^{\overline{w},G}) \not\cong (G'[C'], \chi_{1,r}^{\overline{w'},G'})$, or r rounds of Color Refinement distinguishes $(G, \chi^{\overline{v}})$ from $(G', (\chi')^{\overline{v'}})$.

3.2 WL for Graphs of Bounded Rank-Width

Our goal in this section is to establish the following.

▶ **Theorem 15.** *Let G be a graph on n vertices of rank-width k, and let H be an arbitrary graph such that $G \not\cong H$. We have that the $(6k+3, O(\log n))$-WL algorithm will distinguish G from H.*

▶ **Definition 16** ([22, Definition 4.1]). *Let G be a graph, and let $X, X_1, X_2 \subseteq V(G)$ such that $X = X_1 \sqcup X_2$. Let (A, B) be a split pair for X, and let (A_i, B_i) $(i = 1, 2)$ be a split pair for X_i. We say that (A_i, B_i) are nice with respect to (A, B) if the following conditions hold:*
(a) *$A \cap X_i \subseteq A_i$ for each $i \in \{1, 2\}$, and*
(b) *$B_2 \cap X_1 \subseteq A_1$ and similarly $B_1 \cap X_2 \subseteq A_2$.*
A triple $((A, B), (A_1, B_1), (A_2, B_2))$ of ordered split pairs is nice if the underlying triple of unordered split pairs is nice.

▶ **Lemma 17** ([22, Lem. 4.2]). [3] *Let G be a graph, and let $X, X_1, X_2 \subseteq V(G)$ such that $X = X_1 \sqcup X_2$. Let (A, B) be a split pair for X. There exist nice split pairs (A_i, B_i) for X_i $(i = 1, 2)$ such that additionally $B_i \cap \overline{X_i} \subseteq B$.*

▶ **Definition 18.** *Let G be a graph. A component partition of G is a partition $\mathcal{P}$ of $V(G)$ such that every connected component appears in exactly one block of $\mathcal{P}$. That is, for every connected component C of G, there exists a $P \in \mathcal{P}$ such that $C \subseteq P$.*

▶ **Lemma 19** ([22, Observation 4.3]). *Let G, H be two non-isomorphic graphs, and let $\mathcal{P}, \mathcal{Q}$ be component partitions of G, H respectively. Let $\sigma : V(G) \to V(H)$ be a bijection. There exists a vertex v of G such that $G[P] \not\cong H[Q]$, where $P \in \mathcal{P}$ is the unique set containing v and $Q \in \mathcal{Q}$ is the unique set containing $\sigma(v)$.*

We now prove Thm. 15.

Proof Idea of Thm. 15 . We follow the strategy of [22, Thm. 4.4]. We will briefly discuss the how we modified the proof from [22]; the full proof will appear in the full version. Let $G(V, E, \chi_G)$ be a colored graph of rank width $\leq k$, and let H be an arbitrary graph such that $G \not\cong H$. By [8, Thm. 5], G admits a rank decomposition (T, γ) of width at most $2k$ where T has height at most $3 \cdot (\log(n) + 1)$.

We will show that Spoiler has a winning strategy in the $6k + 3$ pebble game in $O(\log n)$ rounds. In a similar manner as in the proof of [22, Thm. 4.4], we will first argue that $12k + 5$ pebbles suffice, and then show how to improve the bound to use only $6k + 3$ pebbles.

Spoiler's strategy is to play along the rank decomposition (T, γ) starting from the root. As Spoiler proceeds down the tree, the non-isomorphism is confined to increasingly smaller parts of G and H. At a node $t \in V(T)$, Spoiler pebbles a split pair $(\overline{a}, \overline{b}) \mapsto (\overline{a'}, \overline{b'})$, where $(\overline{a}, \overline{b})$ corresponds to a flip extension $\overline{s} = (\overline{a}, \overline{b}, f)$ of $X = \gamma(t)$. Let $\overline{s'} := (\overline{a'}, \overline{b'}, f)$. Now to confine the non-isomorphism, Spoiler identifies, after individualizing the split pair and performing three steps of Color Refinement- the initial coloring and two refinement steps, a pair of non-isomorphic components $C \subseteq X, C' \subseteq V(H)$ in the flipped graphs $G^{\overline{s}}$ and $H^{\overline{s'}}$. In particular, Spoiler seeks to find such components C and C' such that C is increasingly

[3] Grohe & Neuen use in the proof of [22, Thm. 5.5] that [22, Lem. 4.2] holds for the flipped graphs they define in Section 5, and not just the earlier notion of flipped graphs they consider in Section 3. Hence, [22, Lem. 4.2] holds in our setting as well.

further from the root of T. Once Spoiler reaches a leaf node of T, Spoiler can quickly win. Spoiler places a pebble on a vertex in C and its image in C', under Duplicator's bijection at the given round.

We note that the three rounds of Color Refinement suffice for WL to detect the partitioning induced by the flip function, though it is not sufficiently powerful to detect the connected components of $G^{\overline{s}}$ and $H^{\overline{s'}}$. In the argument below, we will technically consider graphs where the refinement step uses $(2, O(\log n))$-WL. This ensures that after individualizing a vertex on a given component C, that the vertices of C receive different colors than those of $V(G) \setminus C$. This will eventually happen, and so in the pebble game characterization, we can continue to descend along T as if the vertices of C have been distinguished from $V(G) \setminus C$. This is a key point where our strategy deviates from that of [22, Thm. 4.4]. The remaining details will appear in the full version. ◀

4 Canonical Forms in Parallel

In this section, we will establish the following.

▶ **Theorem 20.** *Let G be a graph on n vertices, of rank-width k. We can compute a canonical labeling for G using a TC circuit of depth $O(\log^2 n)$ and size $n^{O(16^k)}$.*

We will prove Thm. 20 via the individualization-and-refinement paradigm. Our strategy is similar to that of Köbler & Verbitsky [34], who established the analogous result for treewidth. We will begin by briefly recalling their approach. Köbler & Verbitsky began by enumerating ordered sequences of vertices of length $\leq k+1$, testing whether each such sequence disconnected the graph. In particular, Köbler & Verbitsky crucially used the fact that a graph of treewidth k admits a so-called *balanced* separator S of size $\leq k+1$, which splits G into connected components each of size $\leq n/2$. Köbler & Verbitsky then colored the vertices of each connected component of $G - S$ according to how they connected back to S. As graphs of bounded treewidth are hereditary (closed under taking induced subgraphs), Köbler & Verbitsky were then able to recurse on the connected components. The existence of balanced separators guarantees that only $O(\log n)$ such recursive calls are needed.

Instead of relying on balanced separators, it is sufficient to guarantee that after $O(\log n)$ recursive calls, each connected component will be a singleton. To this end, we again leverage the result of [8], who showed that a graph of rank-width k admits a rank decomposition (T, γ) of width $\leq 2k$ and height $O(\log n)$.

Thus, we would intuitively like to descend along such a rank decomposition (T, γ) of width $\leq 2k$ and height $O(\log n)$. Fix a node $t \in V(T)$, and let t_1 be the left child and t_2 be the right child of t. We would then enumerate over all pairs of flip extensions $((\overline{a_1}, \overline{b_1}, f_1), (\overline{a_2}, \overline{b_2}, f_2))$, where intuitively $\overline{s_i} := (\overline{a_i}, \overline{b_i}, f_i)$ is a flip extension for $\gamma(t_i)$. Then for each $i = 1, 2$ and each component $C_i \in \text{Comp}(G[\gamma(t)], \overline{s_i})$, we apply the construction recursively. Note that we are not able to efficiently compute a rank decomposition of width $\leq 2k$ and height $O(\log n)$. Nonetheless, Lem. 11 guarantees the existence of flip extensions that witness the decomposition of a fixed rank decomposition (T, γ). Following an idea of Wagner [46], we consider all possible flip extensions in parallel, and thus ensure that the flip extension which respects a fixed rank decomposition is considered by the algorithm. As we will show in Lem. 24, the existence of a rank decomposition of height $O(\log n)$ allows us to guarantee that at least one of the flip extensions considered by the algorithm will produce a labeling, and Lem. 25 will then guarantee that the minimum such labeling (which is the labeling the algorithm will return) is in fact canonical. Now to the details.

We first show that we can enumerate the split pairs in a canonical manner. To this end, we will need the following lemma, which is essentially well-known amongst those working on the Weisfeiler–Leman algorithm (cf., [29, 22]).

Let G be a graph. The (k, r)-Weisfeiler-Leman algorithm *determines orbits of ℓ-tuples* if, for every graph H, every $v \in V(G)^{\ell}$ and every $w \in V(H)^{\ell}$ such that $\chi_{k,r}(v) = \chi_{k,r}(w)$, there is an isomorphism $\varphi : V(G) \to V(H)$ such that $\varphi(v) = w$.

▶ **Lemma 21.** *Let $\mathcal{C}$ be a class of graphs such that (k, r)-WL identifies all (colored) $G \in \mathcal{C}$. Then for any $\ell \geq 1$ and all (colored) $G \in \mathcal{C}$, $(k + \ell, r)$-WL determines the orbits of all ℓ-tuples of vertices in G.*

By Thm. 2, we have that $(6k + 3, O(\log n))$-WL identifies all graphs of rank-width k. As we will need to enumerate split pairs, which have length $\leq 4k$, we will run $(10k + 3, O(\log n))$-WL at each stage. Lem. 21 ensures that enumerating the split pairs in color class order is canonical. Note that a flip function is represented as a tuple in $\{0, \ldots, n\}^{2^{4k}}$. So for a fixed split pair $(\overline{a}, \overline{b})$, we can canonically enumerate the flip functions in lexicographic order. Thus, flip extensions can be enumerated in a canonical order.

▶ Remark 22. Now let $(\overline{a}, \overline{b})$ be a split pair on G and $(\overline{c}, \overline{d})$ be a split pair on H such that $\chi_{10k+3,O(\log n)}((\overline{a}, \overline{b})) = \chi_{10k+3,O(\log n)}((\overline{c}, \overline{d}))$. Let f be a given flip function, and let $\overline{s} = (\overline{a}, \overline{b}, f), \overline{s'} = (\overline{c}, \overline{d}, f)$ be flip extensions. By Lem. 21, there is an isomorphism mapping $(\overline{a}, \overline{b}) \mapsto (\overline{c}, \overline{d})$. Hence, Lem. 12 provides that the flipped graphs $G^{\overline{s}}, H^{\overline{s'}}$ are isomorphic whenever $G \cong H$. In particular, if there is an isomorphism $\varphi : G \cong H$ mapping $(\overline{a}, \overline{b}) \mapsto (\overline{c}, \overline{d})$, then φ is also an isomorphism of $G^{\overline{s}} \cong H^{\overline{s}'}$.

▶ **Lemma 23.** *Let G be a graph, $X \subseteq V(G)$, and $\overline{s} = (\overline{a}, \overline{b}, f)$ be a flip extension for $G[X]$. We may write down the flipped graph $G^{\overline{s}}$ and identify the connected components of $G[X]^{\overline{s}}$ in L.*

We will now pause to outline the procedure for the reader. Let $\overline{s} := (\overline{a}, \overline{b}, f)$ be a flip extension for $V(G)$. We will first individualize $(\overline{a}, \overline{b})$ and apply $(10k + 3, O(\log n))$-WL to G. For each component $C \in \text{Comp}(G, \overline{s})$, this will encode the isomorphism class of $G[C]$ (as $(6k + 3, O(\log n))$-WL identifies all graphs of rank-width $\leq k$– see Thm. 15), as well as how $G[C]$ connects back to the rest of G. It is easy to see that for any two vertices v, w, if v, w receive the same color under $\chi^{(\overline{a},\overline{b})}_{10k+3,O(\log n)}$, then the following conditions hold:

(a) $N(v) \cap (\overline{a} \cup \overline{b}) = N(w) \cap (\overline{a} \cup \overline{b})$, and
(b) For any vertex u, $|N(v) \cap [u]_{\approx_{\overline{s}}}| = |N(w) \cap [u]_{\approx_{\overline{s}}}|$.

Intuitively, this coloring encodes how each given vertex connects to the rest of G. Precisely, let $G \cong H$ be graphs of rank-width $\leq k$, and suppose that the algorithm returns the labeling $\lambda : V(G) \to [n]$ for G and labeling $\kappa : V(H) \to [n]$ for H (where $n = |G| = |H|$). If $v, w \in V(G)$ belong to different components of $\text{Comp}(G, \overline{s})$, then we need to ensure that $\{v, w\} \in E(G)$ if and only if $\{(\kappa^{-1} \circ \lambda)(v), (\kappa^{-1} \circ \lambda)(w)\} \in E(H)$. By the definition of the flipped graph (Sec. 3), conditions (a) and (b) determine precisely whether $\{v, w\} \in E(G)$.

By Lem. 23, we may write down the connected components for the flipped graph $G^{\overline{s}}$ in L. We will then sort these connected components in lexicographic order by color class, which is L-computable. It may be the case that for two connected components $C_i, C_j \in \text{Comp}(G, \overline{s})$, $G[C_i]$ and $G[C_j]$ are isomorphic and connect to the rest of G in the same way, and so receive the same multiset of colors. In this case, we may arbitrarily choose whether $G[C_i]$ will be sorted before $G[C_j]$. The output will not depend on this particular choice, as there is an automorphism of G which exchanges the two components. Now for each $C \in \text{Comp}(G, \overline{s})$, we will apply the procedure recursively on $G[C]$, incrementing the local depth variable by 1. If for each connected component of $\text{Comp}(G, \overline{s})$ we are given a valid labeling, we may recover a labeling for G as follows. Let $C_j \in \mathrm{Comp}(G, \overline{s})$, with the labeling function $\ell_j : V(C_j) \to \{1, \ldots, |V(C_j)|\}$ returned by applying our canonization procedure recursively to $G[C_j]$. Let $h_j := |C_1| + \cdots + |C_{j-1}|$. We will recover a canonical labeling $\ell : V(G) \to [n]$ by, for each such j and $v \in C_j$, setting $\ell(v) := \ell_j(v) + h_j$. As each vertex of G appears in exactly one C_j, ℓ is well-defined.

We stress here again that the recursive calls to the canonization procedure track the depth to ensure that we do not make $\geq 3 \cdot (\log(n) + 1)$ recursive calls. If the depth parameter is ever larger than $3 \cdot (\log(n) + 1)$, then the algorithm returns $\perp$ to indicate an error. In the recombine stage of our divide and conquer procedure, if any of the labelings returned for the components of $\mathrm{Comp}(G, \overline{s})$ are $\perp$, then the algorithm simply returns $\perp$. Thus, a priori, our algorithm may not return a labeling of the vertices. We will prove later (see Lem. 24) that our algorithm actually does return a labeling.

We now give a more precise description of our algorithm and proceed to prove its correctness. We define a canonical labeling $\mathrm{Can}(G)$ of a graph, via a subroutine $\mathrm{Can}(G, d)$. The subroutine $\mathrm{Can}(G, d)$ takes an n-vertex graph G and a depth parameter d, and outputs either a bijection $\lambda : V(G) \to [n]$ or a failure symbol $\perp$. In pseudocode, our canonical labeling subroutine works as follows:

Algorithm 1 **Can**(G, d).

```
Input: A colored graph $G = (V, E, \chi)$ of rank-width $\leq k$, and a parameter $d$ for depth.
1. If $d > 3 \cdot (\log(n) + 1)$, return $\perp$.
2. If $d \leq 3 \cdot (\log(n) + 1)$ and $|V| = 1$, return $\lambda(v) = 1$.
3. Otherwise, if $d \leq 3 \cdot (\log(n) + 1)$ and $|V| > 1$, do the following steps:
4. Run $(10k + 3, O(\log n))$-WL on $G$.
5. In parallel, enumerate all possible flip extensions $\overline{s} = (\overline{a}, \overline{b}, f)$ in lexicographic order, where
   the order on $(\overline{a}, \overline{b})$ is considered with respect to the ordering induced by the coloring
   $\chi_{10k+3,O(\log n)}$ (by [25], the colors are represented by numbers, and so color class order is
   well-defined).
6. For each flip extension, $\overline{s} = (\overline{a}, \overline{b}, f)$,
   a. Compute the coloring $\chi^{(\overline{a},\overline{b})}_{10k+3,O(\log(n))}$ applied to $G$.
   b. Construct the flipped graph $G^{\overline{s}}$.
   c. Compute the set of connected components $\mathrm{Comp}(G, \overline{s})$. If $G^{\overline{s}}$ is connected, then return
      $\perp$. Note that there exists a rank decomposition $(T, \gamma)$ in which for all $u, v \in V(T)$,
      $\gamma(u) \neq \gamma(v)$. So there exists a flip extension $\overline{s}$ that splits $G^{\overline{s}}$ into at least two connected
      components.
   d. Order the components $C \in \mathrm{Comp}(G, f)$ by lexicographic ordering of the multiset of
      colors $\chi^{\overline{a},\overline{b}}_{10k+3,O(\log(n))}(G[C])$. Let $C_1, \ldots, C_\ell$ be the components in this ordering.
   e. Compute $\mathrm{Can}(d+1, G[C_1]), \ldots, \mathrm{Can}(d+1, G[C_\ell])$ and let $\lambda_{\overline{s},1}, \ldots, \lambda_{\overline{s},\ell}$ be the resulting
      labelings.
   f. If $\lambda_{\overline{s},i} = \perp$ for any $i \in [\ell]$ set $\lambda_{\overline{s}} = \perp$. Otherwise, if $\lambda_{\overline{s},1}, \ldots, \lambda_{\overline{s},\ell}$ are the (canonical)
      labelings returned by the recursive calls, set

          $\lambda_{\overline{s}}(v) = \lambda_{\overline{s},i}(v) + \sum_{j=1}^{i-1} |C_j|$

      where $C_i \ni v$.
7. Return the labeling $\lambda_{\overline{s}}$ corresponding to the first flip extension $\overline{s}$ (relative to the order in
   which the flip extensions were enumerated) that is not $\perp$.
```

We then define the canonical labeling by setting $\text{Can}(G) = \text{Can}(G, 0)$ via the subroutine. We now show that our subroutine satisfies the desired properties.

▶ **Lemma 24.** *If G is a graph of rank-width at most k, the above procedure terminates and does not return $\perp$.*

Proof. For termination, we observe that at each step the depth parameter d increases and that if d becomes larger than $3\log(n) + 1$, the procedure returns. Hence, the procedure must terminate.

We will now show by induction that Algorithm 1 returns a labeling instead of $\perp$. Fix (T, γ) to be a rank decomposition of G, of width $\leq 2k$ and height $\leq 3 \cdot (\log(n) + 1)$. Let $t \in V(T)$, and let t_1, t_2 be the children of t in T. We will use Lem. 11, which provides that for each $t \in V(T)$, there exists a flip extension $\overline{s}$ so that for every $C \in \text{Comp}(G[\gamma(t)], \overline{s})$, there exists an $i = 1, 2$ such that $C \subseteq \gamma(t_i)$. We will use this to show that the algorithm constructs a non-empty set of labelings for G. As the algorithm chooses the least such labeling[4], it follows that the algorithm in fact returns a labeling. Note that while the algorithm will not be explicitly constructing (T, γ), the algorithm still descends along (T, γ) in one of its parallel computations.

Consider first the case when $|V(G)| = 1$. Here, the algorithm returns $\lambda(v) = 1$, where $v \in V(G)$. Now fix a node $t \in V(T)$, and let $\gamma(t)$ be the corresponding set of vertices. Suppose that $|\gamma(t)| > 1$. Let t_1, t_2 be the children of t in T. By Lem. 11, there exists a flip extension $\overline{s} = (\overline{a}, \overline{b}, f)$ such that for every component $C \in \text{Comp}(G[\gamma(t)], \overline{s})$, either $C \in \gamma(t_1)$ or $C \in \gamma(t_2)$. As we consider all flip extensions of $\gamma(t)$ in parallel, one of our parallel computations will consider $\overline{s}$. We will analyze this parallel computation.

Prior to recursively invoking the algorithm on each $G[C]$ ($C \in \text{Comp}(G[\gamma(t)], \overline{s})$), the algorithm first sorts said components (For the purposes of showing that the algorithm yields a (not necessarily canonical) labeling, the precise ordering does not matter. We will argue later that the ordering used by the algorithm is canonical– see Lem. 25). For each $C \in \text{Comp}(G[\gamma(t)], \overline{s})$, the algorithm is then applied recursively to $G[C]$.

Now for $i = 1, 2$, let $C_{i,1}, \ldots, C_{i,j_i} \in \text{Comp}(G[\gamma(t)], \overline{s})$ be precisely the components in $\gamma(t_i)$. Observe that a flip extension on $\gamma(t_i)$ restricts to a flip extension on an individual component $C_{i,h}$ ($h \in [j_i]$). Conversely, given flip extensions $\overline{s}_{i,h}$ ($h \in [j_i]$), the union of these flip extensions induce a flip extension $\overline{s}$ on $\gamma(t_i)$.

By Lem. 11, there exists a flip extension $\overline{s}_i$ such that for every component $C' \in \text{Comp}(G[\gamma(t_i)], \overline{s_i})$, $C' \in \gamma(t_{i,1})$ or $C' \in \gamma(t_{i,2})$. Suppose that s_i is the union of the flip extensions $(\overline{s}_{i,h})_{h \in [j_i]}$. As, for each $h \in [j_i]$, the recursive call of the algorithm applied to $C_{i,h}$ will consider all flip extensions of $C_{i,h}$ in parallel. Thus, via the recursive calls to the components $C_{i,h}$ ($h \in [j_i]$), the algorithm will consider all flip extensions of $\gamma(t_i)$, including the flip extension $\overline{s_i}$. Thus, some parallel choice will descend along (T, γ), and so we may assume that the algorithm computes a labeling for each $C \in G([\gamma(t)], \overline{s})$. As these components are disjoint and listed in a fixed order, the algorithm in fact computes a labeling for $\gamma(t)$. The result now follows by induction. ◀

▶ **Lemma 25.** *Let G be a colored graph of rank-width at most k and let H be an arbitrary graph. If $\lambda : V(G) \to [n]$ and $\kappa : V(H) \to [n]$ are the labelings output by Algorithm 1 on G and H respectively, then $G \cong H$ if and only if the map $\kappa^{-1} \circ \lambda$ is an isomorphism.*

[4] with respect to the order in which the flip extensions were enumerated– See Algorithm 1, Line 5

Proof. If $\kappa^{-1} \circ \lambda$ is an isomorphism, then clearly $G \cong H$. We show that if $G \cong H$, then $\kappa^{-1} \circ \lambda$ is an isomorphism. The proof is by induction on the number of vertices in G. Assume that $|V(G)| = 1$ and $G \cong H$. Note that the algorithm returns $\lambda = \kappa = \text{id}$, the identity permutation, on a graph with one vertex. Thus, $\kappa^{-1} \circ \lambda$ is an isomorphism as desired.

Suppose that $|V(G)| > 1$. Let λ be the labeling returned for G and κ the labeling returned for H. Let $(\overline{a}, \overline{b})$ be the split pair the algorithm selects for λ on the initial call (when the algorithm is invoked on G with depth $= 0$). By the algorithm, $\chi_{10k+3,O(\log n)}((\overline{a}, \overline{b}))$ belongs to the minimal color class where a labeling was returned. Let $(\overline{a}', \overline{b}')$ be the corresponding split pair of H selected for κ. Observe[5] that $(10k + 3, O(\log(n))$-WL must assign the same color to the tuples $(\overline{a}, \overline{b})$ and $(\overline{a}', \overline{b}')$.

As the algorithm enumerates the flip extensions in lexicographical order, it considers the flip functions in lexicographical order. As the ordering on flip functions does not depend on the choice of split pair, and we have that $G \cong H$, the flip function[6] $f : (2^{\overline{a} \cup \overline{b}})^2 \to [n] \cup \{\perp\}$ selected for G will also be used for H. Write $\overline{s} := (\overline{a}, \overline{b}, f)$ and $\overline{s}' := (\overline{a}', \overline{b}', f)$. The algorithm next computes the flipped graphs $G^{\overline{s}}$ and $H^{\overline{s}'}$. By Lem. 12, we have that: $(G^{\overline{s}}, \chi^{(\overline{a},\overline{b})}_{6k+3,O(\log n)}) \cong (H^{\overline{s}'}, \chi^{(\overline{a}',\overline{b}')}_{6k+3,O(\log n)})$. It follows that $\ell := |\text{Comp}(G, \overline{s})| = |\text{Comp}(H, \overline{s}')|$. Label the components of $\text{Comp}(G, \overline{s})$ as $C_1, \ldots, C_\ell$, and the components of $\text{Comp}(H, \overline{s}')$ as $D_1, \ldots, D_\ell$. Furthermore, by (4), there exists a bijection $\psi : [\ell] \to [\ell]$ such that for all $i \in [\ell]$, $G[C_i] \cong H[D_{\psi(i)}]$. In particular, as we compute $\chi^{(\overline{a},\overline{b})}_{10k+3,O(\log n)}$ at line 6(a), the isomorphism class of $G[C_i] \cong H[D_{\psi(i)}]$ takes into account how $G[C_i]$ connects to the rest of G and how $H[D_{\psi(i)}]$ connects back to the rest of H (see the discussion in the two paragraphs immediately below Lem. 23). As the algorithm sorts the components of $\text{Comp}(G, \overline{s})$ (respectively, $\text{Comp}(H, \overline{s}')$), we may without loss of generality take ψ to be the identity permutation.

By the inductive hypothesis, we may assume that for each $i \in [\ell]$, the algorithm computes a labeling $\ell_i : C_i \to [|C_i|]$, a labeling $\kappa_i : \psi'(C_i)) \to [|C_i|]$, and that $\kappa_i^{-1} \circ \ell_i$ is an isomorphism. Now by construction, if $v \in C_i$, then

$$\lambda(v) = \lambda_i(v) + \sum_{j=1}^{i-1} |C_j|,$$

and κ is defined analogously. As $C_i \cap C_h = \emptyset$ (resp., $D_i \cap D_h = \emptyset$) whenever $i \neq h$, λ and κ are well-defined. Furthermore, as $\kappa_i^{-1} \circ \lambda_i$ is an isomorphism of $G[C_i] \cong H[D_i]$ for each $i \in [\ell]$, $\kappa|^{-1}_{H[D_i]} \circ \lambda|_{G[C_i]}$ is an isomorphism of $G[C_i] \cong H[D_i]$.

Now suppose that v, w belong to different components of $\text{Comp}(G, \overline{s})$. Let $v' := (\kappa^{-1} \circ \lambda)(v)$ and $w' := (\kappa^{-1} \circ \lambda)(w)$. We will show that $vw \in E(G)$ if and only if $v'w' \in E(H)$. By the definition of the flipped graph (see Section 3), we can determine whether $vw \in E(G)$ based on $N(v) \cap (\overline{a} \cup \overline{b})$, $N(w) \cap (\overline{a} \cup \overline{b})$, and $|N(v) \cap [w]_{\equiv s}|$. All of this information is encoded in $\chi^{(\overline{a},\overline{b})}_{10k+3,O(\log n)}((v, w))$, and $\chi^{(\overline{a}',\overline{b}')}_{10k+3,O(\log n)}((v, w))$. Thus, $vw \in E(G)$ if and only if $v'w' \in E(H)$. It follows that the map $\kappa^{-1} \circ \lambda$ is an isomorphism. The result follows. ◀

Proof of Thm. 20. Let $\lambda : V(G) \to [n]$ be the output of $\text{Can}(G, 3\log(n) + 1)$. Correctness follows from Lem. 24 and Lem. 25. We now establish the complexity. At each recursive call to $\text{Can}(G, d)$, we invoke $(10k + 3, O(\log n))$-WL on G, once at line (4), and then in parallel for each flip extension. Our calls to $(10k + 3, O(\log n))$-WL are TC^1-computable [25]. We

[5] Full details of this claim will appear in the full version.

[6] We abuse $\overline{a} \cup \overline{b}$ to denote the indices of the vertices as they appear in $(\overline{a}, \overline{b})$.

write down the flipped graph and identify its connected components in L (Lem. 23). So the non-recursive work within a single call to $\text{Can}(G, d)$ is TC^1-computable. $\text{Can}(G, d)$ makes $n^{O(16^k)}$ recursive calls. The height of our recursion tree is $O(\log n)$. The result follows. ◀

5 Logarithmic Weisfeiler–Leman and Treewidth

In the process of our work, we came across a way to modestly improve the descriptive complexity for graphs of bounded treewidth. Our main result in this section is the following.

▶ **Theorem 26.** *The $(3k + 6)$-dimensional Weisfeiler–Leman algorithm identifies graphs of treewidth k in $O(\log n)$ rounds.*

In order to prove Thm. 26, we utilize a result of [5] that graphs of treewidth k admit a *binary* tree decomposition of width $\leq 3k + 2$ and height $O(\log n)$. With this decomposition in hand, we leverage a pebbling strategy that is considerably simpler than that of Grohe & Verbitsky [25] and improves the descriptive complexity (Cor. 6).

▶ **Lemma 27.** *Let G, H be graphs. Suppose that a separator $S \subseteq V(G)$ has been pebbled. If the corresponding pebbled set $S' \subseteq V(H)$ is not a separator of H, then Spoiler can win with 3 additional pebbles and $O(\log n)$ additional rounds.*

This next lemma states that if we have pebbled the vertices of some node $\beta(t)$ of the tree decomposition (T, β), then Spoiler can force Duplicator to preserve a given subtree T' (setwise) of the tree decomposition by pebbling some vertex $v \in V(G)$ where there exists $u \in V(T')$ such that $v \in \beta(u)$.

▶ **Lemma 28.** *Let G be a connected graph, and let (T, β) be the binary tree decomposition of G afforded by [5]. Let $t \in V(T)$, and suppose that each vertex in $\beta(t)$ has been pebbled. Let C be the connected component of $T - tu$ that contains u, and let $T' := C \cup tu$.*

Let $v, w \in V(G)$ be vertices contained in the subgraph of G induced by T', such that $v, w \notin \beta(t)$. Suppose that $(v, w) \mapsto (v', w')$ are pebbled. Let $f : V(G) \to V(H)$ be Duplicator's bijection. If v', w' belong to different components of $H \setminus f(\beta(t) \setminus \beta(u))$, then Spoiler can win with 1 additional pebble and $O(\log n)$ additional rounds.

Proof Sketch of Thm. 26. Full details will in the full version. If G is not connected, it is easy to see that Spoiler can force Duplicator to play on non-isomorphic components. Thus, without loss of generality, we may assume that G is connected. Let (T, β) be a tree decomposition for G of width $\leq 3k+2$ and height $O(\log n)$, with T a binary tree, as prescribed by [5]. Let s be the root node of T. Spoiler begins by pebbling the vertices of $\beta(s)$, using $\leq 3k + 3$ pebbles. Let $f : V(G) \to V(H)$ be Duplicator's bijection. If $G[\beta(s)] \not\cong H[f(\beta(s))]$, then Spoiler wins. So suppose that $G[\beta(s)] \cong H[f(\beta(s))]$.

Let ℓ be the left child of s, and r be the right child of s in T. At the next two rounds, Spoiler places a pebble on some vertex of $\beta(\ell) \setminus \beta(s)$ and a pebble on some vertex of $\beta(r) \setminus \beta(s)$. By Lem. 28, Duplicator must select bijections preserving the left and right subtrees. Necessarily, either the left or right sub-tree is mapped to a non-isomorphic component of H. Without loss of generality, suppose the left sub-tree is mapped to a non-isomorphic component of H. In this case, Spoiler removes the pebble in $\beta(r)$ and all but one pebble of $\beta(s) \setminus \beta(\ell)$.

We may thus iterate on the above argument, starting from ℓ as the root node in our subtree in the tree decomposition. As $G \not\cong H$, we will eventually reach a stage (such as when all of $\beta(t)$ is pebbled for some leaf node $t \in V(T)$) where the map induced by the pebbled vertices does not extend to an isomorphism. In the full version, we will carefully show that only $3k + 6$ pebbles on the board and $O(\log n)$ rounds suffice. ◀

References

1 V. Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5):835–852, 2006. doi:10.1016/j.ic.2006.02.002.

2 L. Babai, E. Luks, and A. Seress. Permutation groups in NC. In *STOC 1987*, STOC '87, pages 409–420, New York, NY, USA, 1987. Association for Computing Machinery. doi:10.1145/28395.28439.

3 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 684–697. ACM, New York, 2016. Preprint of full version at arXiv:1512.03547v2 [cs.DS]. doi:10.1145/2897518.2897542.

4 László Babai. Canonical form for graphs in quasipolynomial time: Preliminary report. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1237–1246, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316356.

5 Hans L. Bodlaender. NC-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, pages 1–10, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:10.1007/3-540-50728-0_32.

6 Harry Buhrman and Steven Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 1992. doi:10.1007/3-540-56287-7_99.

7 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. Originally appeared in SFCS '89. doi:10.1007/BF01305232.

8 Bruno Courcelle and Mamadou Kanté. Graph operations characterizing rank-width and balanced graph expressions. In *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, pages 66–75, June 2007. doi:10.1007/978-3-540-74839-7_7.

9 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

10 Bireswar Das, Anirban Dasgupta, Murali Krishna Enduri, and I. Vinod Reddy. On nc algorithms for problems on bounded rank-width graphs. *Information Processing Letters*, 139:64–67, 2018. doi:10.1016/j.ipl.2018.07.007.

11 Bireswar Das, Jacobo Torán, and Fabian Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. *Information and Computation*, 217:71–83, 2012. doi:10.1016/j.ic.2012.05.003.

12 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 203–214, 2009. doi:10.1109/CCC.2009.16.

13 Heinz-Dieter Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer, 2 edition, 1994. doi:10.1007/978-1-4757-2355-7.

14 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 383–392, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591865.

15 Michael Elberfeld and Pascal Schweitzer. Canonizing graphs of bounded tree width in logspace. *ACM Trans. Comput. Theory*, 9(3), October 2017. doi:10.1145/3132720.

16 Vyacheslav Futorny, Joshua A. Grochow, and Vladimir V. Sergeichuk. Wildness for tensors. *Lin. Alg. Appl.*, 566:212–244, 2019. Preprint arXiv:1810.09219 [math.RT]. doi:10.1016/j.laa.2018.12.022.

17 Joshua A. Grochow and Youming Qiao. Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions. arXiv:1907.00309 [cs.CC], 2019. doi:10.48550/arXiv.1907.00309.

18 Martin Grohe. Isomorphism testing for embeddable graphs through definability. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 63–72, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335313.

19 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5), November 2012. doi:10.1145/2371656.2371662.

20 Martin Grohe and Sandra Kiefer. A Linear Upper Bound on the Weisfeiler-Leman Dimension of Graphs of Bounded Genus. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 117:1–117:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.117.

21 Martin Grohe and Sandra Kiefer. Logarithmic Weisfeiler-Leman Identifies All Planar Graphs. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 134:1–134:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.134.

22 Martin Grohe and Daniel Neuen. Canonisation and definability for graphs of bounded rank width. *ACM Trans. Comput. Log.*, 24(1):6:1–6:31, 2023. doi:10.1145/3568025.

23 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An improved isomorphism test for bounded-tree-width graphs. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3382082.

24 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1010–1029, 2015. doi:10.1109/FOCS.2015.66.

25 Martin Grohe and Oleg Verbitsky. Testing graph isomorphism in parallel by playing a game. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2006. doi:10.1007/11786986_2.

26 Lauri Hella. Definability hierarchies of generalized quantifiers. *Annals of Pure and Applied Logic*, 43(3):235–271, 1989. doi:10.1016/0168-0072(89)90070-5.

27 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

28 Sang il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

29 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

30 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

31 Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The Weisfeiler–Leman dimension of planar graphs is at most 3. *J. ACM*, 66(6), November 2019. doi:10.1145/3333003.

32 Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representations in logspace. *SIAM Journal on Computing*, 40(5):1292–1315, 2011. doi:10.1137/10080395X.

33 Johannes Köbler, Uwe Schöning, and Jacobo Torán. Graph isomorphism is low for PP. *Comput. Complex.*, 2:301–330, 1992. doi:10.1007/BF01200427.

34 Johannes Köbler and Oleg Verbitsky. From invariants to canonization in parallel. In Edward A. Hirsch, Alexander A. Razborov, Alexei Semenov, and Anatol Slissenko, editors, *Computer Science – Theory and Applications*, pages 216–227, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-79709-8_23.

35 Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. On the isomorphism problem for helly circular-arc graphs. *Information and Computation*, 247:266–277, 2016. doi:10.1016/j.ic.2016.01.006.

36 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, January 1975. doi:10.1145/321864.321877.

37 Leonid Libkin. *Elements of Finite Model Theory.* Springer, 2004. doi:10.1007/978-3-662-07003-1_1.

38 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM Journal on Computing*, 46(1):161–189, 2017. doi:10.1137/140999980.

39 Rudolf Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979. doi:10.1016/0020-0190(79)90004-8.

40 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.

41 Sang-il Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008. doi:10.1002/jgt.20280.

42 Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988. doi:10.1016/0022-0000(88)90010-4.

43 Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010. doi:10.1007/S00224-009-9188-4.

44 Jacobo Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.

45 Oleg Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *STACS 2007*, pages 682–693, Berlin, Heidelberg, 2007. Springer-Verlag. doi:10.5555/1763424.1763505.

46 Fabian Wagner. Graphs of bounded treewidth can be canonized in AC^1. In *Proceedings of the 6th International Conference on Computer Science: Theory and Applications*, CSR'11, pages 209–222, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-20712-9_16.

Sparse Cuts in Hypergraphs from Random Walks on Simplicial Complexes

Anand Louis
Indian Institute of Science, Bangalore, India

Rameesh Paul
Indian Institute of Science, Bangalore, India

Arka Ray
Indian Institute of Science, Bangalore, India

Abstract

There are a lot of recent works on generalizing the spectral theory of graphs and graph partitioning to k-uniform hypergraphs. There have been two broad directions toward this goal. One generalizes the notion of graph conductance to hypergraph conductance [Louis, Makarychev – TOC'16; Chan, Louis, Tang, Zhang – JACM'18]. In the second approach, one can view a hypergraph as a simplicial complex and study its various topological properties [Linial, Meshulam – Combinatorica'06; Meshulam, Wallach – RSA'09; Dotterer, Kaufman, Wagner – SoCG'16; Parzanchevski, Rosenthal – RSA'17] and spectral properties [Kaufman, Mass – ITCS'17; Dinur, Kaufman – FOCS'17; Kaufman, Openheim – STOC'18; Oppenheim – DCG'18; Kaufman, Openheim – Combinatorica'20].

In this work, we attempt to bridge these two directions of study by relating the spectrum of *up-down walks* and *swap walks* on the simplicial complex, a downward closed set system, to hypergraph expansion. More precisely, we study the simplicial complex obtained by downward closing the given hypergraph and random walks between its levels $X(l)$, i.e., the sets of cardinality l. In surprising contrast to random walks on graphs, we show that the spectral gap of swap walks and up-down walks between level m and l with $1 < m \leqslant l$ cannot be used to infer any bounds on hypergraph conductance. Moreover, we show that the spectral gap of swap walks between $X(1)$ and $X(k-1)$ cannot be used to infer any bounds on hypergraph conductance. In contrast, we give a Cheeger-like inequality relating the spectra of walks between level 1 and l for any $l \leqslant k$ to hypergraph expansion. This is a surprising difference between swaps walks and up-down walks!

Finally, we also give a construction to show that the well-studied notion of *link expansion* in simplicial complexes cannot be used to bound hypergraph expansion in a Cheeger-like manner.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Sparse Cuts, Random Walks, Link Expansion, Hypergraph Expansion, Simplicial Complexes, High Dimensional Expander, Threshold Rank

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.33

Related Version *Full Version*: https://arxiv.org/abs/2212.13406

Funding *Anand Louis*: Supported in part by SERB Award CRG/2023/002896 and the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).
Rameesh Paul: Supported by Prime Minister's Research Fellowship, India.
Arka Ray: Supported by the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).

Acknowledgements We would like to thank Madhur Tulsiani, Fernando Jeronimo, and anonymous referees for their helpful comments.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 33; pp. 33:1–33:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In recent years, there have been two broad directions of generalizations of graph partitioning and the spectral theory of graphs to hypergraphs. One direction attempts to generalize the notion of conductance in graphs to hypergraphs [23, 8]. The graph expansion (also referred to as graph conductance) is defined as

$$\phi_G \overset{\text{def}}{=} \min_{\substack{S\subseteq V\\ \mathsf{vol}_G(S)\leqslant\frac{\mathsf{vol}_G(V)}{2}}} \phi_G(S), \text{ where } \phi(S) \overset{\text{def}}{=} \frac{w(\partial_G(S))}{\mathsf{vol}_G(S)}$$

with $\mathsf{vol}_G(S)$ being the sum of degrees of the vertices in S and $\partial_G(S)$ being the edges crossing the boundary of the set S, hence $w(\partial_G(S))$ is the sum of weights of the edges on the boundary. Analogously, the hypergraph expansion/conductance is defined as

$$\phi_H \overset{\text{def}}{=} \min_{\substack{S\subseteq V\\ \mathsf{vol}_H(S)\leqslant\frac{\mathsf{vol}_H(V)}{2}}} \phi_H(S), \text{ where } \phi_H(S) \overset{\text{def}}{=} \frac{\Pi\left(\partial_H(S)\right)}{\mathsf{vol}_H(S)}$$

with $\mathsf{vol}_H(S)$ being the sum of degrees of the vertices in S, and $\partial_H(S)$ being the edges crossing the boundary of the set S, and $\Pi(\partial_H(S))$ is the sum of the weight of edges on the boundary.

Another direction views a hypergraph as a *simplicial complex*, a downward closed set system, and studies its various topological properties [22, 24, 12, 26] and spectral properties [19, 11, 20, 21, 25]. The work [11] introduced a generalization of random walks on graphs to random walks over the faces[1] of the simplicial complex; this random walk has found numerous applications in a myriad of other problems [11, 9, 4, 3, 1], etc., to state a few.

There has been a lot of work on understanding the relationship between random walks on graphs (including the spectra of the random walk operator) and graph partitioning. The celebrated Cheeger's inequality gives one such relation between the graph expansion and the second eigenvalue of the random walk matrix λ_2 as,

$$\frac{1-\lambda_2}{2} \leqslant \phi_G \leqslant \sqrt{2(1-\lambda_2)}.$$

In this work, we aim to bridge the gap between these two directions by studying the relationship between hypergraph expansion and random walks on the corresponding simplicial complex.

In a seminal work, [5] showed that if a graph has a "small" threshold rank[2], then they can compute a near-optimal assignment to unique games in time exponential in the threshold rank. The works [7, 15] gave an SoS hierarchy-based algorithm generalizing this result to any 2-CSP. The work [2] introduces the notion of *swap walks* and uses that to define a notion of threshold rank for simplicial complexes. Using their notion of threshold rank, they generalized the results of [7, 15] to k-CSPs. Further, [5] showed that large threshold rank graphs must have a small non-expanding set (they also gave a polynomial time algorithm to compute such a set). A natural open question from the work of [2, 17] is whether hypergraphs with large threshold rank (the hypergraph analogue is called non-splittability) have a small, non-expanding set. Our first result answers this question negatively.

[1] The faces (the hyperedges) here may belong to different levels. A level $X(l)$ denotes the set of hyperedges of cardinality l.

[2] the number of "large" eigenvalues of the adjacency matrix, see Definition 30 for formal definition.

▶ **Theorem 1** (Informal Version of Theorem 34 and Corollary 35). *For any $n \geqslant 6, k \geqslant 3$, there exists a k-uniform hypergraph H with at least n vertices such that $\phi_H \geqslant \frac{1}{k}$ but for any m, l, if either $m, l \geqslant 2$ or $m = k - l$, the swap walk from $X(m)$ to $X(l)$ has threshold rank at least $\Omega_k(n)$ (for any $\tau \in [-1, 1]$ as choice of threshold). Moreover, H is not $(\tau, \Omega_k(n))$-splittable for any $\tau \in [-1, 1]$.*

For a splittable hypergraph, there is some l, such that the swap walk graph between $X(l)$ and $X(k-l)$ has low threshold rank. Then, it follows from Theorem 1 that there are non-splittable expanding hypergraphs (see Corollary 35 for the precise statement).

[2, 9] show that for a high dimensional expander (HDX)[3] the swap walks indeed have a large spectral gap[4]. However, we are interested in the case when the hypergraph instance is not an HDX. One recalls that for a non-expanding graph, Cheeger's inequality and Fiedler's algorithm allow us to compute a combinatorial sparse cut in the graph. Similarly, we ask whether one can compute a sparse cut in the input hypergraph in this setting.

Unfortunately, in the light of Theorem 1, computing a sparse cut in the hypergraph when swap walks (in the setting studied by [2, 17]; see Theorem 34 for the precise statement) have a small spectral gap is generally not possible. This is in surprising contrast to the case of graphs where the swap walk reduces to the usual random walk, and the second largest eigenvalue of the random walk matrix is related to graph expansion via Cheeger's inequality.

Next, we investigate whether the spectral gap of the up-down walk introduced by [11] can be related to hypergraph expansion. More formally, we investigate whether the spectral gap of the up-down walk between levels $X(m)$ and $X(l)$ $(l > m)$ be related to the hypergraph expansion in a Cheeger-like manner. Here, the answer depends on m and l. We first show that if $m \geqslant 2$, then no such relation is possible.

▶ **Theorem 2** (Informal Version of Theorem 36). *For any positive integers n, k with $n \geqslant 6, k \geqslant 3$, there exists a k-uniform hypergraph H on at least n vertices such that $\phi_H \geqslant \frac{1}{k}$ and for all positive integers $2 \leqslant m < l \leqslant k$ the threshold rank of the up-down walk matrix between levels $X(m)$ and $X(l)$ is at least $\Omega_k(n)$ (for any $\tau \in [-1, 1]$ as choice of threshold).*

Contrasting this, we show that if $m = 1$, then such a relationship is indeed possible.

▶ **Theorem 3** (Informal Version of Theorem 18). *Given a hypergraph, where the second largest eigenvalue of the up-down walk matrix (of simplicial complex induced by the hypergraph) between levels $X(1)$ and $X(l)$, for some $l \in [k]$ is $1 - \varepsilon$ we have $\frac{\varepsilon}{k} \leqslant \phi_H \leqslant 4\sqrt{\varepsilon}$. Furthermore, there exists a polynomial time algorithm which, when given such a hypergraph, outputs a set S such that its expansion in the hypergraph $\phi_H(S) \leqslant 4\sqrt{\varepsilon}$.*

Theorem 3 and Theorem 1 also show a surprising difference between up-down walks and swap walks whereby we can compute sparse cut on the hypergraph using up-down walk from $X(1)$ to $X(l), l \in [k]$ using a Cheeger-like inequality, whereas it is not possible (in general) to compute a sparse cut by considering the spectrum of swap walks from $X(1)$ to $X(k-1)$.

Yet another notion of spectral expansion called *link expansion* of a simplicial complex has been studied recently in many works [19, 11, 20, 21, 25] having applications in [11, 9, 4, 3, 1] (see Definition 9 for formal definition). Our final result shows that hypergraphs with large hypergraph expansion and arbitrarily small link expansion exist. Therefore, hypergraph expansion cannot be bounded by link expansion in a Cheeger-like manner.

[3] For formal definition see Definition 9.

[4] For a linear operator $\mathsf{A} : V \to W$ where $V \neq W$ the spectral gap refers to $\sigma_1(A) - \sigma_2(A)$, while for a linear operator $B : V \to V$, it refers to $\lambda_1(A) - \lambda_2(A)$.

▶ **Theorem 4** (Informal Version of Theorem 43). *Let n, k be any positive integers such that $n \geqslant 3k$ and $k \geqslant 3$, there exists a k-uniform hypergraph H on $n + k - 2$ vertices such that the link expansion of the induced simplicial complex X is at most $\mathcal{O}(\frac{1}{n^2})$ and the expansion of H is at least $\Omega_k(1)$.*

To the best of our knowledge, this is the first construction to show this.

The work [23] (see Remark 1.9) used an example similar in spirit to our constructions to show that another notion of expansion on simplicial complexes called co-boundary expansion is incomparable to the hypergraph expansion. In particular, they constructed a class of k-uniform hypergraphs, each with co-boundary expansion (at dimension k) as one but containing hypergraphs with essentially arbitrary hypergraph expansion. Still, [23] did not give an explicit example that shows a separation between hypergraph expansion and quantities like the link expansion, spectral gap, or threshold rank of the random walks on a simplicial complex (i.e., up-down walk, swap walk).

The m-dimensional co-boundary expansion may also seem related to the expansion of the up-walk from the level $m - 1$ to m as both of these consider the ratio of the number of m-dimensional faces containing a set of $m - 1$-dimensional faces to the volume of the set with the only difference being how the volume is computed. Yet, we do not know if such a relation exists. One may similarly compare the expansion of the down-walk and the boundary expansion. But still, Steenbergen, Klivian, and Mukherjee [28] and Gundert and Wagner [14] were able to show that for the m-dimensional co-boundary expansion no Cheeger-type inequality can be shown, whereas such a relation is immediate from Cheeger's inequality in case of up-walk. Nevertheless, [28] obtained (under some minor assumptions) an extension of Cheeger's inequality on the m-dimensional boundary expansion. Finally, [10] showed that the operator norm of the difference between up-down and down-up walks between two consecutive levels is within an $\mathcal{O}(k)$ factor of link expansion. In contrast, no such relation between up-Laplacian, down-Laplacian (see [28] for definition) and link expansion is known.

1.1 Additional Related Works

The work [8] generalized the Laplacian of graphs to hypergraphs by expressing the graph Laplacian in terms of a non-linear diffusion process. They showed an analogue of Cheeger's inequality relating the expansion of the hypergraph to the second smallest eigenvalue of the Laplacian. Yoshida [30] introduced the notion of submodular transformations and extended the notions of degree, cut, expansion, and Laplacian to them. They derived the Cheeger's inequality in this setting. This generalizes Cheeger's inequality on graphs and hypergraphs (as in [8]) while showing similar inequalities for entropy.

There are also several works exploring Cheeger-like inequalities for simplicial complexes. Parzanchevski, Rosenthal, and Tessler [27] defined the notion of Cheeger constant $h(X)$ for a simplicial complex, a generalization of the sparsity of a graph. The quantity $h(X)$ is the minimum over all partitions of the vertex set V into k sets the fraction of k-dimensional faces present crossing the partition compared to the maximum possible k-dimensional faces crossing the partition. They also showed that for simplicial complex X with a complete skeleton $h(X) \geqslant \lambda(X)$ where $\lambda(X)$ is the link expansion of the simplicial complex. Gundert and Szedlák [13] extended this result to any simplicial complex. Very recently, Jost and Zhang [18] extended the Cheeger-like inequality for bipartiteness ratio[5] on graphs due to Trevisan [29] to a cohomology based definition of bipartiteness ratio for simplicial complexes.

[5] The bipartiteness ratio of G is defined as $\beta_G = \min_{S \subseteq V, L \sqcup R = S} \frac{2\partial(L) + 2\partial(R) + \partial(S)}{\mathsf{vol}_G(S)}$.

In the case of an HDX, Bafna, Hopkins, Kaufmann, and Lovett [6] consider high-dimensional walks (a generalization of swap walks and up-down walks) on levels $i < k$. They then relate the (non-)expansion of a link[6] of a level-j face (with $j \leqslant i$) in the graph corresponding to the walk and level-j approximate eigenvalue of the walk. Here λ_j is the level-j approximate eigenvalue of a high-dimensional walk M if there is a function f_j such that $\|\mathsf{M} f_j - \lambda_j f_j\| \leqslant O(\sqrt{\gamma})\,\|f_j\|$ and $f_i = \mathsf{U}^{i-j} g$ where $g \in \mathbb{R}^{X(j)}$.

1.2 Preliminaries

1.2.1 Simplicial Complexes

▶ **Definition 5.** *A simplicial complex X is a set system that consists of a ground set V and a downward closed collection of subsets of V, i.e., if $s \in X$ and $t \subseteq s$ then $t \in X$. The sets in X are called the faces of X.*

We define a level/slice $X(l)$ of the simplicial complex X as $X(l) = \{s \in X | |s| = l\}$. Note that for the simplicial complex corresponding to the hypergraph, the top level $X(k)$ is the set of k-uniform hyperedges and the ground set of vertices[7] is denoted by $X(1)$. By convention we have that $X(0) = \{\emptyset\}$. Similarly, we define $X(\leqslant l) = \{s \in X | |s| \leqslant l\}$.

We call a simplicial complex X as k-dimensional if k is the smallest integer for which $X(\leqslant k) = X$.[8] A k-dimensional simplicial complex X is a pure simplicial complex if for all $s \in X$ there exists $t \in X(k)$ such that $s \subseteq t$.

▶ Remark 6. We note that our definition of dimension deviates slightly from the standard definition. In the standard definition, the dimension is the cardinality of the largest face minus 1.

Given a k-uniform hypergraph $H = (V, E)$, we obtain a pure simplicial complex X where the ground set is V and downward close the set system E of hyperedges. Given a distribution Π_k on the hyperedges, we have an induced distribution Π_l on sets s in level $X(l)$ given by $\Pi_l(s) = \frac{1}{\binom{k}{l}} \sum_{e \in E | s \subseteq e} \Pi_k(e)$. We refer to the joint distribution as $\Pi = (\Pi_k, \Pi_{k-1}, \ldots, \Pi_1)$. If the input hypergraph is unweighted, then we take the distribution Π_k to be the uniform distribution on $X(k)$. We thus obtain a weighted simplicial complex (X, Π). We refer to (X, Π) as the (weighted[9]) simplicial complex *induced* by (H, Π_k).

▶ **Lemma 7** (Folkore). *For any two non-negative integers $m \leqslant l$ and any $s \in X(m)$, we have that $\sum_{t \in X(l) | t \supseteq s} \Pi_l(t) = \binom{l}{m} \Pi_m(s)$.*

In this work, we consider a notion of expansion for weighted simplicial complexes called link expansion. To that end, we first define the notion of a link of a complex and its skeleton.

▶ **Definition 8.** *For a simplicial complex X and some $s \in X$, X_s denotes the link complex of s defined by $X_s = \{t \setminus s | s \subseteq t \in X\}$. The skeleton of a link X_s for a face $s \in X(\leqslant k-2)$ (where k is the size of the largest face) denoted by $G(X_s)$ is a weighted graph with vertex set $X_s(1)$, edge set $X_s(2)$ and weights proportional to Π_2.*

[6] [6] uses a different (albeit related) notion of the link of a face $\sigma \in X(j)$. There, the link of a face σ is the set of level-i faces containing σ.

[7] We shall often simply write v for a face $\{v\} \in X(1)$

[8] We shall often write $X(\leqslant k)$ for X to stress the fact that X is k-dimensional

[9] Whenever it is clear from the context, we use X in place of (X, Π) for the sake of brevity.

▶ **Definition 9** (γ-HDX, [19, 11]). *A simplicial complex $X(\leqslant k)$ is a γ-High Dimensional Expander (γ-HDX) if for all $s \in X(\leqslant k-2)$, the second singular value of the adjacency matrix of the graph $G(X_s)$ (denoted by $\sigma_2(G(X_s))$) satisfies $\sigma_2(G(X_s)) \leqslant \gamma$. We refer to $1-\gamma$ as the link-expansion of X.*

▶ **Definition 10** (Weighted inner product). *Given two functions $f, g \in \mathbb{R}^S$, i.e., $f, g : S \to \mathbb{R}$ and a measure μ on S, we define the weighted inner product of these functions as, $\langle f, g\rangle_\mu = \mathbb{E}_{s\sim\mu}[f(s)g(s)] = \sum_{s\in S} f(s)g(s)\mu(s)$. We drop the subscript μ from $\langle\cdot,\cdot\rangle_\mu$ whenever μ is clear from context.*

▶ Remark. In this paper, we will use the weighted inner product between two functions $f, g \in \mathbb{R}^{X(m)}$ on levels $X(m)$ of the simplicial complex X under consideration and with the measure Π_m, unless otherwise specified. In particular, for any linear operator $\mathsf{A} : \mathbb{R}^{X(m)} \to \mathbb{R}^{X(l)}$ the adjoint $\mathsf{A}^\dagger$ and the i-th largest singular value $\sigma_i(\mathsf{A})$ are with respect to this inner-product.

1.2.2 Walks on a Simplicial Complex

▶ **Definition 11** (Up and Down operators). *Given a simplicial complex (X, Π), we define the up operator $\mathsf{U}_i : \mathbb{R}^{X(i)} \to \mathbb{R}^{X(i+1)}$ that acts on a function $f \in \mathbb{R}^{X(i)}$ as*

$$[\mathsf{U}_i f](s) = \mathop{\mathbb{E}}_{s'\in X(i), s'\subseteq s}[f(s')] = \frac{1}{i+1}\sum_{x\in s} f(s\setminus\{x\})$$

and the down operator $\mathsf{D}_{i+1} : \mathbb{R}^{X(i+1)} \to \mathbb{R}^{X(i)}$ that acts on a function $g \in \mathbb{R}^{X(i+1)}$ as

$$[\mathsf{D}_{i+1} g](s) = \mathop{\mathbb{E}}_{s'\sim\Pi_{i+1}, s'\supset s}[g(s')] = \frac{1}{i+1}\sum_{x\notin s} g(s\cup\{x\})\frac{\Pi_{i+1}(s\cup\{x\})}{\Pi_i(s)}\,.$$

As a consequence of the definition of the up and down operators, the following holds.

▶ **Lemma 12** (Folklore). $\mathsf{U}_i^\dagger = \mathsf{D}_{i+1}$.

The up operator, U_i, can be thought of as defining a random walk moving from $X(i+1)$ to $X(i)$ where a subset of size i is selected uniformly for a given face $s \in X(i+1)$. Similarly, the down operator D_{i+1} can be thought of as defining a random walk moving from $X(i)$ to $X(i+1)$ where a superset $s' \in X(i+1)$ of size $i+1$ is selected for a given face $s \in X(i)$ with probability $\frac{\Pi_{i+1}(s')}{\Pi_i(s)}$. This leads us to the following definition.

▶ **Definition 13.** *Given a simplicial complex (X, Π) and its two levels $X(m)$, $X(l)$, we define a bipartite graph on $X(m) \cup X(l)$ as $B_{m,l} = (X(m) \cup X(l), E_{m,l}, w_{m,l})$ where $E_{m,l} = \{\{s,t\}\,|\,s \in X(m), t \in X(l),$ and $s \subseteq t\}$ and $m \leqslant l$. The weight of an edge $\{s,t\}$ where $s \in X(m)$ and $t \in X(l)$ is given by $w_{m,l}(s,t) = \binom{k}{l}\Pi_l(t)$.*

As we will show in Fact 16, in the random walk on $B_{m,l}$ the block corresponding to the transition from a vertex in $X(m)$ to a vertex in $X(l)$ is the up walk (i.e., the down operator) and the block corresponding to the transition from a vertex in $X(l)$ to a vertex in $X(m)$ is the down-walk (i.e., the up operator).

Now, we define the $B^{(2)}_{m,l}$ graph such that the random walk on it corresponds to the two-step walk starting from vertices in $X(m)$ on $B_{m,l}$, i.e., the random walk on $B^{(2)}_{m,l}$ corresponds to an up-walk followed by a down-walk. Fact 17 shows that this correspondence indeed holds.

▶ **Definition 14.** *Given a simplicial complex* (X, Π) *and its two levels* $X(m)$, $X(l)$ *with* $m \leqslant l$, *we define a graph on* $X(m)$ *as* $B^{(2)}_{m,l} = (X(m), E^{(2)}_{m,l}, w^{(2)}_{m,l})$ *where*

$$E^{(2)}_{m,l} = \{\{s,t\} \,|\, s,t \in X(m) \text{ and } \exists s' \in X(l) \text{ such that } s' \supseteq s \cup t\}.$$

The weight of an edge $\{s,t\}$ *where* $s,t \in X(m)$ *is given by* $w^{(2)}_{m,l}(s,t) = \sum_{s' \supseteq s \cup t} w_{m,l}(s,s') = \binom{k}{l} \sum_{s' \supseteq s \cup t} \Pi_l(s')$. *The normalized adjacency matrix corresponding to* $B^{(2)}_{m,l}$ *is denoted by* $\mathsf{A}^{(2)}_{m,l}$.

▶ **Definition 15** (Up-Down Walk, [19, 20]). *For positive integers* $m \leqslant l$, *let* $\mathsf{D}_{m,l}$ *and* $\mathsf{U}_{l,m}$ *denote the products,* $\mathsf{D}_{m+1}\mathsf{D}_{m+2}\ldots\mathsf{D}_{l-1}\mathsf{D}_l$ *and* $\mathsf{U}_{l-1}\mathsf{U}_{l-2}\ldots\mathsf{U}_{m+1}\mathsf{U}_m$ *respectively. We denote the following walk between* $X(m)$ *and* $X(l)$ *as* $\mathsf{N}_{m,l}$,

$$\mathsf{N}_{m,l} = \begin{bmatrix} 0 & \mathsf{D}_{m,l} \\ \mathsf{U}_{l,m} & 0 \end{bmatrix} = \begin{bmatrix} 0 & \mathsf{D}_{m,l} \\ \mathsf{D}^{\dagger}_{m,l} & 0 \end{bmatrix},$$

where the second equality is due to Lemma 12. The up-down walk on $X(m)$ *through* $X(l)$ *is a random walk on* $X(m)$ *whose transition matrix (denoted by* $\mathsf{N}^{(2)}_{m,l}$*) is given by* $\mathsf{N}^{(2)}_{m,l} = \mathsf{D}_{m,l}\mathsf{U}_{l,m}$.

▶ **Fact 16.** *The transition matrix for random walk on* $B_{m,l}$ *is* $\mathsf{N}_{m,l}$.

▶ **Fact 17.** *The transition matrix for random walk on* $B^{(2)}_{m,l}$ *is* $\mathsf{D}_{m,l}\mathsf{U}_{l,m}$.

1.2.3 Notations

We use $[n]$ for the set $\{1, 2, \ldots, n\}$ and $A \sqcup B$ for disjoint union of sets A and B.

2 Computing Sparse Cut in Hypergraphs

Theorem 18 shows an analogue of Cheeger's inequality based on the eigenvalues of up-down walks $\mathsf{N}_{1,l}$.

▶ **Theorem 18.** *Let* $H = (V, E)$ *be a* k*-uniform hypergraph such that the induced simplicial complex* X *has a up-down walk* $\mathsf{N}^{(2)}_{1,l}$ *such that* $\lambda_2(\mathsf{N}_{1,l}) = 1 - \varepsilon$ *for some* $\varepsilon > 0$ *and some* $l \in \{2, 3, \ldots, k\}$. *Then* $\frac{\varepsilon}{k} \leqslant \phi_H \leqslant 4\sqrt{\varepsilon}$. *Furthermore there is an algorithm which on input* H, *outputs a set* $S \subset V$ *such that* $\phi_H(S) \leqslant 4\sqrt{\varepsilon}$ *in* $\mathsf{poly}(|V|, |E|)$ *time where* poly *is a polynomial.*

Fact 19 will allow us to work with $\mathsf{D}_{1,2}$ instead of $\mathsf{N}_{1,l}$ for some $l \in \{3, 4, \ldots, k\}$.

▶ **Fact 19** (Folklore). *Let* $\mathsf{A} \in \mathbb{R}^{n \times m}$, $\mathsf{B} \in \mathbb{R}^{m \times p}$ *and* σ_i *denote the* i^{th} *singular value. Then, we have*

$$\sigma_i(\mathsf{AB}) \leqslant \sigma_1(\mathsf{A})\sigma_i(\mathsf{B}) \text{ and } \sigma_i(\mathsf{AB}) \leqslant \sigma_i(\mathsf{A})\sigma_1(\mathsf{B}),$$

for $i = 1, \ldots, r$, *where* $r = \mathsf{rank}(\mathsf{AB})$.

▶ **Corollary 20.** *If* $\sigma_2(\mathsf{D}_{1,l}) = 1 - \varepsilon$ *for an arbitrary* $l \in \{2, 3, \ldots, k\}$, *we have that* $\sigma_2(\mathsf{D}_{1,2}) \geqslant 1 - \varepsilon$.

Proof. The proof follows by using Fact 19 and writing $\mathsf{D}_{1,l} = \mathsf{D}_{1,2}\mathsf{D}_{2,l}$ to get

$$\sigma_2(\mathsf{D}_{1,l}) = \sigma_2(\mathsf{D}_{1,2}\mathsf{D}_{2,l}) \overset{\text{Fact 19}}{\leqslant} \sigma_2(\mathsf{D}_{1,2})\sigma_1(\mathsf{D}_{2,l}) = \sigma_2(\mathsf{D}_{1,2}),$$

where the last equality holds since $\sigma_1(\mathsf{D}_{2,l}) = 1$. ◀

Next, we show that we can use this information about $\sigma_2(\mathsf{D}_{1,2})$ to compute a set $S \subset V$ such that its expansion in the graph $B^{(2)}_{1,2}$ is at most $2\sqrt{\varepsilon}$.

▶ **Lemma 21.** *If $\sigma_2(\mathsf{D}_{1,2}) = 1 - \varepsilon$ for some $\varepsilon \in (0,1)$, then there exists a set $S \subseteq X(1)$ such that $\phi_{B^{(2)}_{1,2}}(S) \leqslant 2\sqrt{\varepsilon}$. Furthermore, there is a $\mathsf{poly}(|V_{B^{(2)}_{1,2}}|, |E_{B^{(2)}_{1,2}}|)$ time algorithm to compute such a set S.*

A natural choice for our set S with low conductance in input hypergraph is this set S guaranteed by Fiedler's algorithm for which $\phi_{B^{(2)}_{1,2}}(S)$ is small. We show in Lemma 22 that $B^{(2)}_{1,2}$ is a weighted graph where the weight of an edge between two distinct vertices in $X(1)$ is the multiplicity of that edge in the construction of $B^{(2)}_{1,2}$ graph. We note that a hyperedge e, induces a clique on the vertices in the hyperedge e, in the $B^{(2)}_{1,2}$ graph. This is commonly known as the clique expansion of the hypergraph.

▶ **Lemma 22.** *For any k-uniform hypergraph $H = (V, E)$, let X be the induced simplicial complex and let $\{s,t\}$ be an edge in $B^{(2)}_{m,l}$ with $s,t \in X(m)$. Then $w(s,t) = \binom{k-|s\cup t|}{l-|s\cup t|} \sum_{e\in E|s\cup t\subseteq e} \Pi_k(e)$ and $\mathsf{deg}_{B^{(2)}_{m,l}}(s) = \binom{l}{m}^2 \frac{\binom{k}{l}}{\binom{k}{m}} \sum_{e\in E|e\supseteq s} \Pi_k(e)$.*

Now in Lemma 23, we show how the weight of edges cut in the boundary of the weighted graph $B^{(2)}_{1,2}$ and the input hypergraph are related.

▶ **Lemma 23.** *Given a set $S \subset X(1)$ we have*

$$(k-1)\Pi_k\left(\partial_H(S)\right) \leqslant w(\partial_{B^{(2)}_{1,2}}(S)) .$$

Proof. By Lemma 22, $B^{(2)}_{1,2}$ is a weighted graph where the weight $w(i,j)$ of an edge $\{i,j\}$ where $i \neq j$ is given by $w(i,j) = \sum_{e\in E|\{i,j\}\subseteq e} \Pi_k(e)$. Therefore, to compute $w(\partial_{B^{(2)}_{1,2}}(S))$ we sum over all $i \in S$ and $j \in V \setminus S$, the number of hyperedges containing $\{i,j\}$, i.e.,

$$w(\partial_{B^{(2)}_{1,2}}(S)) = \sum_{i\in S, j\in V\setminus S} \sum_{\substack{e\in H\\ e\supseteq\{i,j\}}} \Pi_k(e) = \sum_{e\in H} \sum_{\substack{i\in S, j\in V\setminus S\\ \{i,j\}\subseteq e}} \Pi_k(e),$$

where the last equality in the equation above follows by exchanging the order of summation. Now, we note that the number of $\{i,j\} \subseteq e$ where $i \in S$ and $j \in V \setminus S$ is non-zero if and only if $e \in \partial_H(S)$, and hence,

$$w(\partial_{B^{(2)}_{1,2}}(S)) = \sum_{e\in\partial_H(S)} \sum_{\substack{i\in S, j\in V\setminus S\\ \{i,j\}\subseteq e}} \Pi_k(e). \tag{1}$$

Now, let $e \cap S = \{i_1, i_2, \ldots, i_t\}$ for some $t \in \{1, 2, \ldots, k-1\}$. For the lower bound, we note that the number of $\{i,j\} \subseteq e$ where $i \in S$ and $j \in V \setminus S$ is $t(k-t)$. Therefore, for some $e \in \partial_H(S)$, we have the minimum value of $t(k-t)$ as $k-1$ and hence u eqn. (1) to get,

$$w(\partial_{B^{(2)}_{1,2}}(S)) \geqslant \sum_{e\in\partial_H(S)} (k-1)\Pi_k(e) = (k-1)\Pi_k\left(\partial_H(S)\right) \left(\Pi_k(\partial_H(S)) = \sum_{e\in\partial_H(S)} \Pi_k(e)\right).$$

◀

We now show an upper bound for the boundary of $B^{(2)}_{1,l}$ in terms of the boundary of H.

▶ **Lemma 24.** *For any l, such that $2 \leqslant l \leqslant k$, Given a set $S \subset X(1)$ we have*

$$w(\partial_{B^{(2)}_{1,l}}(S)) \leqslant \binom{k}{l}\binom{l}{2}\Pi_k\left(\partial_H(S)\right) .$$

Next, in Lemma 25, we will use these bounds to analyze the expansion of this set S in the input hypergraph.

▶ **Lemma 25.** *For an arbitrary set $S \subset X(1)$, we have that $\phi_H(S) \leqslant 2\phi_{B^{(2)}_{1,2}}(S)$.*

Proof. We start by comparing the numerator in the expressions for expansion of the given arbitrary set S in original hypergraph $|\partial_H(S)|$ and in the $B^{(2)}_{1,2}$ graph, i.e., $w(\partial_{B^{(2)}_{1,2}}(S))$. Using Lemma 23 we have that, $\Pi_k\left(\partial_H(S)\right) \leqslant \frac{1}{(k-1)} \cdot w(\partial_{B^{(2)}_{1,2}}(S))$.

Next, we compare the denominators in the respective expression for expansions, i.e., $\mathsf{vol}_H(S)$ and $\mathsf{vol}_{B^{(2)}_{1,2}}(S)$. For the hypergraph, by definition we have that $\mathsf{vol}_H(S) = \sum_{i\in S}\mathsf{deg}(i)$. By Lemma 22 we have

$$\mathsf{vol}_{B^{(2)}_{1,2}}(S) = \sum_{i\in S}\mathsf{deg}_{B^{(2)}_{1,2}}(i) = \sum_{i\in S}\binom{2}{1}^2\frac{k(k-1)}{2k}\mathsf{deg}_H(i) = 2(k-1)\mathsf{vol}_H(S).$$

Now, putting everything together, we have

$$\phi_H(S) = \frac{\Pi_k\left(\partial_H(S)\right)}{\mathsf{vol}_H(S)} = 2(k-1)\frac{\Pi_k\left(\partial_H(S)\right)}{\mathsf{vol}_{B^{(2)}_{1,2}}(S)} \leqslant 2\cdot\frac{(k-1)}{(k-1)}\cdot\frac{w(\partial_{B^{(2)}_{1,2}}(S))}{\mathsf{vol}_{B^{(2)}_{1,2}}(S)} = 2\phi_{B^{(2)}_{1,2}}(S)\,. \blacktriangleleft$$

▶ **Lemma 26.** *For an arbitrary set $S \subset X(1)$, we have that $\phi_H(S) \geqslant \frac{2}{k}\phi_{B^{(2)}_{1,l}}(S)$.*

Proof of Theorem 18. First, we note by Fact 51, $1-\varepsilon \leqslant \sqrt{1-\varepsilon} \leqslant \sqrt{\lambda_2(\mathsf{N}^{(2)}_{1,l})} = \sigma_2(\mathsf{D}_{1,l})$.

Now, using Corollary 20 we conclude that $\sigma_2(\mathsf{D}_{1,2}) = 1-\varepsilon' \geqslant 1-\varepsilon$ for some $\varepsilon' \leqslant \varepsilon$. Further, in Lemma 21, we show that we can use this information about the spectrum of $\mathsf{D}_{1,2}$ to compute a set $S \subset V$ such that its expansion in the graph $B^{(2)}_{1,2}$ is at most $2\sqrt{\varepsilon}$. We fix this as the set S we return in our sparse cut. In Lemma 25 we show that expansion of this set S in the input hypergraph is at most $2\phi_{B^{(2)}_{1,2}}(S)$ and hence

$$\phi_H(S) \leqslant 2\phi_{B^{(2)}_{1,2}}(S) \leqslant 4\sqrt{\varepsilon}\,.$$

Now, by Fact 17 the matrices $\mathsf{N}^{(2)}_{1,l}$ and $\mathsf{A}^{(2)}_{1,l}$ are similar and hence have the same eigenvalues and therefore by Cheeger's inequality, we have $\phi_{B^{(2)}_{1,l}} \geqslant \frac{\varepsilon}{2}$. Therefore by Lemma 26, we have

$$\phi_H \geqslant \frac{2}{k}\phi_{B^{(2)}_{1,l}} \geqslant \frac{\varepsilon}{k}. \qquad \blacktriangleleft$$

3 An expanding hypergraph with walks having small spectral gap

3.1 Splittability of a Hypergraph

In this section, we consider a "non-lazy" version of the up-down walk. While typically, for a walk on the graph to be non-lazy, we require that there be no transition from a vertex to itself, we obtain the swap walks by imposing an even stronger condition where we don't allow any face to have a transition to another face with a non-empty intersection with the starting face.

▶ **Definition 27** (Swap walk, [2, 9]). *Given a k-dimensional simplicial complex (X, Π), for non-negative integers m, l such that $l + m \leqslant k$ we define the swap walk denoted by $\mathsf{S}_{m,l} : \mathbb{R}^{X(l)} \to \mathbb{R}^{X(m)}$ that acts on a $f \in \mathbb{R}^{X(l)}$ as,*

$$[S_{m,l}f](s) = \mathop{\mathbb{E}}_{s' \sim \Pi_{m+l} | s' \supseteq s} f(s' \setminus s).$$

▶ **Lemma 28** ([2]). $\mathsf{S}_{m,l}^{\dagger} = \mathsf{S}_{l,m}$.

Again, the swap walk $\mathsf{S}_{m,l}$ can be thought of as defining a random walk moving from $X(m)$ to $X(l)$ where we first move from $s \in X(m)$ to a superset $s'' \in X(m+l)$ with probability $\frac{\Pi_{m+l}(s'')}{\Pi_m(s)}$ and then determistically move to $s' = s'' \setminus s$, i.e., we move from face $s \in X(m)$ to a disjoint face $s' \in X(l)$ with probability $\frac{\Pi_{m+l}(s \sqcup s')}{\Pi_m(s)}$. This leads us to the following definition for swap graphs.

▶ **Definition 29** (Swap graph, Section 6 in [2]). *Given a simplicial complex (X, Π) and its two levels $X(m), X(l)$, the swap graph (denoted by $G_{m,l}$) is defined as a bipartite graph $G_{m,l} = (X(m) \cup X(l), E(m,l), \mathrm{w}_{m,l})$ where the weight function is defined as, $\mathrm{w}_{m,l}(s,t) = \frac{\Pi_{m+l}(s \sqcup t)}{\binom{m+l}{m}}$ and $E(m,l) = \{\{s,t\} \,|s \in X(m), t \in X(l),$ and $s \sqcup t \in X(m+l)\}$.*

The random walk matrix corresponding to these walks denoted by $\mathsf{W}_{m,l}$ is a matrix of size $(|X(m)| + |X(l)|) \times (|X(m)| + |X(l)|)$ and is given by,

$$\mathsf{W}_{m,l} = \begin{bmatrix} 0 & \mathsf{S}_{m,l} \\ \mathsf{S}_{l,m} & 0 \end{bmatrix} = \begin{bmatrix} 0 & \mathsf{S}_{m,l} \\ \mathsf{S}_{m,l}^{\dagger} & 0 \end{bmatrix}, \tag{2}$$

where the last equality is a consequence of Lemma 28.

Arora, Barak, and Steurer [5] introduced the notion of the threshold rank of a graph.

▶ **Definition 30** (Threshold rank of a graph, [5]). *Given a weighted graph $G = (V, E, w)$ and its normalized random walk matrix W such that $\lambda_n(\mathsf{W}) \leqslant \lambda_{n-1}(\mathsf{W}) \leqslant \ldots \leqslant \lambda_1(\mathsf{W}) = 1$ and a threshold $\tau \in (0,1]$, we define the τ-threshold rank of the graph G (denoted by $\mathsf{rank}_{\geqslant\tau}(\mathsf{W})$) as $\mathsf{rank}_{\geqslant\tau}(\mathsf{W}) = |\{i|\lambda_i(\mathsf{W}) \geqslant \tau\}|$.*

[2] proposed an analogue of the threshold rank for hypergraphs called (τ, r)-splittability by considering specific sets of swap walks given by the following class of binary tree.

▶ **Definition 31** (k-splitting tree, Section 7 in [2]). *A binary tree $\mathcal{T}$ given with its labeling is called a k-splitting tree if*
- *$\mathcal{T}$ has exactly k leaves.*
- *The root of $\mathcal{T}$ is labeled with k and all other vertices in $\mathcal{T}$ are labeled with a positive integer.*
- *All the leaves are labeled with 1.*
- *The label of every internal node of $\mathcal{T}$ is the sum of the labels of its two children.*

Now, we define a set of swap walks and its threshold rank based on a k-splitting tree $\mathcal{T}$.

▶ **Definition 32** (Swap graphs in a tree, Section 7 in [2]). *For a simplicial complex $X(\leqslant k)$ and a k-splitting tree $\mathcal{T}$, we consider all swap graphs (denoted by $\mathsf{Swap}(\mathcal{T}, X)$) from $X(a)$ to $X(b)$ where a and b are labels of a non-leaf node in $\mathcal{T}$. Further, we extend the definition of threshold rank as*

$$\mathsf{rank}_{\geqslant\tau}(\mathsf{Swap}(\mathcal{T}, X)) = \max_{G \in \mathsf{Swap}(\mathcal{T}, X)} \mathsf{rank}_{\geqslant\tau}(G).$$

Finally, define (τ, r)-splittability by considering all such sets of swap walks.

▶ **Definition 33** ((τ, r)-splittability, Definition 7.2 in [2])**.** *A k-uniform hypergraph with an induced simplicial complex $X(\leqslant k)$ is said to be (τ, r)-splittable if there exists some k-splittable tree $\mathcal{T}$ such that $\mathsf{rank}_{\geqslant \tau}(\mathsf{Swap}(\mathcal{T}, X)) \leqslant r$.*

3.2 The main results

In Theorem 34, we show an example of an expanding hypergraph such that for all m, l such that $m + l \leqslant k$ the swap walk from $X(m)$ to $X(l)$ in the corresponding simplicial complex has its top r singular values as 1 (for $r \approx n/k$) if either $m, l \geqslant 2$ or $m = k - l$.

▶ **Theorem 34.** *For any positive integers r, k with $r \geqslant 2, k \geqslant 3$, there exists an k-uniform hypergraph H on $n(= r(k-1)+1)$ vertices such that $\phi_H \geqslant \frac{1}{k}$ and for any m, l such that $m + l \leqslant k$, if either $m, l \geqslant 2$ or $m = k - l$ then $\lambda_r(G_{m,l}) = \sigma_r(\mathsf{S}_{m,l}) = 1$, where $\mathsf{S}_{m,l}$, $G_{m,l}$ are the swap walk and the swap graph on the induced simplicial complex X, respectively.*

Now, Corollary 35 is a simple consequence of Theorem 34 and the definition of splittability.

▶ **Corollary 35.** *For any positive integers r, k with $r \geqslant 2, k \geqslant 3$, there exists an k-uniform hypergraph H on $n(= r(k-1)+1)$ vertices, such that $\phi_H \geqslant \frac{1}{k}$ and the induced simplicial complex X is not (τ, r)-splittable for all $\tau \in [-1, 1]$.*

We were also able to show that in the above example, for all m, l such that $2 \leqslant m < l \leqslant k$, the up-down walk from $X(m)$ to $X(l)$ has its top r singular value as 1 (for $r \approx n/k$).

▶ **Theorem 36.** *For any positive integers r, k with $r \geqslant 2, k \geqslant 3$, there exists a k-uniform hypergraph H on $n(= r(k-1)+1)$ vertices such that $\mathsf{rank}_{\geqslant \tau}\left(\mathsf{N}^{(2)}_{m,l}\right) \geqslant r$ for all $\tau \in [-1, 1]$ but $\phi_H \geqslant \frac{1}{k}$.*

We use the following construction to show Theorem 34, Corollary 35 and Theorem 36.

▶ **Construction 37.** *Take the vertex set of the hypergraph $H(V, E)$ to be $V = [n]$ where $n = r(k-1)+1$ and the edge set $E = \{e_1, e_2, \ldots, e_r\}$ where $e_i = \{0, (k-1)(i-1)+1, \ldots, (k-1)i\}$. Let X be the simplicial complex induced by H and $\mathsf{S}_{m,l}, \mathsf{N}_{m,l}$ be the corresponding walk matrices.*

▶ Remark 38. Remark 1.9 of [23] considers all hypergraphs whose edges intersected at most $k-2$ vertices to show a separation between co-boundary expansion and hypergraph expansion. Here, we consider a sub-class of such hypergraphs with edges intersecting exactly one vertex. Although the second singular value of the up-down walks and co-boundary expansion may seem related, a relation between them is not known. Also, the way in which [23] bounds the co-boundary expansion is similar to how we bound the spectrum of the up-down walks. However, here, we also prove that the threshold rank (for any threshold) can be made arbitrarily large while having the same bound on the hypergraph expansion.

First, we show that any swap walk $\mathsf{S}_{l,k-l}$ has $\sigma_i = 1$, for any $i \in [r]$.

▶ **Lemma 39.** *Given a hypergraph as per Construction 37, we have that $\lambda_r(G_{1,k-1}) = \sigma_r(\mathsf{S}_{1,k-1}) = \sigma_r(\mathsf{S}_{k-1,1}) = 1$.*

Proof. Firstly, using Fact 52 and eqn. (2) we have $\lambda_i(G_{1,k-1}) = \sigma_i(\mathsf{S}_{1,k-1}), \forall i \in [r]$.

We note that for any $i \in [r]$, the edge $\{\{(k-1)(i-1)\}, e_i \setminus \{(k-1)(i-1)\}\}$ is the only edge in $G_{1,k-1}$ (and $G_{k-1,1}$) incident on the vertices $\{(k-1)(i-1)\}, e_i \setminus \{(k-1)(i-1)\}$. Again, $G_{1,k-1}$ has r connected components, and hence $\lambda_r(G_{1,k-1}) = \sigma_r(\mathsf{S}_{1,k-1}) = \sigma_r(\mathsf{S}_{k-1,1}) = 1$. ◀

▶ **Lemma 40.** *Given a hypergraph as per Construction 37, and for any $m, l \geqslant 2$ such that $m + l \leqslant k$, we have that $\lambda_r(G_{m,l}) = \sigma_r(\mathsf{S}_{m,l}) = 1$.*

▶ **Lemma 41.** *Given a hypergraph as per Construction 37 and an arbitrary set $S \subseteq V$ where $\mathsf{vol}_H(S) \leqslant \mathsf{vol}_H(V)/2$, we have that $\phi_H(S) \geqslant \frac{1}{k}$.*

Proof. We consider an arbitrary (non-empty) set $S \subset V$ such that $\mathsf{vol}_H(S) \leqslant \mathsf{vol}_H(V)/2$. Let $|S \cap e_1| = t_1, |S \cap e_2| = t_2, \ldots, |S \cap e_r| = t_r$ and let $t = t_1 + t_2 + \ldots t_r$. We note that $\mathsf{vol}_H(V) = r(k-1) + r$ where $r(k-1)$ is the contribution from the vertices in $V \setminus \{0\}$ and we have a contribution of r from the vertex $\{0\}$. Next, we will precisely compute the expansion $\phi_H(S)$. We will break into cases depending upon whether $\{0\} \in S$ or $\{0\} \notin S$.

First, consider the case where $\{0\} \in S$. We note that in this case, $t_i \geqslant 1, \forall i \in [r]$. In this case, we have that $|\{i | t_i = k\}| < r/2$. This is because otherwise $\mathsf{vol}_H(S) \geqslant r + \frac{r}{2}(k-1) > \frac{rk}{2}$ which contradicts $\mathsf{vol}_H(S) \leqslant \mathsf{vol}_H(V)/2$. Thus, $|\{i | t_i < k\}| \geqslant r/2$ and hence $\partial_H(S) \geqslant r/2$. Next we have that $\mathsf{vol}_H(S) = r + \sum_{i=1}^{r}(t_i - 1) = t_1 + t_2 + \ldots t_r = t$. Using $\mathsf{vol}_H(S) \leqslant \mathsf{vol}_H(V)/2$, we have that $t \leqslant rk/2$ and we get

$$\phi_H(S) = \frac{|\partial_H(S)|}{\mathsf{vol}_H(S)} \geqslant \frac{r}{2t} \geqslant \frac{1}{k}.$$

Next, we consider the case where $\{0\} \notin S$. Let $t^+ = |\{i\} \, | t_i > 0|$. Since $\{0\} \notin S$, we know that $t_i < k, \forall i \in [r]$ and hence the number of edges in the boundary of S is exactly t^+. Moreover we can bound the volume of S as $\mathsf{vol}_H(S) \leqslant t^+(k-1)$ and hence we have

$$\phi_H(S) = \frac{|\partial_H(S)|}{\mathsf{vol}_H(S)} \geqslant \frac{t^+}{t^+(k-1)} \geqslant \frac{1}{k}.$$ ◀

Proof of Theorem 34. Immediate from Lemma 41, Lemma 40, and Lemma 39. ◀

Proof of Corollary 35. Consider the hypergraph H (and the induced simplicial complex) guaranteed by Theorem 34. Fix any $\tau \in [-1, 1]$ and any k-splitting tree $\mathcal{T}$. We note $G_{l,k-1} \in \mathsf{Swap}(\mathcal{T}, X)$ for some $l \in [k-1]$ as children of the root of $\mathcal{T}$ must be labeled l and $k - l$ for some l. Note that we have $\lambda_r(G_{l,k-l}) = 1$. Hence, we have $\mathsf{rank}_{\geqslant\tau}(\mathsf{Swap}(\mathcal{T}, X)) \geqslant \mathsf{rank}_{\geqslant\tau}(G_{l,k-l}) \geqslant r$. Since, $\mathsf{rank}_{\geqslant\tau}(\mathsf{Swap}(\mathcal{T}, X)) \geqslant r$ for any k-splitting tree $\mathcal{T}$, therefore (X, Π) is not (τ, r)-splittable for any $\tau \in [-1, 1]$. ◀

▶ **Lemma 42.** *Given a hypergraph as per Construction 37, and any $m, l \in [k]$ such that $2 \leqslant m \leqslant l$, we have that $\lambda_r(\mathsf{N}^{(2)}_{m,l}) = 1$.*

Proof of Theorem 36. Immediate from Lemma 41 and Lemma 42. ◀

4 An expanding hypergraph with low link expansion

In Theorem 43, we show that there is a family of expanding k-uniform hypergraphs H with the induced simplicial complex having low link expansion.

▶ **Theorem 43.** *Let n, k be any positive integers such that $n \geqslant 3k$ and $k \geqslant 3$, there exists a k-uniform hypergraph H on $n + k - 2$ vertices such that the link expansion of the induced simplicial complex X is at most $1 - \cos\frac{2\pi}{n}$ and the expansion of H is at least $\frac{1}{(3k)^k}$.*

Construction 44 is a k-hypergraph with $n + k - 2$ vertices such that its expansion is $\frac{1}{(3k)^k}$ while the link expansion for the induced simplicial complex is $1 - \cos\frac{2\pi}{n}$.

▶ **Construction 44.** *Take the vertex set of the hypergraph $H(V, E)$ to be $V = [n+k-2]$ and the edge set $E = \binom{[n]}{k} \cup \{e \cup \{n+1, \ldots, n+k-2\} \,|e \in C_n\}$ where $C_n = \{\{i, i+1\} \,|i \in [n-1]\} \cup \{\{n, 1\}\}$, i.e., C_n is the set of edges in a cycle on $[n]$. Let X be the simplicial complex induced by H.*

The idea behind this construction is to have the cycle C_n as the link of $\{n+1, ..., n+k-2\}$ while adding sufficient edges to make the hypergraph into an expanding hypergraph.

▶ **Lemma 45.** *For any n, k such that $n \geqslant 3k$ and $k \geqslant 3$, the hypergraph H as defined in Construction 44 has expansion $\phi_H \geqslant \frac{1}{(3k)^k}$.*

We now show that the simplicial complex X is not a γ-HDX (refer to Definition 9). For this we consider the face $\tau = \{n+1, n+2, \ldots, n+k-2\}$ and the link complex X_τ.

By definition of X_τ and our construction in Construction 44, the two-dimensional link complex X_τ is the downward closure of C_n. Hence, the corresponding skeleton graph $G(X_\tau)$ is the cycle on $[n]$.

▶ **Fact 46** (Folklore). *The second singular value of the normalized adjacency matrix of an n-cycle is $\cos \frac{2\pi}{n}$.*

Therefore, we have the following lemma by Definition 9.

▶ **Lemma 47.** *X has link expansion at most $1 - \cos \frac{2\pi}{n}$.*

Theorem 43 follows directly from Lemma 47 and Lemma 45.

References

1 Dorna Abdolazimi, Kuikui Liu, and Shayan Oveis Gharan. A matrix trickle-down theorem on simplicial complexes and applications to sampling colorings. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science – FOCS 2021*, pages 161–172. IEEE Computer Soc., Los Alamitos, CA, 2022.

2 Vedat Levi Alev, Fernando Granha Jeronimo, and Madhur Tulsiani. Approximating constraint satisfaction problems on high-dimensional expanders. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science*, pages 180–201. IEEE Comput. Soc. Press, Los Alamitos, CA, 2019. doi:10.1109/FOCS.2019.00021.

3 Nima Anari, Kuikui Liu, and Shayan Oveis Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 1319–1330. IEEE Computer Soc., Los Alamitos, CA, 2020. doi:10.1109/FOCS46700.2020.00125.

4 Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: High-dimensional walks and an FPRAS for counting bases of a matroid. In *STOC'19 – Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1–12. ACM, New York, 2019. doi:10.1145/3313276.3316385.

5 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science – FOCS 2010*, pages 563–572. IEEE Computer Soc., Los Alamitos, CA, 2010.

6 Mitali Bafna, Max Hopkins, Tali Kaufman, and Shachar Lovett. High dimensional expanders: Eigenstripping, pseudorandomness, and unique games. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1069–1128. SIAM, 2022. doi:10.1137/1.9781611977073.47.

7 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science – FOCS 2011*, pages 472–481. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.95.

8 T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. Spectral properties of hypergraph Laplacian and approximation algorithms. *J. ACM*, 65(3):Art. 15, 48, 2018. doi:10.1145/3178123.

9 Yotam Dikstein and Irit Dinur. Agreement testing theorems on layered set systems. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science*, pages 1495–1524. IEEE Comput. Soc. Press, Los Alamitos, CA, 2019. doi:10.1109/FOCS.2019.00088.

10 Yotam Dikstein, Irit Dinur, Yuval Filmus, and Prahladh Harsha. Boolean function analysis on high-dimensional expanders. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, volume 116 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 38, 20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

11 Irit Dinur and Tali Kaufman. High dimensional expanders imply agreement expanders. In *58th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2017*, pages 974–985. IEEE Computer Soc., Los Alamitos, CA, 2017. doi:10.1109/FOCS.2017.94.

12 Dominic Dotterrer, Tali Kaufman, and Uli Wagner. On expansion and topological overlap. In Sándor P. Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, volume 51 of *LIPIcs*, pages 35:1–35:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.SoCG.2016.35.

13 Anna Gundert and May Szedlák. Higher dimensional discrete cheeger inequalities. *J. Comput. Geom.*, 6(2):54–71, 2015. doi:10.20382/jocg.v6i2a4.

14 Anna Gundert and Uli Wagner. On eigenvalues of random complexes. *Israel Journal of Mathematics*, 216:545–582, 2016.

15 Venkatesan Guruswami and Ali Kemal Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for graph partitioning and quadratic integer programming with PSD objectives. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 482–491. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.36.

16 Kenneth Hoffman and Ray Kunze. *Linear Algebra.* Prentice-Hall, 2nd edition, 1971.

17 Fernando Granha Jeronimo, Shashank Srivastava, and Madhur Tulsiani. Near-linear time decoding of ta-shma's codes via splittable regularity. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1527–1536. ACM, 2021. doi:10.1145/3406325.3451126.

18 Jürgen Jost and Dong Zhang. Cheeger inequalities on simplicial complexes. *arXiv preprint*, 2023. arXiv:2302.01069.

19 Tali Kaufman and David Mass. High dimensional random walks and colorful expansion. In *8th Innovations in Theoretical Computer Science Conference*, volume 67 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 4, 27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

20 Tali Kaufman and Izhar Oppenheim. Construction of new local spectral high dimensional expanders. In *STOC'18 – Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 773–786. ACM, New York, 2018. doi:10.1145/3188745.3188782.

21 Tali Kaufman and Izhar Oppenheim. High order random walks: Beyond spectral gap. *Comb.*, 40(2):245–281, 2020. doi:10.1007/S00493-019-3847-0.

22 Nathan Linial and Roy Meshulam. Homological connectivity of random 2-complexes. *Comb.*, 26(4):475–487, 2006. doi:10.1007/s00493-006-0027-9.

23 Anand Louis and Yury Makarychev. Approximation algorithms for hypergraph small-set expansion and small-set vertex expansion. *Theory Comput.*, 12:Paper No. 17, 25, 2016. doi:10.4086/toc.2016.v012a017.

24 Roy Meshulam and N. Wallach. Homological connectivity of random k-dimensional complexes. *Random Struct. Algorithms*, 34(3):408–417, 2009. doi:10.1002/rsa.20238.

25 Izhar Oppenheim. Local spectral expansion approach to high dimensional expanders Part II: Mixing and geometrical overlapping. *Discrete Comput. Geom.*, 64(3):1023–1066, 2020. doi:10.1007/s00454-019-00117-7.

26 Ori Parzanchevski and Ron Rosenthal. Simplicial complexes: spectrum, homology and random walks. *Random Structures Algorithms*, 50(2):225–261, 2017. doi:10.1002/rsa.20657.

27 Ori Parzanchevski, Ron Rosenthal, and Ran J. Tessler. Isoperimetric inequalities in simplicial complexes. *Comb.*, 36(2):195–227, 2016. doi:10.1007/s00493-014-3002-x.

28 John Steenbergen, Caroline J. Klivans, and Sayan Mukherjee. A cheeger-type inequality on simplicial complexes. *Adv. Appl. Math.*, 56:56–77, 2014. doi:10.1016/j.aam.2014.01.002.

29 Luca Trevisan. Max cut and the smallest eigenvalue. *SIAM J. Comput.*, 41(6):1769–1786, 2012. doi:10.1137/090773714.

30 Yuichi Yoshida. Cheeger inequalities for submodular transformations. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2582–2601. SIAM, 2019. doi:10.1137/1.9781611975482.160.

A Additional Preliminaries

Linear Algebra

We recall a few facts and definitions from linear algebra.

▶ **Fact 48** ([16]). *Let V, W be two vector spaces with inner products $\langle \cdot,\cdot\rangle_V$, $\langle \cdot,\cdot\rangle_W$. If $A : V \to W$ be a linear operator, then there exists a unique linear operator $B : W \to V$ such that $\langle \mathsf{A}f, g\rangle_W = \langle f, \mathsf{B}g\rangle_V$. If $v \in V$ then there exists a unique linear operator $C : V \to \mathbb{R}$ such that $Cu = \langle v, u\rangle_V$ for any $u \in V$.*

▶ **Definition 49.** *Given a linear operator $\mathsf{A} : V \to W$ between two vector spaces V and W with inner products $\langle \cdot,\cdot\rangle_V$ and $\langle \cdot,\cdot\rangle_W$ defined on them, the adjoint of A is defined as the (unique) linear operator $\mathsf{A}^\dagger : W \to V$ such that $\langle \mathsf{A}f, g\rangle_W = \left\langle f, \mathsf{A}^\dagger g\right\rangle_V$ for any $f \in V$ and $g \in W$. Furthermore, given any $v \in V$ we define $v^\dagger : V \to \mathbb{R}$ as the linear operator which satisfies $v^\dagger u = \langle v, u\rangle_V$ for any $u \in V$.*

It can be easily verified that most properties of the transpose of an operator also hold for the adjoint, e.g., $(\mathsf{A}^\dagger)^\dagger = \mathsf{A}$, $(\mathsf{AB})^\dagger = \mathsf{B}^\dagger\mathsf{A}^\dagger$, etc.

▶ **Definition 50.** *Given a linear operator $\mathsf{A} : V \to W$ between two inner product spaces V and W a singular value σ is a non-negative real number such that there exists $v \in V$ and $w \in W$ which satisfy $\mathsf{A}v = \sigma w$ and $w^\dagger\mathsf{A} = \sigma v^\dagger$. The vectors v and w are called the right and left singular vectors, respectively, associated with the singular value σ. We denote the i-th largest singular value of A by $\sigma_i(\mathsf{A})$.*

▶ **Fact 51.** *Let V,W be two inner product spaces, and $\mathsf{A} : V \to W$ be a linear operator. Then the eigenvalues $\lambda_i(\mathsf{A}^\dagger\mathsf{A})$ are non-negative. Furthermore, the singular values $\sigma_i(\mathsf{A}) = \sqrt{\lambda_i(\mathsf{A}^\dagger\mathsf{A})}$.*

▶ **Fact 52.** *Let V,W be two inner product spaces and $\mathsf{A} : V \to W$ be a linear operator and let B be defined by the expression,*

$$\mathsf{B} = \begin{bmatrix} 0 & \mathsf{A} \\ \mathsf{A}^\dagger & 0 \end{bmatrix}$$

then for any $i \in \{1, \ldots, r\}$, $\sigma_i(\mathsf{A}) = \lambda_i(\mathsf{B})$ where $r = \mathsf{rank}(\mathsf{A})$.

Reconfiguration Algorithms for Cubic Modular Robots with Realistic Movement Constraints

MIT–NASA Space Robots Team[1]
Massachusetts Institute of Technology, Cambridge, MA, USA
NASA Ames Research Center, Moffett Field, CA, USA

Josh Brunner ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Kenneth C. Cheung ✉
NASA Ames Research Center, Moffett Field, CA, USA

Erik D. Demaine ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Jenny Diomidova ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Christine Gregg ✉
NASA Ames Research Center, Moffett Field, CA, USA

Della H. Hendrickson ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Irina Kostitsyna ✉
KBR at NASA Ames Research Center, Moffett Field, CA, USA

Abstract

We introduce and analyze a model for self-reconfigurable robots made up of unit-cube modules. Compared to past models, our model aims to newly capture two important practical aspects of real-world robots. First, modules often do not occupy an exact unit cube, but rather have features like bumps extending outside the allotted space so that modules can interlock. Thus, for example, our model forbids modules from squeezing in between two other modules that are one unit distance apart. Second, our model captures the practical scenario of many passive modules assembled by a single robot, instead of requiring all modules to be able to move on their own.

We prove two universality results. First, with a supply of auxiliary modules, we show that any connected polycube structure can be constructed by a carefully aligned plane sweep. Second, without additional modules, we show how to construct any structure for which a natural notion of external feature size is at least a constant; this property largely consolidates forbidden-pattern properties used in previous works on reconfigurable modular robots.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Modular robotics, programmable matter, digital materials, motion planning

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.34

Related Version *Full Version*: https://arxiv.org/abs/2405.15724

1 Introduction

Algorithmic shape formation with self-reconfigurable modular robots has attracted significant interest by the computational geometry community in the past two decades [16, 2, 7, 9, 8, 10, 6, 34, 17, 24, 29, 4, 3, 5, 15, 12, 13, 25, 18, 28, 27]. In general, the idea is to build a

[1] Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT–NASA Space Robots Team" (without "et al.").

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 34; pp. 34:1–34:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

self-reconfigurable "robot" out of n identical modules, each of which can move in some way relative to its neighbors, subject to some constraints like maintaining global connectivity. This approach enables the robot to drastically change its overall shape, often with algorithmic universality results showing that any shape is possible, up to some constant feature size and/or forbidden small patterns. Real-world modular robots have been built by multiple robotics groups [31, 11, 33, 21, 32, 36], with the ultimate goal of building "programmable matter": objects that can arbitrarily change their shape.

Sliding cubes. One of the first and simplest models for modular robot reconfiguration is ***sliding squares*** in 2D [16] or ***sliding cubes*** in 3D [2, 1]. Each module is a unit square/cube placed at a node of a square/cube lattice, and the modules must at each step have a connected dual graph (according to facet adjacencies). Figure 1 illustrates the two possible moves: (1) ***straight slide*** moves a (green) module to an adjacent empty location along two faces of a pair of adjacent (blue) modules; and (2) ***corner slide*** moves a (green) module to an adjacent empty location, and immediately turning the 90° corner around its original (blue) neighbor, moves the module one more unit to restore the adjacency with the neighbor. This model enables universal reconfiguration between polyominoes/polycubes, with $\Theta(n^2)$ moves necessary and sufficient in the worst case for 2D [16] and (in a recent breakthrough) for 3D [1, 23].

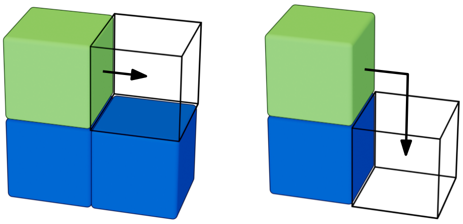


Figure 1 The two moves in the sliding-cubes model. Left: straight slide. Right: corner slide.

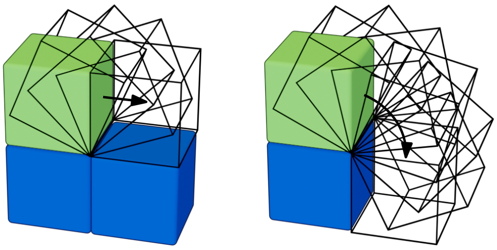


Figure 2 The two moves in the pivoting-cubes model. Left: straight pivot. Right: corner pivot.

Pivoting cubes. Another extensively studied model for modular robot reconfiguration is the ***pivoting squares/hexagons*** in 2D [13, 3, 4] and ***pivoting cubes*** in 3D [34, 17, 24]. In the pivoting-cube model, a module can move by rotating around a common edge shared with an adjacent module. Similarly to the previous model, a module moves to an adjacent empty location or to an empty location around the corner of an adjacent module (see Figure 2). However, unlike in the sliding-cube model, for a pivoting move to be valid, all cells of the grid intersected by the pivoting module must remain empty. Existing results rely on the definition of so called ***forbidden pattern***, that is, a specific constant-size configuration of empty and non-empty cells, whose existence may block a possible reconfiguration. Specifically, the forbidden patterns are of the form: for any $k_1 \times k_2 \times k_3$ (for specific small values of k_1, k_2, and k_3) axis aligned bounding box of grid cells with two modules present in the two diagonally opposite corners of the box, the remaining cells of the box must not be empty. The set of forbidden patterns consists of the $3 \times 1 \times 1$-pattern (two modules with a single empty cell in between), the $2 \times 2 \times 1$-pattern (two edge adjacent modules with no other mutually adjacent modules), and the $3 \times 2 \times 1$-pattern. The series of works on reconfiguration in the pivoting-cube model [34, 17, 24] resulted in a universal reconfiguration algorithm for a class of shapes that do not contain any of the three forbidden patterns.

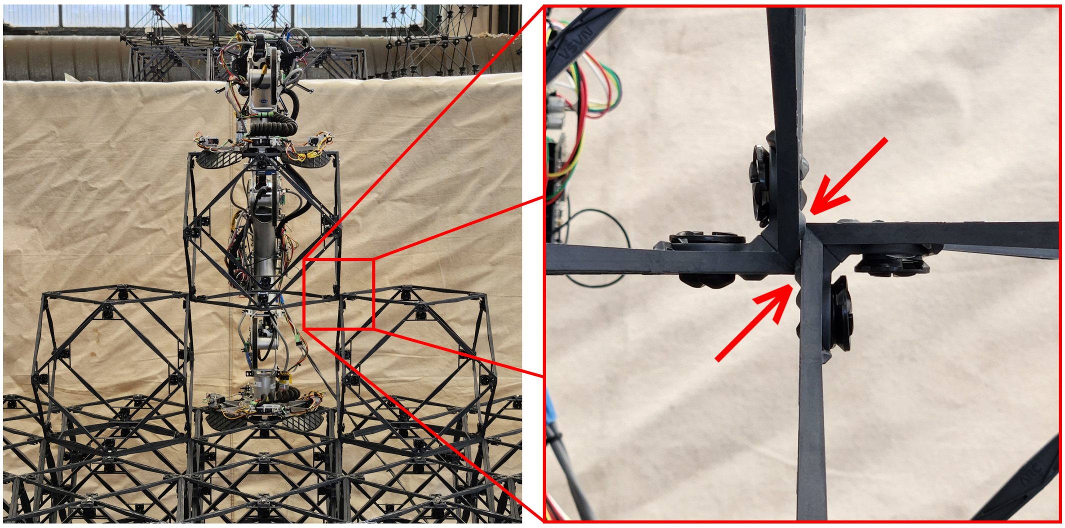


Figure 3 Photograph of ARMADAS robot attempting to place a module between two modules and failing due to collisions of mechanical alignment features (red arrows).

1.1 Our Model

In this paper, we introduce a new model for modular robot reconfiguration that aims to capture two important practical aspects of real-world systems, motivated by our experience with the robots and structural modules of the NASA Ames Automated Reconfigurable Mission Adaptive Digital Assembly Systems (ARMADAS) project [30]. Our model is a refinement of the sliding-cubes model, adding constraints to the moves. Notably, the constraints on the moves in our model are strictly stronger than in the pivoting-cube model as well. Thus our universality results can be seen as strengthenings of past work to better apply to real-world robotics.

Loose sliding. The first practical issue we address is that modules typically need more room than a unit cube to actually move without collision. Figure 3 shows an example of this issue in the context of ARMADAS. To enable secure and precise relative positioning of adjacent modules, modules often have mechanical alignment features (matching bumps and indentations) that extend outside the bounding box. Having these features integrated into modules allows for high-precision, high-repeatability, and high-throughput manufacturing processes such as injection molding to make high-quality connections, rather than requiring the added complexity and weight of active attachment mechanisms [14, 20].

To avoid collision between modules with these bumps, modules need to move slightly away from neighboring modules before sliding to an adjacent cell. Thus a moving module needs the space to move a small distance away from its neighbors. In particular, it is impossible for a module to pass through a unit-wide gap between two other modules; see Figure 4.

We formalize this requirement as the ***loose-sliding*** constraint (refer to Figure 5): a moving module must at all times be within an otherwise empty $2 \times 2 \times 2$ cube in space that moves continuously (or equivalently, moves in unit axis-aligned steps on a grid). In other words, each unit step taken by the moving module (one for a straight slide, two for a corner slide) must have both its start and end positions within a common $2 \times 2 \times 2$ cube empty of other modules; in addition, in the case of a corner slide, the first $2 \times 2 \times 2$ empty cube must be translatable (continuously or in unit axis-aligned steps) to the second $2 \times 2 \times 2$ empty

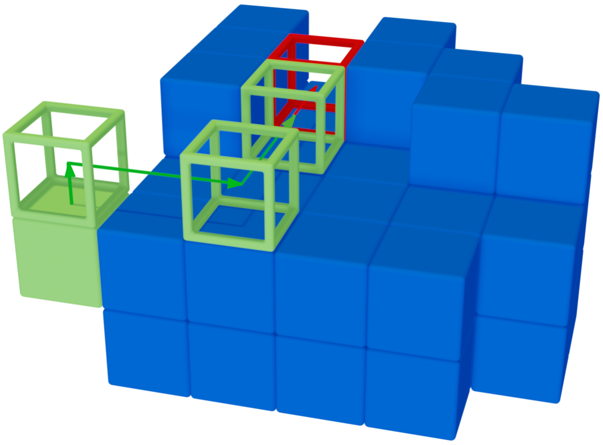


Figure 4 Valid (green) and invalid (red) moves of the green module in the loose-sliding model.

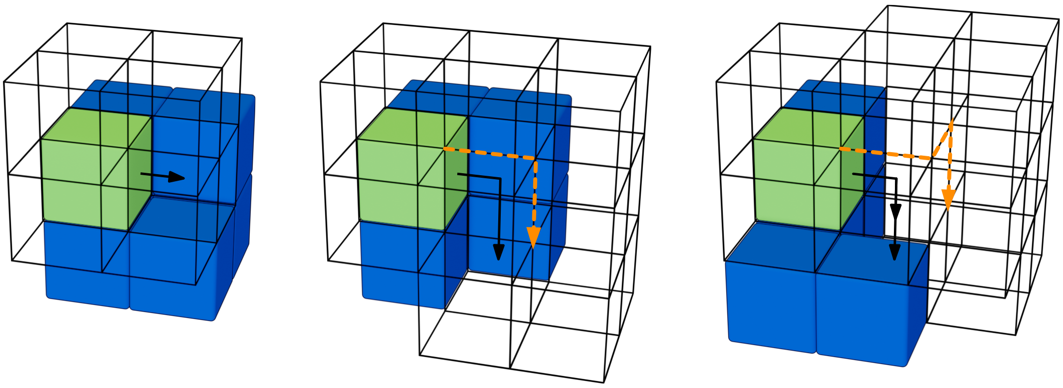


Figure 5 Valid moves in the loose-sliding model. Left: straight slide move and the corresponding $2 \times 2 \times 2$ empty cube. Middle and right: corner slide move and the corresponding translation (indicated by the orange paths) of the $2 \times 2 \times 2$ empty cube.

cube while remaining empty of other modules throughout the translation. In particular, loose sliding prevents a module from sliding into a unit-wide gap between two other modules, because such a gap is not contained in an empty $2 \times 2 \times 2$ cube. More generally, we can define ***k*-loose sliding** to require a $k \times k \times k$ empty cube surrounding the moving module.

Passive modules via accessible sliding. The second practical issue is that modular reconfigurable robots do not scale well. Each module must have its own power, processor, networking, attachment mechanisms, and movement actuators. This module complexity limits the number of modules and practicality of the programmable matter dream.

A more recent alternative approach [30, 22, 35] is to have two types of modules: many passive/static modules that cannot move on their own and primarily serve structural purposes, and a smaller number of robots (even one) that can pick up, carry, place, and attach passive modules. Figure 6 shows this approach in action in the context of the ARMADAS project, where the primary goal is to assemble passive parts into a desired geometry – a digital cellular solid [14, 19]. (Of course, disassembly and reconfiguration is also possible.) The moving robots are more complicated than the passive modules, and thus are naturally a little larger. The simplicity of passive modules enables them to be cheaply constructed in large numbers, vastly increasing applicability [20].

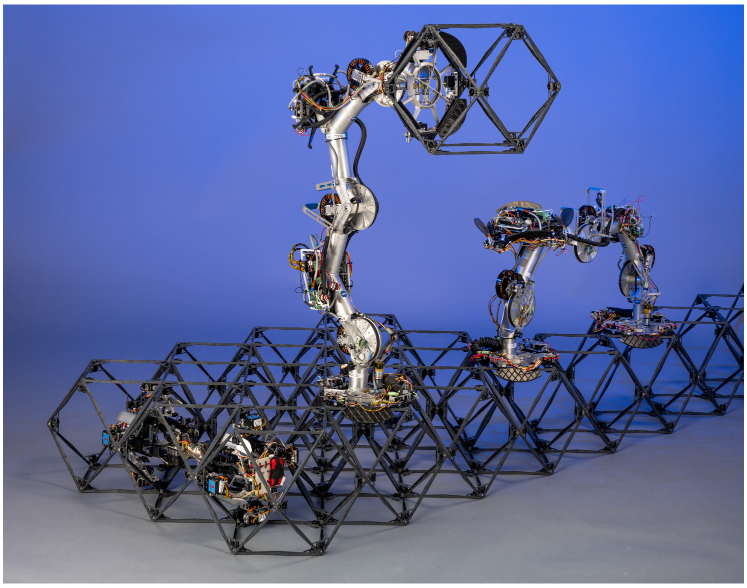


Figure 6 Photograph of ARMADAS robots operating in the laboratory. Each of the external robots has two primary "hands", and can hold onto the already-built structure with either hand while moving the other hand. The right robot is carrying a module via a third hand on its back; this module can be added to the structure by another robot, such as the left robot.

Instead of explicitly modeling both moving robot(s) and passive modules, which would vastly complicate the model, we show how to obtain a similar effect via a small tweak to the loose-sliding-cubes model. Specifically, define a slide to be ***k-accessible*** if it is both k-loose (i.e. is the only module in some otherwise empty $k \times k \times k$ cube), and also that that $k \times k \times k$ is connected to infinity via a path of $k \times k \times k$ empty cubes, both before and after the move. Such a module can be reached by a moving robot on the outside of the shape, removed and picked up by the robot, and then placed and attached in the new location. Thus any sequence of k-accessible moves can be simulated by passive modules plus one moving robot whose size is at most $k \times k \times k$.

Our algorithms for sliding cubes satisfy this k-accessible property. Thus they are equally suitable for both modular robot reconfiguration (where every module can move on its own) and a hybrid system of passive modules and one or more moving robots. By contrast, all past sliding-cube algorithms [16, 2, 1] do not satisfy the accessible property. Indeed, *no* universal algorithm without extra modules can be accessible (even 1-accessible), because it is known that there are 3D configurations with no movable modules on the outside [26].

1.2 Our Results

We develop two very different reconfiguration algorithms, establishing two different universality results in the 2-accessible sliding-cubes model:

1. Allowing extra modules, we show how to construct *any* connected polycube from a straight line, and thus how to reconfigure between any two connected polycubes [Section 2]. The total number of extra modules is linear in the number of modules in the polycube. We also give a negative result that there are structures which are impossible to reconfigure between in the 3-loose sliding-cubes model, thus showing that our algorithm's result is tight.

2. Without extra modules, we show how to construct any connected polycube having "external feature size" at least 2, and thus how to reconfigure between any two such polycubes [Section 3]. Here we define ***external feature size at least k*** as follows:
   a. Every empty $1 \times 1 \times 1$ cube Q is contained in an empty $k \times k \times k$ cube $\hat{Q}$ (see Figure 5 (left)); and
   b. For every pair of edge-adjacent empty $1 \times 1 \times 1$ cubes Q_1 and Q_2, there exist empty $k \times k \times k$ cubes $\hat{Q}_1$ and $\hat{Q}_2$ containing Q_1 and Q_2 respectively, such that we can continuously slide $\hat{Q}_1$ into $\hat{Q}_2$ by axis-parallel unit slides while preserving emptyness of the intermediate $k \times k \times k$ cube and remaining within the bounding box of $\hat{Q}_1 \cup \hat{Q}_2$ (see Figure 5 (middle and right)).

As already mentioned, 2-loose sliding is a slightly stricter constraint than in the pivoting-cubes model. In our second result we build upon the techniques developed for both, the sliding-cube and the pivoting-cube models.

The second construction has a useful additional property called ***monotonicity***. Assume we start from a line or box of modules, and the goal is to assemble the desired polycube adjacent to this starting configuration. Then we move the modules in order, and once a module stops moving, we never move it again. Monotonicity implies both accessibility and the lack of extra modules. Furthermore, monotonicity tells us that the assembly process is particularly efficient to simulate with one moving robot and many passive modules: the robot simply needs to double the motion of each module (once to move the module in place, and then in reverse to get the next module).

Our algorithm descriptions focus on the case $k = 2$, as it is the simplest case where the model differs from past work, and as an ARMADAS robot can fit within a $2 \times 2 \times 2$ cube.

2 Universality with Extra Modules: 3D Printing

In this section, we will give a 2-accessible algorithm to disassemble any connected structure using extra "scaffolding" modules. More precisely, given an initial configuration T of n modules, and a line of $O(n)$ extra modules attached to it, the algorithm will reconfigure the structure into a single line of modules. The intuition behind the algorithm is to "3D print" the structure in reverse by gradually moving a sweep plane through the structure. The sweep plane is filled with modules to preserve the connectivity of the structure, allowing the modules of T to be removed one at a time. We will first describe a simpler version of the algorithm for two-dimensional structures to give intuition.

2.1 2D

Let $T \subseteq \mathbb{Z}^2$ be the shape we want to disassemble and $U \subseteq \mathbb{Z}^2$ be an axis-aligned rectangle which contains T. We will reconfigure it into a single long line L.

The main idea is to sweep U with a diagonal line of scaffolding, placing modules in front of it if they are not already present, and removing modules behind it. We use a possibly counterintuitive slope of 3 : 5 for the sweep line. As it will become clear later, it is hard to maintain connectivity of the intermediate structure with simpler slopes. In particular, a simpler horizontal, vertical or 1 : 1 diagonal slope does not allow us to "dig" a small hole in the sweep line to remove a module from behind it without either temporarily breaking connectivity or moving a module that is not 2-accessible.

Let $f(x, y) = 3x + (5 - \varepsilon)y$ for some small $\varepsilon \ll 1/n$. Our diagonal line of scaffolding will consist of all modules which are intersected by the line $f(x, y) = t$ for some t which we gradually decrease to sweep the line across. This is equivalent to sweeping a 3 : 5 slope line

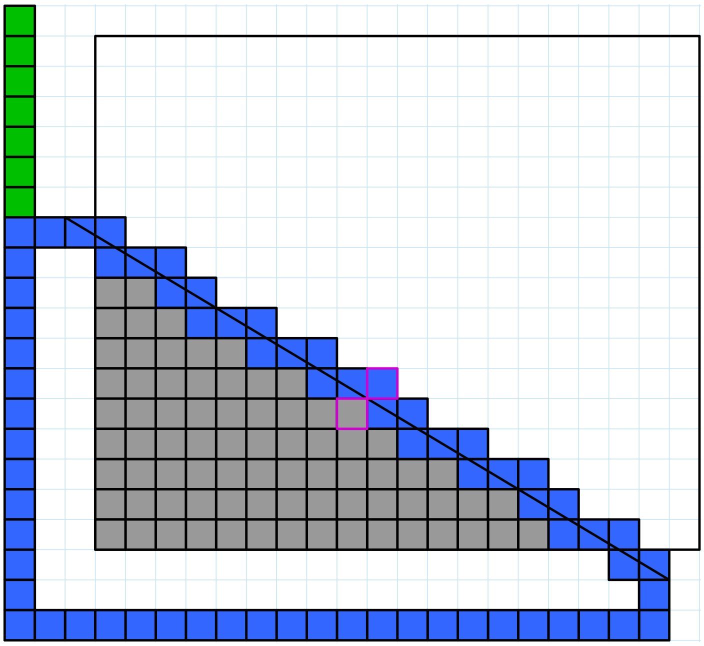


Figure 7 The 2D sweep line approach. Blue squares are part of the sweep line, gray squares are the remaining part of T that has not been deconstructed. (Not all gray squares correspond to existing modules, since T is an arbitrary connected subset.) Purple squares are the next modules to be added or removed. The outer black box is the boundary of U. The green squares are the line L where removed modules are appended.

across, and when it hits several modules simultaneously, tiebreaking in favor of processing the higher y valued ones first. We define an intermediate state R_t to be the union of the following four sets (refer to Figure 7):

- remainder of the structure $\{(x, y) \in T \mid f(x, y) \leq t\}$,
- a diagonal "sweep line" of modules $\{(x, y) \in U \mid t < f(x, y) \leq t + 8 - \varepsilon\}$,
- a path outside of U connecting the two ends of the sweep line. This is just to ensure the structure remains connected when we remove modules from the sweep line, and
- the portion of L that has been built so far: a line of modules that have been removed from T that is attached to the path outside of U.

▶ **Observation 1.** *Any 2-accessible module can slide along the exterior of R_t to the end of the line L in $O(n)$ moves.*

Using the above observation, any extra modules that are not part of R_t at time t are placed in the line L. For the rest of this section, whenever we refer to "removing" a module, we mean sliding it to the end of L.

Initially, we construct a bounding box made of the additional modules around U with an extra gap of two units. Furthermore, we connect T to this bounding box. Specifically, we build a path of modules out from the right-most top-most module of T to the upper right corner of the bounding box. This is done simply to ensure the sweep line is connected to T initially (and we can equivalently think of T as containing this additional path).

We are going to gradually decrease t, starting from $R_\infty = T$ and ending at $R_{-\infty} = \emptyset$. As we decrease t, R_t changes only when $t = f(x, y)$ or $t = f(x, y) - 8 + \varepsilon$ for some $(x, y) \in U$, which happens $|U|$ times. More specifically, a module at (x, y) is added to the sweep line

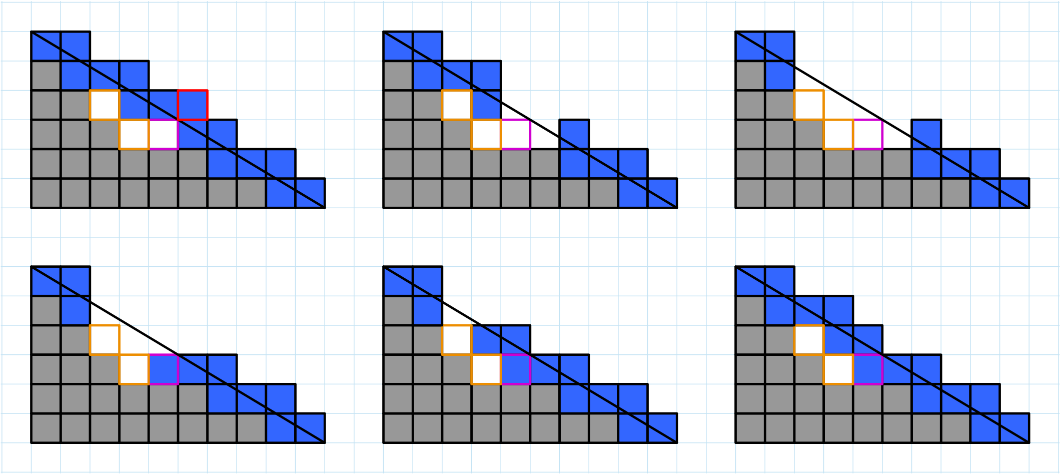


Figure 8 An example of processing a sweep-line event. Reading left to right, this shows the intermediate steps of adding and removing a module from R_t, when both cells outlined in orange are empty. The red outlined module is being removed and the purple outlined one is being added. Other cases are treated similarly; the details can be found in the full version of this paper.

when $t = f(x, y)$, and is removed from it when $t = f(x, y) - 8 + \varepsilon$. Note that, whenever the first condition is met for (x, y), the second condition is met for $(x + 1, y + 1)$. Thus, each event updating R_t will involve removing one module at $(x + 1, y + 1)$ and adding one module at (x, y). Consecutive states of R_t differ in only these two modules. We will show that we can always transition from R_t to $R_{t-\varepsilon}$ in a constant number of moves near the sweep line, plus, if necessary, a linear number of moves to take the removed modules out to the end of the line L.

The proof of the following lemma is omitted and can be found in the full version of this paper.

▶ **Lemma 2.** *There is a sequence of $O(n)$ 2-accessible moves that reconfigures R_t to $R_{t-\varepsilon}$.*

Because each dimension of U is $O(n)$, the sweep line consists of $O(n)$ modules, so we only use $O(n)$ additional modules to deconstruct T.

▶ **Theorem 3.** *Any polyomino shape T can be deconstructed into a line with 2-accessible sliding moves with the help of additional $O(n)$ modules.*

2.2 3D

Next we describe the full three-dimensional algorithm for disassembling a structure. Let $T \subseteq \mathbb{Z}^3$ be the shape we want to disassemble, and $U \subseteq \mathbb{Z}^3$ be an axis-aligned bounding box which contains T. Similarly to the 2D case, we sweep a diagonal plane of scaffolding. At each step, we add a module in front of the sweep plane if it is not already present and remove a module from behind the plane.

Define $f(x, y, z) = x + (3 - \varepsilon)y + (3 - \varepsilon^2)z$, for sufficiently small ε. Again, we define R_t to be the set of modules still present in the structure at time t, which is the union of three sets:

- the remainder of the structure: $\{(x, y, z) \in T \mid f(x, y, z) \leq t\}$,
- the diagonal sweep plane of scaffolding keeping everything connected: $\{(x, y, z) \mid t < f(x, y, z) \leq t + 4 - \varepsilon\}$,

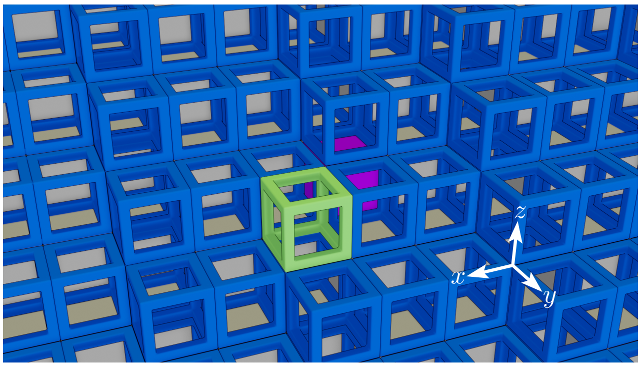


Figure 9 The state of our "3D printing" algorithm before processing an event. The scaffolding plane is shown in blue. Purple cell is at (x, y, z) and a module is to be placed in it; green module is at $(x+1, y+1, z)$ and is to be removed.

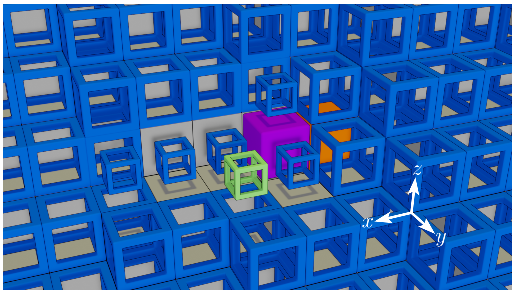


Figure 10 The solid orange module behind the purple cell is part of T. The small blue and green blocks correspond to the modules that are removed in order to access the purple cell without breaking connectivity.

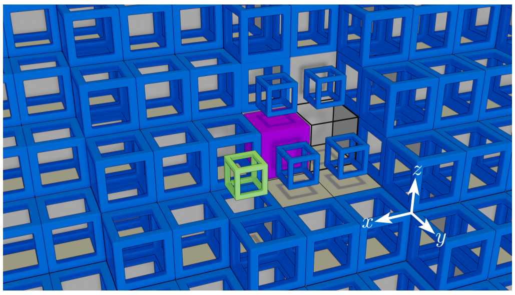


Figure 11 There is an empty cell behind the purple cell. The small blue and green blocks correspond to the modules that are removed in order to access the purple cell without breaking connectivity.

- the portion of L that has been built so far: a line of modules that have been removed from T that is attached to the path outside of U, and
- additional modules near the boundary of the sweep plane which help preserve connectivity of the sweep plane near the boundary of U: $\{(x, y, z) \mid t < f(x, y, z) \leq t + 6 - 2\varepsilon\}$, when (x, y, z) is outside of U but is within four-unit distance of the boundary of U.

Similarly to the 2D case, as the sweep plane moves, we keep the line L attached to the scaffolding. Whenever a module is removed, it can slide along this connection to the end of L to be deposited.

▶ **Lemma 4.** *There is a sequence of $O(n)$ 2-accessible moves that reconfigures R_t to $R_{t-\varepsilon}$.*

Proof. As before, we continuously decrease t. Each time R_t changes, up to one module is added to the sweep plane at (x, y, z) and the module at $(x+1, y+1, z)$ is removed from it (refer to Figure 9). Removing modules is straightforward: they can slide to the end of L. Adding modules, however, requires some care. When we need to add a module at location (x, y, z) in front of the sweep plane, we "dig" a small hole in the plane, add the new module, and then fill in the hole back.

Next we provide specific steps for processing the event $f(x, y, z) = t$. First, if a module at (x, y, z) already exists in R_t, we simply remove the module at $(x+1, y+1, z)$. When a cell (x, y, z) is not occupied and a module needs to be placed in it, we consider the following two cases, based on whether $(x-1, y, z)$ is present or not.

Case 1: cell $(x-1, y, z)$ is occupied. See Figure 10.

1. Remove $(x+i, y+1, z)$ for $i = 1, 0$, and $(x, y, z+1)$.
2. Remove $(x+i, y, z)$ for $i = 1, 2, 3$.
3. Place $(x+i, y, z)$ for $i = 0, 1, 2, 3$.
4. Place $(x, y+1, z)$ and $(x, y, z+1)$.

Case 2: cell $(x-1, y, z)$ is empty. See Figure 11.

1. Remove $(x+i, y+1, z)$ for $i = 1, 0, -1$, and $(x+i, y, z+1)$ for $i = 0, -1$.
2. Place (x, y, z).
3. Place $(x+i, y+1, z)$ and $(x+i, y, z+1)$ for $i = -1, 0$.

Note, that every module that is being removed or added in the steps above is 2-accessible by construction.

When we are outside of U, we will use a different method to add and remove modules from the sweep plane. As the modules of the scaffold that are outside of U are sufficiently removed from T, the empty cells in front of the sweep plane adjacent to them are in fact 2-accessible. Thus, we can simply add a module directly to the front of the sweep plane and remove a module directly from the back side.

Observe that in this case the connectivity of the scaffolding is preserved. Every point (x, y, z) with $f(x, y, z) < t$ that is face-adjacent to a module in the sweep plane maintains the property that at least one of the modules in the sweep plane that it is face-adjacent to is present at all points during the process. This means that if the structure was connected before this step of the algorithm, it remains connected during this step, and will still be connected afterward. ◀

Each dimension of U is $O(n)$. However, because the sweep plane is two-dimensional, this means that our algorithm requires $O(n^2)$ additional modules. We can reduce the number of modules needed by reducing the size of U. Instead of using a full bounding box, we construct U as follows. Consider a discretization of the lattice grid into $2 \times 2 \times 2$ metacells. For each module in T, consider the line passing through the center of the module perpendicular to the sweep plane. We let U be the intersection of the bounding box of T and union of all metacells that are pierced by any such line.

Because we maintain the 4-module offset of the boundary of U be padded with an extra layer of modules in the front of the sweep plane, our sweep plane stays connected at each step of the reconfiguration regardless of the shape of the boundary of U. By construction of U, as it is made of $2 \times 2 \times 2$ metacells, the empty cells, adjacent to the front of the sweep plane, within distance 4 of the boundary of U are necessarily 2-accessible. Since the original algorithm does not depend on the specific shape of U but only the fact that the edge modules are 2-accessible, the correctness of the algorithm follows. The size of the intersection of our sweep plane with U is $O(n)$ by construction, so we now only need $O(n)$ additional modules.

▶ **Theorem 5.** *Any polycube shape T can be deconstructed into a line with 2-accessible sliding moves with the help of additional $O(n)$ modules.*

2.3 Impossibility of 3-loose algorithm

In this section we describe a structure which cannot be reconfigured into a line with any 3-loose algorithm. We will first describe a 2D version and then generalize to a 3D version.

Consider the structure in Figure 12. It consists of a hollow 5×5 square, as well as the four modules in the corners of a concentric 3×3 square (marked in red). Suppose we can disassemble this structure. Consider the first time we place or remove a module in the central

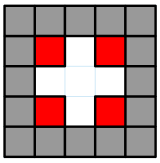


Figure 12 2D configuration that cannot be disassembled with a 3-loose algorithm. The red blocks cannot be removed until at least one other red block is removed.

3×3 square. We cannot place a module anywhere inside the 3×3 square, because it would be blocked by red modules. So we must remove a red module. Then this red module must the only module in some 3×3 square. It is not hard to show that this module cannot have any neighbors, which violates connectivity.

In 3 dimensions, we instead use a hollow $5 \times 5 \times 5$ with 8 modules in the corners of a $3 \times 3 \times 3$ cube. We can use a similar argument to show that no module can be placed or removed inside the central $3 \times 3 \times 3$ cube.

3 Universality without Extra Modules

In this section we show how to reconfigure a modular robot, whose shape satisfies the property of external feature size at least 2, into a line. The resulting reconfiguration will consist of 2-accessible and monotone moves (every sliding move will have a sufficient amount of empty space around it, and each module moves only once in a continuous motion from its starting to its target position). Reversing the sequence moves results in a schedule for the construction of the given structure starting from a line of modules.

The approach we take is similar to the one for the pivoting-cube model, presented in [34] and later further developed in [17, 24]. We deconstruct the robot structure slice by slice (parallel to the horizontal plane), ensuring the connectivity of the intermediate structure and the invariant of external feature size at least 2. In our algorithm, the choice of which slice to deconstruct is made in the same way as in [34, 17, 24]. The difference of our algorithm with these approaches, and its main difficulty, lies in deciding in which order to deconstruct (partially or fully) a given slice, such that the requirements on the connectivity and the external feature size are satisfied for the intermediate structures.

Slice graph. We adapt the definition of a slice graph from [34]. Let $V_{z=k}$ denote the subset of nodes of the lattice with the z-coordinate fixed to k. Let V_R denote the subset of nodes of the lattice which contain a module in an intermediate configuration R. Define a ***slice graph*** $G_s(V_s, E_s)$ of R in the following way. The nodes V_s of G_s correspond to the maximally connected components of the modules with the same z-coordinate, such that these components have at least one module adjacent to the empty outer space. That is, each connected component of the induced graphs on $V_R \cap V_{z=k}$ (for all k), that has a module adjacent to and empty cell of an outer empty space, is added to V_s. Two nodes in V_s are connected with an edge if the corresponding slices contain (vertically) adjacent modules (see Figure 13).

A slice $s \in V_s$ is ***locally maximal*** (***minimal***), if all the adjacent slices of s in G_s contain modules with only lower (higher) z-coordinate than s.

Outline of the algorithm. Consider the initial configuration T with external features of size at least 2. Let s_0 be the slice in G_s with the globally maximum z-coordinate. Select an arbitrary module m_0 from s_0. Define a configuration L to be formed by a vertical line of n

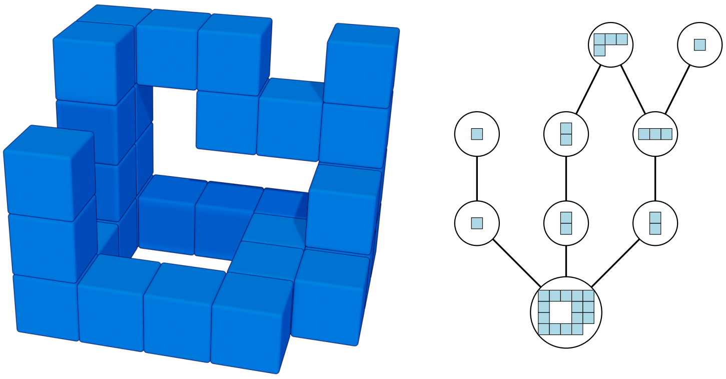


Figure 13 A robot configuration on the left, and the corresponding slice graph on the right.

modules with m_0 as the bottom-most module, i.e., all the other modules have the same x- and y-coordinates as m_0, and have z-coordinates larger than z-coordinate of m_0. We will prove that we can reconfigure from T to L in $O(n^2)$ number of moves.

We iteratively consider a locally extremal outer-surface slice s other than s_0 which is not needed for the connectivity of the current slice graph, and move its modules to the top of the line above m_0. There are two cases: either removing s disconnects the structure or it does not. If removal of s does not disconnect the robot structure, we simply completely deconstruct s and proceed with the next locally extremal outer-surface slice.

The harder case is when s disconnects the structure. In this case, s must lie on the boundary of a ***void***, an enclosed empty space. Furthermore, withing this void there must be some modules attached to s which are connected to the rest of the structure only through s. Then, we partially disassemble s while ensuring the connectivity of the intermediate structure, preserving the property of external feature size 2, and connecting the void to the outer empty space.

To summarize, our algorithm consists of the following steps:

1. Construct the slice graph H;
2. Using H, select a locally extremal outer-surface slice s to deconstruct;
3. Deconstruct s fully (if connectivity is maintained) or partially, update H accordingly;
4. Repeat steps 2 and 3 until L has been constructed.

Step 1: Constructing the slice graph. The definition of a slice graph from [34] does not require that at least one module from each slice is on the outer surface (adjacent to an empty cell of the outer empty space). Their result relies on the assumption that any void, a structure to be deconstructed may have, is convex. Thus, all the slices of the structure have modules that lie on its outer surface. In particular, all the modules of any locally extremal slice lie on the outer surface. In our case, however, the structure may have non-convex voids, and thus some slices (including locally extremal ones) may be “trapped” inside a void. We can only move modules that are on the outer surface of the structure. Thus, to facilitate the choice of a slice to deconstruct, we only maintain the slices in H that are not trapped in a void. That is, we require the nodes of G_s to correspond to the slices with at least one module on the outer surface of the structure.

Step 2: Selecting a slice to deconstruct. We require the slice we select for deconstruction to satisfy two properties: (1) its removal should not disconnect the remaining structure, and (2) it should be locally extremal in the z-direction to give enough space for the modules to move on its top (or bottom). Note that naively selecting a slice that corresponds to a non-cut node in H may violate property (2), and selecting a top-most or bottom-most slice may violate property (1).

Slice s_0, containing the node m_0 (the end of the line L) will be the last one to be deconstructed. If there are no other nodes left in H, we deconstruct s_0. Otherwise, consider the degrees of the nodes in V_s. If there is a node $s \neq s_0$ with degree 1, we select it for deconstruction. Otherwise, select a locally extremal slice s that is furthest away from s_0 in H. Removing s from H does not disconnect the graph. Indeed, otherwise, after the removal of s from H, the component not containing s_0 would have at least two locally extremal slices. At least one of these slices existed in H before removing s, and is further away from s_0 than s. Thus, the lemma follows.

▶ **Lemma 6.** *If H has more than one slice, there is always a locally extremal slice $s \neq s_0$ in H whose removal does not break the connectivity of H.*

Step 3: Deconstructing a locally extremal slice. Consider a set of empty cells V_e of the empty outer-space component, adjacent to the modules of the structure. Connect two face-adjacent nodes of V_e with an edge, if the corresponding move is 2-loose or is a convex transition consisting of two 2-loose slide steps. Observe, that by definition of the external feature size property, graph G_e is connected. And furthermore, any two nodes of V_e corresponding to face-adjacent cells have an edge connecting them.

▶ **Observation 7.** *Consider a structure T that has external feature size at least 2, and a top-most node m_0. Consider an additional module m anywhere on the outer surface of T. Then m can move to the top of m_0 with loose-sliding moves.*

Let s be our locally extremal slice, and let z_s be the value of the z-coordinate of the modules in s. By symmetry, suppose s has no adjacent modules in the positive z direction, thus all the modules adjacent to the modules of s have z-coordinate equal to $z_s - 1$.

Following the notation of [24], we assign colors to the modules of s, depending on whether they have adjacent modules below or not (refer to Figure 14). A module m in s is colored:

- ***red*** if the cell below m is occupied by some module m', and the slice containing m' is adjacent to the outer empty component,
- ***green*** if the cell below m is empty and belongs to the outer component of R,
- ***blue*** if the cell below m is empty and does not belong to an outer empty component (i.e., it is a void), and
- ***orange*** if the cell below m is occupied by some module m', and the slice containing m' is not adjacent to the outer empty component (i.e., it is enclosed in a void).

We consider two cases, whether the removal of s from the modular structure breaks its connectivity, or not. If the removal of s does not break the connectivity of the structure, we move all the modules of s one by one to the end of L. The proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 8.** *Suppose that removing a locally extremal slice s of H does not disconnect the modular structure. Then s can be completely deconstructed, and the resulting structure maintains the external-feature-size-2 property.*

Consider now the case when removing s disconnects the modular structure. This is the case if the structure contains a void, and the void has trapped modules inside, attached to s. In this case we no longer can completely deconstruct s, and our goal instead is to cut a hole into at least one void without breaking connectivity and while preserving the external feature size of at least 2.

To do so, we define a directed graph G_s on the modules of s. All pairs of face-adjacent modules in s are connected with a directed edge. Using G_s, we select which modules from s are to be removed at this step of the algorithm. In particular, all the modules of s that are reachable from orange modules in G_s remain in the slice. All the other modules are deconstructed. Below, we argue that this preserves the external-feature-size-2 property, and eliminates at least one void. Afterwards, we update the slice graph H by adding to it new slices that were trapped inside the eliminated void(s), and proceed to the next iteration of the algorithm.

We construct G_s in the following way. Add all modules of s as nodes in G_s. We call a module of s ***reddish*** if it has a module in the layer below it (and so is either red or orange in color), and ***bluish*** if it does not (and thus is either green or blue). Add the following edges to G_s (refer to Figure 14):

- For two face-adjacent modules with the same y-coordinate, add an edge directed to the left.
- For two face-adjacent modules with the same x-coordinate, add an edge directed upward.
- For a triplet of bluish-bluish-reddish modules oriented left-to-right (or top-to-bottom), add a directed edge from the left-most (top-most) bluish module to the reddish module.

▶ **Lemma 9.** *If there are modules present at both locations a and b, and a and b are at most 2 units apart in each dimension, then there is a path in the adjacency graph of s from a to b within the bounding box spanned by a and b.*

Proof. Suppose this is the minimal such counterexample. Consider a sequence of empty cells within this bounding box which starts adjacent to a and ends adjacent to b. (If there are no such adjacent empty cells to a and b, then this example was not minimal: we could have instead taken one of the occupied cells adjacent to a or b and gotten a smaller counterexample). Now we know that each of these cells is contained in an empty $2 \times 2 \times 2$ region. Consider this sequence of $2 \times 2 \times 2$ regions. It must be possible to slide between each adjacent pair of modules of empty space because we have external feature size 2. However, this is not going to be possible because we cannot fit an empty $2 \times 2 \times 2$ cube between a and b. ◀

Lemma 9 is the key reason why we use external feature size 2. It follows directly that none of the $1 \times 1 \times 3$, $1 \times 2 \times 2$, and $1 \times 2 \times 3$ forbidden patterns in [34] will be present in our structure, and also ensures that every module that is accessible is 2-accessible.

The proof of the following lemma can be found in the full version of this paper.

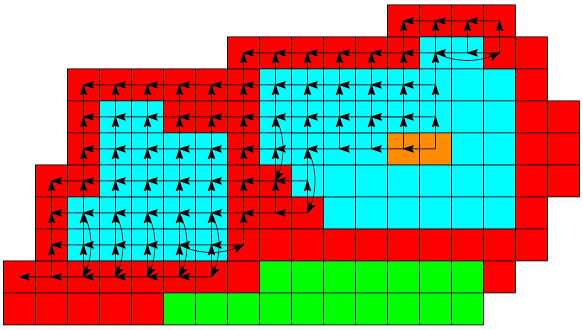


Figure 14 Extremal slice with its modules colored according to the adjacency with the nodes in the slice below. Component of the directed graph G_s of modules reachable from orange modules, used in Lemma 9.

▶ **Lemma 10.** *Every minimal path between any module and a blueish module is bitonic in both x and y: it never goes right after going left and it never goes down after going up.*

▶ **Corollary 11.** *All blueish cells reachable from an orange cell are either strictly above it or strictly to the left of it.*

Proof. Consider the shortest path to some cell. By construction of G_s, the first move must be to the left or up. If it has non-zero leftward component, then it can never go right. If it has non-zero upward component, then it can never go down. ◀

▶ **Corollary 12.** *There is an orange module such that a 2×2 square directly below and to the right of it is unreachable.*

▶ **Corollary 13.** *If two modules a and b on the same row or column are reachable from an orange module, then every module in between also is.*

▶ **Lemma 14.** *Any two $2 \times 2 \times 2$ edge-adjacent empty regions can be slid together until they overlap in at least one cell.*

Proof. Suppose for contradiction that we have two such edge-adjacent $2 \times 2 \times 2$ empty cubes that cannot be slid closer together. This means there is a $2 \times 2 \times 1$ pattern of diagonally opposite modules with empty space at the other two corners. Suppose we created such a pattern containing one of the modules we removed. Because the region we removed is convex by Corollary 13, it must contain a module from the layer below. In particular, the removed cell from this layer must be red, and there must be an adjacent bluish cell that is not removed. However, in our directed graph there is always an edge from a blue cell to an adjacent red cell, so this is impossible. ◀

▶ **Lemma 15.** *Every removed blue cell is part of a 2×2 removed square of blue cells.*

Proof. Suppose that a removed blue module is not part of a blue 2×2 disassembled square. Then it must have two neighbors on opposite sides that are each either reddish or unremoved blue. It cannot have two unremoved blue neighbors on each side because of the convexity of the removed area the algorithm generates. It cannot have two reddish neighbors on each side because that would violate the external feature size beforehand. Finally, it cannot have a unremoved blue pixel on one side and a reddish on the other side because the rules that generate the graph would have connected the blue to the reddish neighbor. ◀

▶ **Lemma 16.** *For any pair of overlapping $2 \times 2 \times 2$ empty cubes, at least one of which was created by removing a module from this layer, we can continuously slide one into the other while preserving emptiness.*

Proof. Because every removed cube is part of a 2×2 removed square of blue pixels by Lemma 15, every new empty space is part of some empty $2 \times 2 \times 2$ cube.

Let the deconstructed layer have $z = z_s$. Consider two overlapping $2 \times 2 \times 2$ empty cubes $\hat{Q}_1$ and $\hat{Q}_2$ after partially deconstructing s. If both cubes reside in $z = z_s - 1$ and $z = z_s$, then we consider cubes $\hat{Q}'_1$ and $\hat{Q}'_2$ that are the projection of $\hat{Q}_1$ and $\hat{Q}_2$ to layers $z = z_s - 1$ and $z = z_s - 2$. Due to the external feature size, $\hat{Q}'_1$ and $\hat{Q}'_2$ must both have been empty and it must have been possible to slide between them. We project that motion back up to $\hat{Q}_1$ and $\hat{Q}_2$ and get a motion that sweeps between the two cubes.

If both $\hat{Q}_1$ and $\hat{Q}_2$ reside in $z = z_s$ and $z = z_s + 1$, we consider cases based on the difference in their x and y coordinates. If they share the same x or y coordinates, then there is a direct motion between them maintaining the shared coordinate. If one of them is larger in both x and y, we know that their bounding box is empty by the convexity of the removed area, so any motion suffices. If one is smaller in x and larger in y, then we slide it first in the increasing x direction, and then in the decreasing y direction.

If, w.l.o.g., $\hat{Q}_1$ resides in $z = z_s$ and $z = z_s - 1$, and $\hat{Q}_2$ in $z = z_s$ and $z = z_s + 1$, then we slide $\hat{Q}_1$ up one layer first, then proceed with a similar argument to the case above. ◀

By Lemmas 14 and 16, everything must still have external feature size at least 2.

Finally, by Corollary 12, we have made a hole in slice s, which allows us to access modules inside the void. We have made progress by opening up at least one void. Since at each step of the algorithm we either clear one slice or open at least one void, and there are finitely many voids in the structure, we must eventually terminate with the entire shape deconstructed. Because each module needs to move a distance of at most $O(n)$, and each module is only moved once, the total number of moves is at most $O(n^2)$.

▶ **Theorem 17.** *A modular structure with external-feature size at least 2 can be reconfigured into a line in $O(n^2)$ 2-accessible moves.*

4 Open Problems

The main open question is how many extra modules we need to achieve universal reconfiguration in the k-accessible sliding-cubes model. Our solution in Section 2 uses $\Theta(n)$ extra modules in the worst case. But plausibly $O(1)$ extra modules suffice via a different method; such a result was previously obtained for the pivoting-squares model [3].

Another open problem is whether universal reconfiguration is possible for the k-loose sliding-cubes model – without the k-accessible constraint. In particular, this would require handling complex 3D configurations with no movable modules on the outside [26]. Previous solutions in the sliding-cubes model [27] are not loose.

References

1 Zachary Abel, Hugo Akitaya, Matias Korman, Scott Kominers, and Frederick Stock. A universal in-place reconfiguration algorithm for sliding cube-shaped robots in quadratic time. To appear at the 40th International Symposium on Computational Geometry (SoCG), 2024.

2 Zachary Abel and Scott D. Kominers. Pushing hypercubes around, 2008. arXiv:0802.3414v2.

3 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmović, Robin Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

4 Hugo A. Akitaya, Erik D. Demaine, Andrei Gonczi, Della H. Hendrickson, Adam Hesterberg, Matias Korman, Oliver Korten, Jayson Lynch, Irene Parada, and Vera Sacristán. Characterizing universal reconfigurability of modular pivoting robots. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.SoCG.2021.10.

5 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Proc. 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:19, 2022. doi:10.4230/LIPIcs.SWAT.2022.4.

6 Greg Aloupis, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Robin Flatland, John Iacono, and Stefanie Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Computational Geometry: Theory and Applications*, 46(8):917–928, October 2013. doi:10.1016/j.comgeo.2013.03.004.

7 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Dania El-Khechen, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Val Pinciu, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Realistic reconfiguration of crystalline (and telecube) robots. In *Proceedings of the 8th International Workshop on the Algorithmic Foundations of Robotics (WAFR 2008)*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 433–447, Guanajuato, México, December 7–9 2008.

8 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *Computational Geometry: Theory and Applications*, 42(6–7):652–663, August 2009.

9 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *Proceedings of the 19th Annual International Symposium on Algorithms and Computation (ISAAC 2008)*, pages 342–353, Gold Coast, Australia, December 15–17 2008.

10 Greg Aloupis, Sébastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Val Pinciu, Suneeta Ramaswami, Vera Sacristán, and Stefanie Wuhrer. Efficient constant-velocity reconfiguration of crystalline robots. *Robotica*, 29(1):59–71, 2011. doi:10.1017/S026357471000072X.

11 Byoung Kwon An. EM-Cube: Cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3155, 2008. doi:10.1109/ROBOT.2008.4543690.

12 Nora Ayanian, Paul J. White, Ádám Hálász, Mark Yim, and Vijay Kumar. Stochastic control for self-assembly of XBots. In *Proc. ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE)*, pages 1169–1176, 2008. doi:10.1115/DETC2008-49535.

13 Nadia M. Benbernou. Geometric algorithms for reconfigurable structures. PhD thesis, Massachusetts Institute of Technology, 2011.

14 Kenneth C. Cheung and Neil Gershenfeld. Reversibly assembled cellular composite materials. *Science*, 341(6151):1219–1221, 2013.

15 Chih-Jung Chiang and Gregory S. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10:91–106, 2001. doi:10.1023/A:1026552720914.

16 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006. doi:10.1007/s00373-005-0640-1.

17 Daniel Feshbach and Cynthia Sung. Reconfiguring non-convex holes in pivoting modular cube robots. *IEEE Robotics and Automation Letters*, 6(4):6701–6708, 2021. doi:10.1109/LRA.2021.3095030.

18 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*, pages 2460–2467, 2003. doi:10.1109/IROS.2003.1249239.

19 Christine E. Gregg, Damiana Catanoso, Olivia Irene B. Formoso, Irina Kostitsyna, Megan E. Ochalek, Taiwo J. Olatunde, In Won Park, Frank M. Sebastianelli, Elizabeth M. Taylor, Greenfield T. Trinh, and Kenneth C. Cheung. Ultralight, strong, and self-reprogrammable mechanical metamaterials. *Science Robotics*, 9(86):eadi2746, 2024. doi:10.1126/scirobotics.adi2746.

20 Christine E. Gregg, Joseph H. Kim, and Kenneth C. Cheung. Ultra-light and scalable composite lattice materials. *Advanced Engineering Materials*, 20(9):1800213, 2018.

21 Kazuo Hosokawa, Takehito Tsujimori, Teruo Fujii, Hayato Kaetsu, Hajime Asama, Yoji Kuroda, and Isao Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 2858–2863, 1998. doi:10.1109/ROBOT.1998.680616.

22 Benjamin Jenett, Amira Abdel-Rahman, Kenneth Cheung, and Neil Gershenfeld. Material–Robot System for Assembly of Discrete Cellular Structures. *IEEE Robotics and Automation Letters*, 4(4):4019–4026, October 2019. doi:10.1109/LRA.2019.2930486.

23 Irina Kostitsyna, Tim Ophelders, Irene Parada, Tom Peters, Willem Sonke, and Bettina Speckmann. Optimal in-place compaction of sliding cubes. Submission to SWAT, 2024.

24 Lloyd E. Lo-Wong. Reconfiguration of pivoting modular robots. MSc thesis, Technical University Eindhoven, 2021.

25 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019. doi:10.1016/j.jcss.2018.12.001.

26 Tillmann Miltzow, Irene Parada, Willem Sonke, Bettina Speckmann, and Jules Wulms. Hiding sliding cubes: Why reconfiguring modular robots is not easy. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, volume LIPIcs 164, pages 78:1–78:5, 2020. doi:10.4230/LIPIcs.SoCG.2020.78.

27 Joel Moreno. In-place reconfiguration of lattice-based modular robots. Bachelor's thesis, Universitat Politècnica de Catalunya, 2019.

28 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proc. 36th European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020.

29 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. doi:10.1007/s10514-021-09977-6.

30 In-Won Park, Damiana Catanoso, Olivia Formoso, Christine Gregg, Megan Ochalek, Taiwo Olatunde, Frank Sebastianelli, Pascal Spino, Elizabeth Taylor, Greenfield Trinh, and Kenneth Cheung. Soll-e: A module transport and placement robot for autonomous assembly of discrete lattice structures. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

31 John Romanishin, Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, pages 4288–4295, Tokyo, Japan, November 2013. IEEE. doi:10.1109/IROS.2013.6696971.

32 Daniela Rus and Marsette Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1726–1733, 2000. doi:10.1109/ROBOT.2000.844845.

33 John W. Suh, Samuel B. Homans, and Mark Yim. Telecubes: mechanical design of a module for self-reconfigurable robotics. In *Proc. 2002 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4095–4101, 2002. doi:10.1109/ROBOT.2002.1014385.

34 Cynthia Sung, James Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *Proc. 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1933–1940, 2015. doi:10.1109/ICRA.2015.7139451.

35 Yuzuru Terada and Satoshi Murata. Automatic Assembly System for Modular Structure. In *Proceedings of the 22nd International Symposium on Automation and Robotics in Construction*, Ferrara, Italy, September 2005. doi:10.22260/ISARC2005/0028.

36 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007. doi:10.1109/MRA.2007.339623.

Solving a Family Of Multivariate Optimization and Decision Problems on Classes of Bounded Expansion

Daniel Mock
RWTH Aachen University, Germany

Peter Rossmanith
RWTH Aachen University, Germany

Abstract

For some time, it has been known that the model checking problem for first-order formulas is fixed-parameter tractable on nowhere dense graph classes, so we shall ask in which direction there is space for improvements. One of the possible directions is to go beyond first-order formulas: Augmenting first-order logic with general counting quantifiers increases the expressiveness by far, but makes the model checking problem hard even on graphs of bounded tree-depth. The picture is different if we allow only "simple" – but arbitrarily nested – counting terms of the form $\#y\,\varphi(\bar{x}, y) > N$. Even then, only approximate model checking is possible on graph classes of bounded expansion. Here, the largest known logic fragment, on which exact model checking is still fpt, consists of formulas of the form $\exists x_1 \ldots \exists x_k \#y\,\varphi(\bar{x}, y) > N$, where $\varphi(\bar{x}, y)$ is a first-order formula without counting terms. An example of a problem that can be expressed in this way is partial dominating set: Are there k vertices that dominate at least a given number of vertices in the graph? The complexity of the same problem is open if you replace *at least* with *exactly*. Likewise, the complexity of "are there k vertices that dominate at least half of the blue and half of the red vertices?" is also open. We answer both questions by providing an fpt algorithm that solves the model checking problem for formulas of the more general form $\psi \equiv \exists x_1 \ldots \exists x_k\, P(\#y\,\varphi_1(\bar{x}, y), \ldots, \#y\,\varphi_\ell(\bar{x}, y))$, where P is an arbitrary polynomially computable predicate on numbers. The running time is $f(|\psi|)n^{\ell+1}\,\mathrm{polylog}(n)$ on graph classes of bounded expansion. Under SETH, this running time is tight up to almost linear factor.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Logic; Mathematics of computing → Graph algorithms

Keywords and phrases bounded expansion, parameterized algorithms, sparsity, counting logic, dominating set, model checking, multivariate optimization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.35

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Science Foundation) – RO 927/15-2.

1 Introduction

Many problems on finite structures, in particular on graphs, can be expressed by logical formulas and then solved by meta-algorithms for the model checking problem. One of the best known meta-algorithms is due to Courcelle that solves all problems that can be expressed in monadic second order logic on graph classes with bounded tree-width in linear fpt when parameterized by length of the formula [3], that is, in time $f(|\varphi|)n$ where n is the size of the input graph for some function f. Taking a less expressive logic, for example MSO_1 instead of full MSO, you can solve the model checking problem on larger classes of graphs, i.e., on bounded clique width [4]. Unfortunately, there is some evidence that it is unlikely we can extend these results beyond bounded tree-width, resp. bounded clique-width [20, 14].

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 35; pp. 35:1–35:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

First-order logic (FO) is still less expressive, but powerful enough to model many decision problems. For example, the dominating set problem can be expressed by the FO-formula

$$\psi \equiv \exists x_1 \dots \exists x_k \forall y \bigvee_{i=1}^{k} (y = x_i \vee y \sim x_i),$$

which says "are there vertices $x_1, \dots, x_k$ such that every vertex y is one of them or adjacent to one of them?" Viewing the dominating set problem as parameterized problem, where the solution size k is the parameter, the parameter for for the model checking problem beocomes the length of the fomula $|\varphi|$. Many other problems like this, for example independent set or chordal deletion, can be modeled in FO-logic, making FO model checking a powerful tool to solve parameterized problems. As FO is less expressive than MSO, it is not surprising that efficient model checking algorithms exist beyond bounded tree-width. There is a series of results proving fpt running times on bounded degree graphs, planar graphs, minor-closed classes, topological minor-closed classes, locally bounded tree-width and many more [26, 13, 6, 18]. All these graph classes are generalized by *bounded expansion* or *nowhere dense* classes which are part of the beautiful sparsity theory introduced by Nešetřil and Ossona de Mendez [24]. Dvořák, Kráľ, and Thomas [11] showed that FO model checking is linear fpt on bounded expansion and Grohe, Kreutzer, and Siebertz [16] showed an almost linear fpt run time on nowhere dense graph classes (the run time is $f(|\varphi|)n^{1+o(1)}$ for some function f depending on the graph class). The latter result is in some sense final for sparse graph classes: If a graph class is monotone, that is, closed under taking subgraphs, then FO model checking is fpt on this class iff the class is nowhere dense. One recent development that reaches beyond sparsity is the result by Dreier, Mählmann, and Siebertz who look at *structurally* sparse graph classes [7].

If we turn from decision problems to optimization problems, the picture changes again. In some cases an optimization problem can be solved by using the underlying decision problem as a subroutine. For example, we can easily find the smallest dominating set for a graph by trying all k's. On the other hand if we fix k and ask what is the best partial dominating set with k vertices, i.e., how many vertices can be dominated by k vertices, a self-reduction seems to be impossible. For such questions, we need a more expressive logic. Kuske and Schweikardt introduced the logic FOC($\mathbf{P}$), which is FO with powerful counting extensions [21] and showed that its model checking problem is fpt on classes of bounded degree. However, FOC($\mathbf{P}$) is so expressive that its model checking problem becomes hard on trees of bounded depth. This stays true even for the fragment FO($\{>0\}$), which is still very expressive, but at least allows for an *approximative* model checking algorithm in fpt time [10] on classes of bounded expansion; the algorithms gives the correct answer or responds with "I do not know". The latter answer is allowed only if small pertubations in the number symbols of the formula make it both satisfied and unsatisfied.

The most general type of counting formulas known so far that can efficiently and exactly be evaluated on graph classes of bounded expansion are of the form

$$\exists x_1 \dots \exists x_k \, \#y \, \varphi(y, x_1, \dots, x_k) > N, \tag{1}$$

where N is a number that can depend on the order of the graph [8]. The semantics of the counting term $\#y \, \varphi(y, u_1, \dots, u_k)$ for $u_i \in V(G)$ is the number of vertices $v \in V(G)$ with $G \models \varphi(v, u_1, \dots, u_k)$. In this way, partial dominating set can be expressed as $\exists x_1 \dots \exists x_k \, \#y \bigvee_{i=1,\dots,k}(y = x_i \vee y \sim x_i) > N$, where $u \sim v$ denotes adjacency in the graph. Instead of graphs we consider colored graphs, where nodes can have colors. In a formula we talk about colors by using a unary predicate for each color. In that way we can express questions like "are there k nodes dominating at least N of the blue nodes?" A corresponding formula might look like

$$\exists x_1 \dots \exists x_k \; \#y \bigvee_{i=1}^{k} \big(blue(y) \wedge (y = x_i \vee y \sim x_i)\big) > N. \tag{2}$$

While we know that formulas like that can be evaluated efficiently on classes of bounded expansion, as it is of the form of (1), not much is known how much further we can extend the fragment of counting formulas beyond (1) without losing fixed parameter tractability. One might expect that changing "$> N$" into "$= N$" in (2) should not make the problem W-hard, yet neither is such a result known, nor does it follow from a simple self-reducibility argument such as using (2) and binary search. Similarly, asking for k vertices that simultaneously dominate at least N blue and N red vertices seems not fundamentally harder than the same problem with only one color class, but cannot be expressed in the form of (1) either.

For the sake of completeness, we mention two other fragments, which are orthogonal to the ones discussed before. The fragment $\mathrm{FOC}_1(\mathbf{P})$ of $\mathrm{FOC}(\mathbf{P})$ introduced by Grohe and Schweikardt extends FO by allowing to formulate cardinality constraints on counting terms that have at most one free variable. The model checking problem for $\mathrm{FOC}_1(\mathbf{P})$ is fpt on nowhere dense classes [17]. For bounded expansion classes, Toruńczyk introduced a stronger query language (also orthogonal to ours) extending FO logic by aggregation over semirings [27].

Algorithmic Results and Techniques. The main result (Theorem 6) of this paper is to answer these questions affirmatively and to go significantly further. We develop an algorithm that evaluates formulas of the form

$$\psi \equiv \exists x_1 \dots \exists x_k \, P\big(\#y \, \varphi_1(x_1, \dots, x_k, y), \dots, \#y \, \varphi_\ell(x_1, \dots, x_k, y)\big), \tag{3}$$

where P is an arbitrary polynomially computable predicate on numbers and the φ_i's are FO formulas. If P can be evaluated in constant or at least in $g(\ell)$ time[1] (in the uniform cost model for some function g) then evaluating ψ can be done in time $f(|\psi|)n^{\ell+1}\operatorname{polylog}(n)$ for some function f on graph classes of bounded expansion. If P is a boolean combination of $\geq$- or $\leq$-comparisons to constants, then the running time of the model checking problem improves to $f(|\psi|)n^{\ell}\operatorname{polylog}(n)$ (see Lemma 11). This subsumes the result of [10, Theorem 1.2] up to polylogarithmic factors. Later, we will extend the result to counting tuples with the counting quantifier, which means that $\#y$ can be replaced by $\#(y_1, \dots, y_p)$ in formula 3.

Note that formulas of the form (3) are quite expressive. They capture many multivariate optimization problems where you look for k vertices that optimize a possibly quite complicated target function. For example, you might want to find a set of k vertices U such that every other vertex is reachable from U by a path of length at most ten. You pay a penalty for each vertex v proportional to the distance of v to U. Finding the k vertices with the lowest penalty is fpt on every graph class of bounded expansion because the problem can be expressed by a formula of the form (3). Moreover, the algorithm will not depend on the graph class (but its running time will). In a multivariate optimization problem we can also find witnesses for each Pareto-optimal solution size. For example, if we look for k vertices that dominate as many red and as many blue vertices as possible, there might exist $\Omega(n)$ many different Pareto-optimal ways.

[1] Technically it is sufficient that there is a pseudo-polynomial fpt algorithm that evaluates $P(n_1, \dots, n_\ell)$ in time $f(\ell)(n_1 + \dots + n_\ell)^{O(1)}$ in order to achieve an fpt running time for evaluating ψ.

In a nutshell, the results about formulas of the form (1) were proved as follows (see [10] for more details): First, the innermost existential and universal quantifiers of the input formula are treated with quantifier-elimination, resulting in a simpler formula. In this process, the graph is augmented with additional colored edges while still being part of (another) class of bounded expansion. Then, a counting term $\#y\,\varphi(y\bar{x})$, where φ is now quantifier-free, is replaced in a sequence of transformations by a sum of simpler counting terms. These are simple enough to be directly evaluated, and the evaluation is represented as a family of vertex weight functions. Then, the original task is reduced to finding a certain kind of induced subgraph of maximal weight (which is described by one of the quantifier-free formulas). Then one can use low treedepth colorings to reduce the problem to the case of bounded treedepth and use LinEMSOL optimization [4] as a black box to find the result.

This approach breaks down at several places if we want to evaluate a formula of the more complicated form (3). For example, LinEMSOL optimization always finds the biggest weighted solution and cannot be used to establish the existence of a given weight. Moreover, it can handle only univariate weights. In order to prove our main result, we use dynamic programming on the treedepth decomposition using vectors of weights. This is achieved by formulating the problem of finding a subgraph described by a quantifier-free formula with constraints on the multivariate weights as an *MSO-evaluation problem*. This means that the problem can be expressed as a homomorphism that maps the satisfying assignments of the MSO-formula into a semiring. After applying treedepth colorings as in [10] those problems can be evaluated quickly on graphs of bounded treedepth by the result of Courcelle and Mosbah [2].

Lower Bounds. The second result is a conditional lower bound for formulas of the form (3) (Theorem 18). Under SETH we cannot evaluate formulas of the restricted form

$$\exists x_1 \ldots \exists x_k \bigwedge_{i=1}^{\ell} \#y\,\varphi_i(x_1,\ldots,x_k,y) = N_i,$$

on star forests[2] where the φ_i's are quantifier-free formulas and the N_i's are numbers, in time $f(k+\ell)n^{\ell-\varepsilon}$ for any $\varepsilon > 0$ and function f. We reduce from the k-Sum problem, using a tight conditional lower bound on dense k-Sum instances under SETH [1]. The upper and lower bounds given by Theorems 6 and 18 are tight up to an almost linear factor in n.

In Theorem 20 we show that introducing an additional universal quantifier between the existential quantifiers and the counting quantifier makes evaluating those formulas on forests of depth 2 W[1]-hard. The same holds if one has two nested counting quantifiers.

2 Preliminaries

We write $\mathbf{Z}$ and $\mathbf{N}$ for the integers and natural numbers (including 0), $[k] = \{1,\ldots,k\}$.

Graphs. We consider labeled graphs $G = (V, E, P_1, \ldots, P_m)$ where V is the vertex set, E the edge set and $P_1, \ldots, P_m \subseteq V$ the labels of G. The order $|G|$ of G equals $|V|$. The size $\|G\|$ of G is $|V| + |E| + |P_1| + \cdots + |P_m|$. We often write $V(G)$ and $E(G)$ for the vertex and edge sets of G. Unless stated otherwise, graphs are undirected. An *expansion* G' of a graph G is a graph on the same vertex set as G and its edges form a superset of $E(G)$.

[2] A star forest is a disjoint union of stars.

Sparse graph classes. A *treedepth decomposition* of a graph G is a rooted forest F on the same vertices as G, such that for every edge $uv \in E(G)$ either u is an ancestor of v in F or vice versa. The *treedepth* of a graph G is the minimum depth any treedepth decomposition of G.

A graph G is a *topological depth-r minor* of another graph H if an $\leq r$-subdivision of G is isomorphic to a subgraph of H [24, 5].

▶ **Definition 1** ([24])**.** A graph class $\mathcal{C}$ has *bounded expansion* if for all $r \in \mathbf{N}$ there exists $t = t(r) \in \mathbf{N}$ such that for all $G \in \mathcal{C}$, and all topological depth-r minors H of G, $\|H\|/|H| \leq t$.

It is *nowhere dense* if for all $r \in \mathbf{N}$ there exists a $t \in \mathbf{N}$ such that no $G \in \mathcal{C}$ contains K_t as a topological depth-r minor. If a graph class is not nowhere dense it is *somewhere dense.*

▶ Remark 2. Following [16], a class has *effectively bounded expansion* if $t(r)$ is computable. If the class $\mathcal{C}$ in Lemmas 3 and 5 and Theorem 18 has effectively bounded expansion, then the algorithms in those statements are uniform and their running times are computable.

Logic. The notation $\bar{x}$ stands for $x_1 \dots x_{|\bar{x}|}$. Usually $|\bar{x}|$ is denoted by k. We write $\varphi(\bar{x})$ if φ has $\bar{x}$ as free variables. Let G be a structure and β be the assignment with $\beta(x_i) = u_i$ for $i \in [k]$. For simplicity, we write $G \models \varphi(\bar{u})$ instead of the satisfaction relation $(G, \beta) \models \varphi(\bar{x})$ and $[\![\varphi(\bar{u})]\!]^G$ instead of the interpretation $[\![\varphi(\bar{x})]\!]^{(G,\beta)}$ which can be a number if φ is a counting term from FOC($\mathbf{P}$). A formula is *quantifier-free* if it contains no $\exists, \forall$ or $\#$ quantifiers. A *conjunctive clause* is a conjunction over (possibly negated) predicates. The *length* of a formula φ is denoted by $|\varphi|$ and is the number of symbols in φ. In particular, the length of any number-symbol $N \in \mathbf{Z}$ is one (and should not be confused with the length of the binary encoding of N). Adjacency between two variables x, y is denoted by $x \sim y$. For two signatures we write $\sigma \subseteq \rho$ to indicate that ρ extends σ. All signatures are finite and the cardinality $|\sigma|$ of a signature equals its number of symbols.

We interpret a labeled graph $G = (V, E, P_1, \dots, P_m)$ as a logical structure with universe $V(G) \coloneqq V$, binary relation $E(G) \coloneqq E$ and unary relations $P_1, \dots, P_m$.

Counting Logic. We are interested in fragments of the counting logic FO($\mathbf{P}$), introduced by Kuske and Schweikardt [21], which is a fragment of FOC($\mathbf{P}$). Compared to traditional first-order logic, it introduces counting over one variable and comparing the number of witnesses using numerical predicates from a set $\mathbf{P}$. We give an informal definition. Given a FO($\mathbf{P}$) formula φ and a variable y, $\#y\,\varphi(y\bar{x})$ is a *counting term.* The semantics of $[\![\#y\,\varphi(y\bar{u})]\!]^G$ are the number of vertices v satisfying $[\![\varphi(v\bar{u})]\!]^G$. Multiple counting terms $t_1, \dots, t_m$ with a predicate $P \in \mathbf{P}$ yield an FO($\mathbf{P}$) formula $P(t_1, \dots, t_m)$. The semantics of this formula is true iff the evaluation of the counting terms is in $[\![P]\!]$. This is generalized by FOC($\mathbf{P}$), which allows counting tuples $\#\bar{y}\,\varphi(\bar{y}\bar{x})$ and allows addition and multiplication of counting terms. For more details, we refer the reader to [21]. The logic FO($\{>0\}$) [10] is a special case of FO($\mathbf{P}$) where the unary predicates are of the form $\geq t$ for $t \in \mathbf{N}$. That is, counting terms can be used only in the form $\#y\,\varphi(y\bar{x}) \geq t$. This logic is also known as FO(C) [12].

Computational model. We use the RAM model with a uniform cost measure.

If a function $f\colon Z^\ell \to \mathbf{Z}$ is part of the output of an algorithm, it is represented by a list of its non-zero entries. Thus, even if the domain has infinite size, the representation is finite if f's support is. In most of our cases, the support of f will be $\{-N, \dots, N\}^\ell$ for some N, which usually is linear in the input.

3 Applications

Before we get into the main part of this work, we want to illustrate in detail its possible applications with a few examples. Let us consider a variation of the partial dominating set problem, the Exactly t-Dominating Set Problem [19]. We are interested in set D of size k that dominates *exactly* t vertices in a graph G. By deciding $G \models \psi$ using Theorem 6 for the formula

$$\psi \equiv \exists x_1 \dots \exists x_k \# y \,(\bigvee_{i=1}^{k} y = x_i \vee y \sim x_i) = t$$

we can solve the Exactly t-Dominating Set Problem on classes of bounded expansion in time $f(k)n^2 \operatorname{polylog} n$. As a byproduct we get a list of all t such that there exists such a set of size k. Hence, one could also solve the problem of finding a partial dominating set of size k where the number of dominated vertices t has to satisfy some (easily computable) numerical predicate P, e.g., that t has to be prime.

Another variation: Given a graph where the vertices are colored red or blue, we want to find a set D of k vertices that dominates at least t_1 many red and at least t_2 many blue vertices. This can be expressed by the following formula:

$$\exists x_1 \dots \exists x_k (\# y \,\big(\mathit{blue}(y) \wedge \bigvee_{i=1}^{k} (y = x_i \vee y \sim x_i)\big) \geq t_1 \wedge \# y \,\big(\mathit{red}(y) \wedge \bigvee_{i=1}^{k} (y = x_i \vee y \sim x_i)\big) \geq t_2)$$

Thus, we can evaluate this problem in $f(k)n^2 \operatorname{polylog} n$ time on classes of bounded expansion by Lemma 11. If "$\geq$" is replaced by "$=$" in the formula above, by Theorem 6 we can even compute all pairs (t_1, t_2) where the adjusted formula holds, but in time $f(k)n^3 \operatorname{polylog} n$. Additionally, it is also possible to express that the vertices $x_1, \dots, x_k$ are, e.g., connected or independent. This increases the run time only in the function $f(k)$.

4 Algorithms

In this section we show the algorithmic results, most importantly Theorem 6 in Sections 4.1 and 4.2. Namely, that both the query evaluation and query counting problem for formulas of the form $\psi \equiv \exists x_1 \dots \exists x_k \, P(\# y \, \varphi_1(\bar{x}, y), \dots, \# y \, \varphi_\ell(\bar{x}, y))$ can be solved in $f(|\psi|)n^{\ell+1} \operatorname{polylog} n$ time on graph classes of bounded expansion. This is followed by an improvement in the running time by a factor n for the special case of Lemma 11 that deals with $\leq$-constraints in Section 4.3. In a nutshell, we put the information needed to evaluate general counting terms depending on multiple variables into vertex weights that depend on only one variable. The resulting combinatorial problem is then reduced to the case of bounded treedepth. Solving this problem on bounded treedepth is then deferred to its own Section 4.2 as it is the main technical contribution of this paper. There, we introduce Mosbah's and Courcelle's concept of MSO-evaluation problems. Then we show that our problem can indeed be modeled as an MSO-evaluation problem and that we get the desired running time. In Section 4.3 we consider the special case of $\leq$-constraints, which can be handled more efficiently. At last, in Section 4.4 we show that the previous results also hold if the counting quantifier can count tuples instead of single variables.

4.1 Reduction to Weighted Sets

We build upon the tools used in [10][3] as there a similar but weaker fragment of FO($\{>0\}$) was considered, namely formulas of the form $\exists x_1 \ldots \exists x_k \, \#y \, \varphi(y, x_1, \ldots, x_k) > N$, (see (1)). However, we need to adapt them to the multivariate case, i.e., where we consider multiple counting terms $\#y \, \varphi(y\bar{x})$ at once.

The result of the following lemma is similar to [9, Theorem 4]. The only difference is that equality between $[\![\#y \, \varphi(y\bar{u})]\!]^{\vec{G}}$ and $\sum_{i=1}^{|\bar{x}|} c_{\omega,i}(u_i)$ is generalized to the ℓ formulas $\varphi_1, \ldots, \varphi_\ell$ and ℓ sums over "families" of weight functions $c^{(j)}_{\omega,i}$. Note that the set Ω is the same for every φ_j and that the running time does not change compared to [9, Theorem 4].

The following lemma allows us to represent counting terms that depend on $k+1$ vertices as a sum of vertex weights which depend only on one variable. This procedure works even for multiple counting terms at once.

▶ **Lemma 3.** *Let $\mathcal{C}$ be a class of bounded expansion with signature σ. One can compute for first-order formulas $\varphi_1(y\bar{x}), \ldots, \varphi_\ell(y\bar{x})$ with signature σ a set of conjunctive clauses Ω with free variables $\bar{x}$, and signature $\rho \supseteq \sigma$ that satisfies the following property:*

There exists a class $\mathcal{C}'$ of bounded expansion with signature ρ. such that for every $\vec{G} \in \mathcal{C}$ one can compute in time $O(\|\vec{G}\|)$ an expansion $\vec{G}' \in \mathcal{C}'$ of $\vec{G}$ and functions $c^{(j)}_{\omega,i}(v) \colon V(\vec{G}) \to \mathbf{Z}$ for $\omega \in \Omega$, $i \in [|\bar{x}|]$, and $j \in [\ell]$ with $c^{(j)}_{\omega,i}(v) = O(|\vec{G}|)$ such that for every $\bar{u} \in V(\vec{G})^{|\bar{x}|}$ there exists exactly one formula $\omega \in \Omega$ with $\vec{G}' \models \omega(\bar{u})$. For this formula

$$\left([\![\#y \, \varphi_1(y\bar{u})]\!]^{\vec{G}}, \ldots\ldots, [\![\#y \, \varphi_\ell(y\bar{u})]\!]^{\vec{G}}\right) = \left(\sum_{i=1}^{|\bar{x}|} c^{(1)}_{\omega,i}(u_i), \ldots\ldots, \sum_{i=1}^{|\bar{x}|} c^{(\ell)}_{\omega,i}(u_i)\right).$$

Proof. When we look into the proof of [9, Theorem 4], we see that the expansion $\vec{G}'$ of $\vec{G}$ does not depend on any formula, but only on G. Moreover, the set of conjunctive clauses Ω depends only on the signature ρ. Thus, when applying [9, Theorem 4] to the formulas $\varphi_1, \ldots, \varphi_\ell$ sequentially both the graph extension $\vec{G}'$ and the set of clauses Ω are identical for each application. Only the weight functions depend on $\varphi_1, \ldots, \varphi_\ell$. Thus, we get our result. ◀

For simplicity, let us define $C^{(j)}_\omega(\bar{u}) \coloneqq \sum_{i=1}^{|\bar{x}|} c^{(j)}_{\omega,i}(u_i)$ and abbreviate it as $C^{(j)}(\bar{u})$ if ω is clear from context.

With Lemma 3, the original task boils down to finding a vertex tuple $\bar{u}$ that satisfies a simple quantifier-free formula ω and has "correct" vertex weights, i.e., $C^{(j)}(\bar{u}) = w_j$ for all $j \in \ell$. Let us give a formal definition.

▶ **Definition 4.**

Counting ℓ-Weighted k-Set Problem
Input: $k, \ell \in \mathbf{N}$, a graph G, a quantifier-free FO-formula ω with variables $x_1 \ldots x_k$ and weight functions $c^{(j)}_i \colon V(G) \to \mathbf{Z}$ for $i \in [k], j \in [\ell]$
Output: a partial function $a \colon \mathbf{Z}^\ell \to \mathbf{N}$ where $a(w)$ is the number of $\bar{u} \in V(G)^k$ satisfying

1. $G \models \omega(\bar{u})$
2. $C^{(j)}(\bar{u}) = \sum_{i \in [k]} c^{(j)}_i(u_i) = w_j$ for all $j \in [\ell]$

[3] The relevant results for our approach can be found in more detail in the full version on arXiv [9].

To solve the Counting ℓ-Weighted k-Set Problem, we follow the approach in [9, Theorem 2] and reduce the case for graphs of bounded expansion to graphs of bounded treedepth. These graphs are structurally much simpler and have a rich landscape of meta-theorems. The reduction is achieved by a so-called low treedepth coloring [24].

For now, let us assume we can solve the problem above in time $f(|\omega|, d)n^{\ell+1} \operatorname{polylog} n$ on graphs of tree-depth d, as stated in the following lemma. We give the proof in Section 4.2.

▶ **Lemma 5.** *Assume we are given $k, \ell \in \mathbf{N}$, a quantifier-free first-order formula $\omega(x_1 \dots x_k)$, a treedepth-decomposition of a graph G of depth d and weight functions $c_i^{(j)} \colon V(G) \to \mathbf{Z}$ with $|C^{(j)}(v_1, \dots, v_k)| \leq N$ for $j \in [\ell]$ and $i \in [k]$. Then, we can solve the Counting ℓ-Weighted k-Set Problem with G, ω and $c_i^{(j)}$ as input in time $f(|\omega|, d)nN^\ell \log N$ for some function f.*

With this result at hand, we use a similar technique as in the proof of [9, Theorem 2].

▶ **Theorem 6.** *Let $\mathcal{C}$ be a class of bounded expansion. There exists a function f such that for all graphs $G \in \mathcal{C}$ and first-order formulas $\varphi_1(y\bar{x}), \dots, \varphi_\ell(y\bar{x})$, one can compute a function $a \colon \mathbf{Z}^\ell \to \mathbf{N}$ where $a(w_1, \dots, w_\ell)$ is the number of solutions $\bar{u} \in V(G)^k$ with*

$$G \models \psi(\bar{u}) \qquad \text{where } \psi(\bar{x}) \equiv \bigwedge_{j \in [\ell]} \#y\, \varphi_j(y\bar{x}) = w_j$$

in time $f(|\psi|)n^{\ell+1} \operatorname{polylog}(n)$.

Proof. We apply Lemma 3 to G and $\varphi_1, \dots, \varphi_\ell$ yielding a functional graph $\vec{G}'$ from a class of bounded expansion, a set of conjunctive clauses Ω over an extended signature, and weight functions $c_{\omega,i}^{(j)} \colon V(G) \to \mathbf{Z}$ for $\omega \in \Omega, i \in [k], j \in [\ell]$. Then

$$G \models \bigwedge_{j \in [l]} \#y\, \varphi_j(y\bar{u}) = w_j$$

iff there exists an $\omega \in \Omega$ such that both

$$\vec{G}' \models \omega(\bar{u}) \text{ and } C_\omega^{(j)}(\bar{u}) = w_j \text{ for all } j \in [\ell].$$

The latter is an instance of the Counting ℓ-Weighted k-Set Problem. Using the techniques from the proof of [9, Theorem 2] we reduce this problem from graphs of bounded expansion to graphs of bounded treedepth with the help of low treedepth colorings.

Before we continue, it will be easier to transform $\vec{G}'$ into a relational structure G': Every function f is replaced by a binary relation E_f with $E_f(G') = \{(v, f^{\vec{G}'}(v)) \mid v \in V(\vec{G}')\}$, and we keep unary predicates. For every (functional) conjunctive clause $\omega(\bar{x})$ we construct a relational conjunctive clause $\omega'(\bar{x}\bar{z})$ with $\vec{G}' \models \omega(\bar{u})$ iff $G' \models \exists \bar{z} \omega'(\bar{x}\bar{z})$ for every $\bar{u} \in V(\vec{G}')^{|\bar{x}|}$.

There exists a vertex coloring χ of G, so called *k-treedepth colorings*, such that each subgraph of G induced by at most k colors has at most treedepth k. Denote the set of all such subgraphs with $\mathcal{H}$. As G is from a class of bounded expansion $\mathcal{C}$, one can compute in $f_{\mathcal{C}}(k)n$ time a k-treedepth coloring that uses at most $f_{\mathcal{C}}(k)$ colors [24]. Hence, if we want to "find" a fixed k-vertex tuple $\bar{u} \in V(G)^k$, there must exist k colors such that $\bar{u}$ is contained in the subgraph induced by those k colors. So, we need to consider only $\binom{f(k)}{k}$ such subgraphs when looking for a solution to the ℓ-Weighted k-Set Problem and solve the problem on such a subgraph which has treedepth k. For more details, see the proof of [9, Theorem 2].

Thus, for every $\omega \in \Omega$, possible colorings of $\bar{u}$ with colors from χ and $H \in \mathcal{H}$ we apply Lemma 5 to solve the counting problem. Then, for every weight tuple $(w_1, \dots, w_\ell) \in \{-N, \dots, +N\}^\ell$ we add up the counts over all subgraphs $H \in \mathcal{H}$, possible colorings of $\bar{u}$, and $\omega \in \Omega$, and output this sum for $(w_1, \dots, w_\ell)$. In total, the computation time is $O\big(|\Omega|\binom{f(k)}{k}knN^\ell \operatorname{poly}(\ell) \operatorname{polylog}(N) + f'(|\psi|)n\big) = O\big(g(|\psi|)n^{\ell+1} \operatorname{polylog}(n)\big)$. ◀

By using Theorem 6 and looking for a tuple $(w_1, \dots, w_\ell)$ which is contained in P and has strictly positive output we get the following corollary.

▶ **Corollary 7.** *Let $\mathcal{C}$ be a class of bounded expansion. There exists a function f such that for all graphs $G \in \mathcal{C}$, first-order formulas $\varphi_1(y\bar{x}), \dots, \varphi_\ell(y\bar{x})$, and all computable relations $P \subseteq \mathbf{N}^\ell$ one can decide*

$$G \models \exists x_1 \dots \exists x_k P(\#y\, \varphi_1(y\bar{x}), \dots, \#y\, \varphi_\ell(y\bar{x}))$$

in time $O\big(f(|\varphi_1| + \dots + |\varphi_\ell|)n^{\ell+1}\operatorname{polylog}(n) + rn^\ell\big)$ where r is the time needed to decide membership in P.

4.2 Solving the ℓ-Weighted k-Set Problem on bounded treedepth

In order to construct an algorithm satisfying Lemma 5 (that is, solving the ℓ-Weighted k-Set Problem on bounded treedepth graphs), we will use the machinery of *monadic second-order evaluations* on graphs of bounded treewidth (on bounded treedepth even), introduced by Courcelle and Mosbah [2]. (This should not be confused with Courcelle's theorem for MSO model checking on graphs of bounded treewidth [3]). Monadic second order evaluations can be used to compute a function of an optimal solution if the function can be computed iteratively on a tree decomposition. In the case of a join node the values of the children have to be combined in some way and in the case of introduce and forget nodes the values have to be updated in the right way. When you use this machinery, and typically in other DP algorithms as well, the values you use carry more information than is needed in the end. Therefore, the needed information has to be extracted by an *homomorphism.* Updating the values is done by operations on a *semiring.* For example, if you want to find a vertex cover of minimal size in a node-weighted graph, your semiring would be $(\mathbf{N} \cup \{\infty\}, \min, +, \infty, 0)$ and the homomorphism would map a (partial) solution to its weight, which is the sum of the weights of each contained vertex.

We have to show that Counting ℓ-Weighted k-Set Problem is an MSO-evaluation problem. We refer the reader to [22, Section 3.3.3] for definitions of semirings, weak homomorphism, MSO logic, assignments and the semiring of assignments. We also follow their notation.

Let us very briefly mention the most relevant definitions. Let G be a graph and φ a first-order formula.[4] Then, $\operatorname{sat}(\varphi, G)$ denotes the set of assignments to variables of φ over G, and $\operatorname{assignring}(\varphi, G) = (2^{\operatorname{assignments}(\varphi,G)}, \cup, \hat{\cup}, \emptyset, \hat{\emptyset})$ denotes the *semiring over the power set of assignments* where $\hat{\cup}$ is the Cartesian product of assignments and $\hat{\emptyset}$ the set containing only the empty assignment. The operation $\hat{\cup}$ combines assignments from G_1 and G_2 when a graph G can be "decomposed" into two graphs G_1 and G_2, e.g., as in tree-decompositions.

▶ **Definition 8.** A graph problem P is an *MSO-evaluation problem* if there is an MSO-formula φ and a semiring $R = (U_R, \oplus, \otimes, \hat{0}, \hat{1})$ such that P can be stated as computing $h(sat(\varphi, G))$, where $G = (V, E)$ and the weak semiring homomorphism h between $\operatorname{assignring}(\varphi, G)$ and R are part of the input.

The following fact follows from Proposition 3.1 and Theorem 2.10 from [2].

▶ **Proposition 9.** *Let P be an MSO-evaluation problem, expressed by a weak homomorphism h into a semiring R. Then, P can be computed on a graph G given a tree decomposition $\mathcal{T}$ of width w in time $O(f_P(w)|\mathcal{T}|\eta)$ where η is the time complexity of evaluating the homomorphism and performing the semiring operations.*

[4] The formula can also be from MSO in general. For our needs, FO suffices.

To express the Counting ℓ-Weighted k-Set Problem as an MSO-evaluation problem, we define the semiring ℓ-WeightedSolCount $:= (U, \oplus, \otimes, \hat{0}, \hat{1})$ where elements of $U = \mathbf{N}^W$ are infinite sequence of natural numbers indexed by ℓ-tuples in $W = \mathbf{Z}^\ell$. An entry $a \in U$ means that, for a tuple $w \in W$, there are a_w many assignments $\bar{u} \in V(G)^k$ with multivariate weight w.

More formally, given a graph G with weight functions $c_i^{(j)}$ and a first-order formula φ, we define a weak homomorphism $h\colon \operatorname{assignring}(\varphi, G) \to \ell\text{-WeightedSolCount}$ by

$$h(A) := a \text{ and for } w \in W \text{ with } a_w := \big|\{\, \bar{u} \in A \mid C^{(j)}(\bar{u}) = w_j \text{ for all } j \in [\ell] \,\}\big|$$

where A is some set of assignments[5] $\bar{u} \in V(G)^k$ to φ.

Hence, this problem is an MSO-evaluation problem by Definition 8. Equivalently, instead of a being a vector indexed by ℓ-tuples, we can imagine a being a function mapping an ℓ-tuple w to the number $a(w)$ of solutions with multivariate weight w.

But first, we need to complete the definition of ℓ-WeightedSolCount by defining the operations and constants. The addition $\oplus$ is the element-wise addition defined as

$$(a \oplus b)_w := a_w + b_w \text{ for } w \in W$$

with the neutral element $\hat{0} := (0)^W$, the all-zero vector indexed by W. The multiplication $\otimes$ is defined by the convolution

$$(a \otimes b)_w := \sum_{r+s=w} a_r \cdot b_s \text{ for } w \in W$$

with neutral element $\hat{1}$, where $\hat{1}_{(0,\ldots,0)} = 1$ and $\hat{1}_w = 0$ for $w \neq (0, \ldots, 0)$.

The weak homomorphism h is then defined by

$$h(\{\bar{u}\}) = (a_w)_{w \in W} \text{ with } a_w = \begin{cases} 1 & \text{if } C^{(j)}(\bar{u}) = w_j \text{ for all } j \in [\ell] \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{u}$ is an assignment from $V(G)^k$. The image of h for other sets of assignments is derived from the singleton sets and semiring properties.

One can easily verify that ℓ-WeightedSolCount is a semiring. Indeed, ℓ-WeightedSolCount is similar to the CardCounting semiring in [22, Example 28.4]. However, we are not interested in the number of solutions of some cardinality but of some certain weight. Moreover, the weight is multivariate and not univariate. Also, it should not be confused with the evaluation problem described in [2, Section 4.8]. There, a linear combination of multiple weight functions is considered, which is fundamentally different from our approach.

▶ **Example 10.** To show the Counting ℓ-Weighted k-Set Problem and ℓ-WeightedSolCount in action, let us consider the problem of finding an independent set of certain weight on a graph with two vertex weight functions, $c(v)$ and $c'(v)$. Let us say we are interested in an independent set with weights 24 and 96 w.r.t to c and c' respectively,

This problem can be easily modeled as Counting ℓ-Weighted k-Set Problem with $\ell = 2$ and $\omega(\bar{x}) = \bigwedge_{i<j} x_i \nsim x_j$. The output function $a(24, 96)$ tells us the number of (ordered) independent sets $\bar{u}$ with weight $\sum_i c(u_i) = 24$ and $\sum_i c'(u_i) = 96$.

[5] Here, we only consider assignments of k vertex variables as φ is constrained to have only those as free variables. For MSO formulas, we would need to consider set variables, but this is out of scope for our work.

The weak homomorphism h maps a set $A \subseteq V(G)^k$ of independent sets to a vector $a \in \mathbf{Z}^{\mathbf{N}^\ell}$. The entry $h(A)_{(24,96)}$ tells us the number of independent sets in A that have weight 24 and 96 w.r.t. c and c'.

Before we apply the result about MSO-evaluation problems, discuss the complexity of the operations on ℓ-WeightedSolCount. Even though elements of U have a priori infinite size, in our application their size will be bounded. The weights in the graph are finite, even bounded by $O(n)$ by Lemma 3. Thus, the highest weight possible of a k-vertex tuple is $O(kn)$ which we will denote by N. So let us assume we are given two elements a, b from ℓ-WeightedSolCount, which are *N-bounded*. That is, $a_w = 0$ if $\|w\|_\infty > N$. Such vectors can be represented naturally in $O(N^\ell)$ space. Then the addition $a \oplus b$ trivially needs time $O(N^\ell)$. As $a \otimes b$ is a convolution, it can be computed in time $O(\ell N^\ell \log N)$ using DFT [28].

The following lemma follows from Counting ℓ-Weighted k-Set Problem being an MSO-evaluation problem and applying the result of Courcelle and Mosbah.

▶ **Lemma 5.** *Assume we are given $k, \ell \in \mathbf{N}$, a quantifier-free first-order formula $\omega(x_1 \dots x_k)$, a treedepth-decomposition of a graph G of depth d and weight functions $c_i^{(j)} : V(G) \to \mathbf{Z}$ with $|C^{(j)}(v_1, \dots, v_k)| \le N$ for $j \in [\ell]$ and $i \in [k]$. Then, we can solve the Counting ℓ-Weighted k-Set Problem with G, ω and $c_i^{(j)}$ as input in time $f(|\omega|, d)nN^\ell \log N$ for some function f.*

Proof. First, from a treedepth decomposition $\mathcal{T}$ of depth d we can easily construct a tree decomposition of width d. Also, we know the ℓ-Weighted k-Set Problem is an MSO-evaluation problem, which can be expressed by evaluating $h(sat(\varphi, G))$ as described above in the definition of ℓ-WeightedSolCount. Hence, we can apply Proposition 9 on G, φ and the weight functions $c_i^{(j)}$ for this problem. This yields us an algorithm solving the given problem in time $O(f(d)|\mathcal{T}| \cdot \eta) = O(f(d)n\ell N^\ell \log N)$ where η is the complexity of the operations in the semiring ℓ-WeightedSolCount. Indeed, η is bounded by $O(\ell N^\ell \log N)$ as the vectors appearing during the computation are N-bounded and the image of $C^{(j)}$ is bounded by N as well. ◀

4.3 Run Time Improvements for $\le$-Relations

For the special case, where $\mathbf{P}$ consists of boolean combinations of ℓ $\le$-relations, i.e., $\#y\, \varphi(y\bar{x}) \ge t$ for constant $t \in N$, we shave of a factor of n in the running time.

This case can easily be reduced to a bounded number of conjunctions of counting terms $\#y\, \varphi(y\bar{x}) \ge t$ of length at most ℓ, by transforming the boolean combination into disjunctive normal form (DNF). Then each conjunctive clause is regarded separately by pushing the existential quantifier into the disjunction.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a graph class of bounded expansion. There exists a function f such that for all graphs $G \in \mathcal{C}$, first-order formulas $\varphi_1(y\bar{x}), \dots, \varphi_\ell(y\bar{x})$, and numbers $t_1, \dots, t_\ell \in \mathbf{N}$ one can decide in time $O(f(|\varphi_1| + \cdots + |\varphi_\ell|)n^\ell \operatorname{polylog}(n))$ whether*

$$G \models \exists x_1 \dots \exists x_k (\#y\, \varphi_1(y\bar{x}) \ge t_1 \wedge \cdots \wedge \#y\, \varphi_\ell(y\bar{x}) \ge t_\ell).$$

Proof. In the proof of Theorem 6, the subroutine of Lemma 5 computes semiring operations $O(n)$ times which determines the overall run time. The run time of such operations is almost linear in the table size of the dynamic program. There, the number of tuples was assumed to be the worst-case, namely $O(N^\ell)$, resulting in a running time for the semiring operations of $O(\mathrm{poly}(\ell)N^{\ell}\,\mathrm{polylog}(N))$. In our case, we do not need all tuples, only the Pareto-optimal ones. That is, for any (partial) solutions $\bar{u}, \bar{v} \in V(G)^k$ if $\bar{u}$ dominates $\bar{v}$, that is, $[\![\#y\,\varphi_i(y\bar{u})]\!]^G \geq [\![\#y\,\varphi_i(y\bar{v})]\!]^G$ for all $i \in [\ell]$, then $\bar{v}$ can be disregarded further on.

The number of Pareto-optimal solutions in the domain $\{-N, \ldots, +N\}^{\ell}$ is bounded by $O(N^{\ell-1})$. (Fix values for the first $\ell - 1$ dimensions, then there can be at most one Pareto-optimal tuple agreeing with the first $\ell - 1$ values.)

Thus, the application of the semiring operations in Lemma 5 takes only $O(\ell N^{\ell-1}\log N)$ time instead of $O(\ell N^{\ell}\log N)$. Continuing the run time analysis as in Lemma 5 and Theorem 6, we get the desired result. ◀

Note that this only improves the run time for the decision problem for this fragment. This approach does not work for the counting problem.

We can achieve the same run time improvement for *modulo counting*. If the predicate is a boolean combination of modulo counting terms, that is, the predicate checks if $\#y\,\varphi(y\bar{x}) \equiv a \pmod{b}$ then both the decision and even the counting problem is in time $O(f(|\varphi_1| + \cdots + |\varphi_\ell|)n^{\ell}\,\mathrm{polylog}(n))$ for classes of bounded expansion. However, Nešetřil, Ossona de Mendez and Siebertz [25] showed recently an even stronger result; they achieve a linear fpt time algorithm for the model checking problem of the logic FO+Mod which contains arbirarily nested modulo counting quantifiers.

4.4 Lifting to Counting Tuples $\#(y_1, \ldots, y_p)$

The algorithmic results (Theorem 6, Corollary 7, and Lemma 11) can be lifted to counting *tuples*, that is, to counting quantifiers $\#\bar{y}\,\varphi(\bar{y}\bar{x})$ that are also part of FOC($\mathbf{P}$) (where $\bar{y}$ is tuple of variables $(y_1, y_2, \ldots, y_p)$).[6] This comes at a cost: The polynomial factor of the run time increases to $n^{(\ell+1)p}$.

▶ **Corollary 12.** *Let $\mathcal{C}$ be a class of bounded expansion. There exists a function f such that for a given graph $G \in \mathcal{C}$, first-order formulas $\varphi_1(\bar{y}\bar{x}), \ldots, \varphi_\ell(\bar{y}\bar{x})$ and all computable relations $P \subseteq \mathbf{N}^{\ell}$ one can decide*

$$G \models \exists x_1 \ldots \exists x_k P(\#\bar{y}\,\varphi_1(\bar{y}\bar{x}), \ldots, \#\bar{y}\,\varphi_\ell(\bar{y}\bar{x}))$$

in time $O\big(f(|\varphi_1| + \cdots + |\varphi_\ell|)n^{(\ell+1)|\bar{y}|}\,\mathrm{polylog}(n) + rn^{\ell}\big)$ where r is time needed to decide membership in P.

We achieve this result by adapting both the statements made in this paper and their dependencies, which originate from [10]. This adaptation is quite straightforward; it suffices to replace every occurrence of $\#y$ with the more general counting quantifier $\#\bar{y}$. Additionally, the monovariate weight functions $c_i^{(j)}$ must be extended to $c_I^{(j)} \colon V(G)^p \to \mathbf{Z}$, where $I \in V(G)^p$.

In the statements [9, Lemma 2, 3, 6, 15, and Theorem 2, 4], we apply the same changes. Furthermore, if, for some $i \in [|\bar{x}|]$, clauses are restricted to the form $\tau(y) \wedge \psi(\bar{x}) \wedge f(y) = g(x_i) \wedge \Delta^{\neq}(y\bar{x})$ (with non-empty or empty $\Delta^{\neq}$) in the statements or proofs, we extend them to the form $\tau(\bar{y}) \wedge \psi(\bar{x}) \wedge f_1(y_1) = g_1(x_{i_1}) \wedge \cdots \wedge f_p(y_p) = g_p(x_{i_p}) \wedge \Delta^{\neq}(y\bar{x})$ for some $I = (i_1, \ldots, i_p) \in [k]^p$. Note that the runtime increase in adapting [9] mostly occurs in [9, Lemma 6], where its running time increases to $\|G\|^p$.

[6] We want to thank Michał Pilipczuk as he brought to our attention that this approach may likely extend to counting tuples which we formerly thought to be impossible.

The same applies to Definition 4 and Lemmas 3 and 5 in Section 4.1. It's worth mentioning that the N in Theorem 6 is now $O(n^p)$ instead of being linear in n, which explains the increase in time complexity.

As these changes are rather trivial and clutter the presentation of the results, we decided to omit explicit proofs here.

5 Hardness

We have seen that formulas ψ of the form $\exists x_1 \ldots \exists x_k\, P(\#y\, \varphi_1(\bar{x}, y), \ldots, \#y\, \varphi_\ell(\bar{x}, y))$ can be evaluated in time $O(f(|\psi|)n^{\ell+1}\,\text{polylog}(n))$ on graph classes of bounded expansion. We now show that this cannot be improved by more than an almost linear factor in n under SETH.

For this, let us recall the Subset Sum Problem, which is often used as a starting point in fine-grained complexity theorem. Here, we need a recent conditional lower bound for this problem.

▶ **Definition 13** (k-Sum)**.** Given n integers $x_1, \ldots, x_n \in \mathbf{N}$ and a target value T, the task in the SubsetSum problem is to decide whether there is a subset of the numbers above which sums to T. For the k-Sum problem, the task for the same instance is to decide where there are k numbers which add up to T.

▶ **Proposition 14** ([1])**.** *Assuming SETH, for any $\varepsilon > 0$ there exists a $\delta > 0$ such that Subset Sum is not in time $O(T^{1-\varepsilon}2^{\delta n})$, and k-Sum is not in time $O(T^{1-\varepsilon}n^{\delta k})$.*

Note that δ does not depend on k, only on ε.

We will show a conditional lower bound of the following problem on star forests. Note that those graphs have treedepth 2.

▶ **Definition 15** (ℓ-Variate k-Satisfaction)**.** Given a graph G, quantifier-free FO formulas $\varphi_1(x_1, \ldots, x_k, y), \ldots, \varphi_\ell(x_1, \ldots, x_k, y)$ and integers $w_1, \ldots, w_\ell$ the problem ℓ-Variate k-Satisfaction asks whether

$$G \models \exists x_1 \ldots \exists x_k \bigwedge_{i=1}^{\ell} \varphi_i(x_1, \ldots, x_k, y) = w_i.$$

First, we make a simple observation that any number can be uniquely represented in any (natural) base. The following is tailored to our use case.

▶ **Observation 16.** *Observe, that for every $\ell, T \in \mathbf{N}$, every number $0 \le x \le T$ can be uniquely expressed as $x = \sum_{j=1}^{\ell} a_j \cdot \tau^{j-1}$ for some $a_j \in \{0, \ldots, \tau - 1\}$ and where $\tau = \lceil \sqrt[\ell]{T} \rceil$.*

Now, we relax the conditions on the base and allow an "overlap". As an example, consider the number 35 in base-10. If we allowed the digits to range from 0 to 15 (represented by $0, \ldots, 9, A, \ldots, F$), 35 can then be expressed also by 2F ($= 2 \cdot 10^1 + 15 \cdot 10^0$ in "base-10 with A-F"). The representation is then of course not unique, but the number of such representations is small.

▶ **Lemma 17.** *Consider Observation 16 but where the a_j are allowed to range from 0 to $k\tau - 1$ instead. Then the number of such representations of x is bounded by k^ℓ.*

Proof. We prove the claim by induction over ℓ. First, let us notice that $\sum_{j=1}^{\ell} a_j \cdot \tau^{j-1} < k\tau^\ell$ for $a_j \in \{0, \ldots, k\tau - 1\}$.

For the induction, let $x \in \{0, \ldots, k\tau^\ell\}$. Consider a representation of x with $x = \sum_{j=1}^{\ell} a_j \cdot \tau^{j-1}$ and $a_j \in \{0, \ldots, k\tau - 1\}$ for all $j \in [\ell]$.

For $\ell = 1$, $x = a_1$ holds. Hence, there is only one representation.

For $\ell > 1$, let us consider furthermore $y = x - a_\ell \tau^{\ell-1}$. Then, by our first note $y = \sum_{j=1}^{\ell-1} a_j \cdot \tau^{j-1} < k\tau^{\ell-1}$. By the induction hypothesis there are at most $k^{\ell-1}$ representations of y in this form (where j ranges from 1 to $\ell - 1$). There are at most k valid choices for a_ℓ, as $x - a_\ell \tau^{\ell-1}$ has to fall into the range $\{0, \ldots, k\tau^{\ell-1} - 1\}$. Thus, there are $k \cdot k^{\ell-1} = k^\ell$ possibilities to represent x as described. ◀

▶ **Theorem 18.** *Assuming SETH, for any $\varepsilon > 0$ and $\ell \in \mathbf{N}$ the ℓ-Variate k-Satisfaction on star forests is not in time $f(\ell, k) \cdot |G|^{\ell-\varepsilon}$ for all $k \in \mathbf{N}$ and functions f.*

A few words about the implications of the theorem are in order: The result does *not* rule out that there exists a pair of ℓ and k and an $\varepsilon > 0$ such that ℓ-Variate k-Satisfaction can be solved in time $f(k, \ell)|G|^{\ell-\varepsilon}$. It only says that (for any fixed ℓ) there cannot be a (family of) algorithms which achieves this "form" of running time. Indeed, if one looks closer into the proof, one can show that for every fixed ℓ there can be at most finitely many k where a faster running time can be achieved. Note especially that for $k < \ell$, one can achieve a running time of $f(k, \ell)n^k < f(k, \ell)n^{\ell+1}$ by essentially brute-forcing all solution candidates.

Before we come to the proof, let us consider the (hopefully easier) case of ℓ being 1 or 2. We can express a given k-Sum instance as a graph where every number x is expressed as a star with x endpoints. Now a set of k numbers from the k-sum instance adds up to T iff there are k stars with T endpoints in total.

Generalizing this to $\ell = 2$, we change the construction such that a star does not have x endpoints but $\lfloor x/\sqrt{T} \rfloor$ blue endpoints and $x - \lfloor x/\sqrt{T} \rfloor$ red endpoints, which represent the most significant bits (MSB) and least significant bits (LSB) of x respectively. Now, a set of k numbers add up to T iff the k corresponding stars have $\lfloor T/\sqrt{T} \rfloor$ blue and $T - \lfloor T/\sqrt{T} \rfloor$ red neighbors in total, where the numbers are the MSB and the LSB of T respectively. Due to carryovers the above statement does not hold technically. This issue can be fixed with a small overhead by guessing the form of the carryovers. The number of possible carryovers is small.

In the proof, the technique is generalized to all $\ell \in \mathbf{N}$ and the technicalities concerning carryovers are addressed as well.

Proof. Let $k, \ell \in \mathbf{N}$. Consider a k-Sum instance with n numbers $x_1, \ldots, x_n$ and a target value T. We reduce this to a ℓ-Variate k-Satisfaction instance, where $|G| = \Theta(n\ell\sqrt[\ell]{T})$ and where G is a star forest with n components. The reduction represents the numbers of the k-Sum instance as a graph G with colors $[\ell]$ and each number x_i as a star in G with $a_{i,j}$ many endpoints of color $j \in [\ell]$ where $x_i = \sum_{j=1}^{\ell} a_{i,j} \cdot \tau^{j-1}$, $\tau = \lceil \sqrt[\ell]{T} \rceil$ and $a_{i,j} \in \{0, \ldots, \tau - 1\}$ (see Observation 16). Then, there exist k numbers in the k-Sum instance which add up to T if and only if there are k vertices $v_1, \ldots, v_k$ in G with (in total) B_j many neighbors of color j for every $j \in [\ell]$ such that $T = \sum_{j=1}^{\ell} B_j \tau^{j-1}$.

After having constructed the graph, let us look at the formula. Each number B_j ranges from 0 to $k(\tau - 1)$ as each vertex v_i has at most $\tau - 1$ neighbors of a given color. Let $\mathcal{T}$ be sets of ℓ-tuples $(w_1, \ldots, w_\ell) \in \{0, \ldots, k(\tau - 1)\}^\ell$ with $T = \sum_{j=1}^{\ell} w_j \cdot \tau^{j-1}$. By Lemma 17 the size of $\mathcal{T}$ is bounded in a function of k and ℓ. Let $(w_1, \ldots, w_\ell) \in \mathcal{T}$. Then, the following formula expresses that there are k (central) vertices whose neighbors of color $j \in [\ell]$ (expressed by the predicate C_j) add up to w_j

$$\psi_{(w_1,\ldots,w_\ell)} \equiv \exists x_1 \ldots \exists x_k \bigwedge_{j=1}^{\ell} \Big(\#y \Big(C_j(y) \wedge \bigvee_{i'=1}^{k} y \sim x_{i'} \Big) = w_j \Big).$$

Thus, G together with the formula above and $(w_1, \ldots, w_\ell)$ is the instance of ℓ-Variate k-Satisfaction constructed above.

Hence, for each tuple $(w_1, \dots, w_\ell) \in \mathcal{T}$ we check whether G satisfies $\psi_{(w_1,\dots,w_\ell)}$. If this is the case for some tuple, the k-Sum instance is accepted, otherwise it is not.

The size of the resulting graph G is $n\ell\lceil\sqrt[\ell]{T}\rceil$, the size of each $\psi_{(w_1,\dots,w_\ell)}$ is $O(k\ell)$, the construction of a single instance takes linear time in $|G|$ and $|\psi|$. This construction is repeated $|\mathcal{T}| = f(k,\ell)$ times for each $\psi_{(w_1,\dots,w_\ell)}$.

To show the theorem statement, assume for contradiction that there exist $\ell \in \mathbf{N}$ and $\varepsilon > 0$ such that for every $K \in \mathbf{N}$ there exists $k \geq K$ such that the ℓ-Variate k-Satisfaction on star forests can be solved in time $O(|G|^{\ell(1-\varepsilon)} f(l,k))$ some function f. We consider any k-Sum instance for k. It has n numbers and a target value T. As we have seen above, it can be reduced to $|\mathcal{T}|$ instances of ℓ-Variate k-Satisfaction on star forests of size $n\ell\sqrt[\ell]{T}$.

Then, k-Sum could be solved in time

$$\Theta(|G| + |\varphi|) + |\mathcal{T}|O(|G|^{\ell(1-\varepsilon)} f(k,\ell)) = O((n\ell\sqrt[\ell]{T})^{\ell(1-\varepsilon)} f(k,\ell)) = O(T^{1-\varepsilon} n^{\ell(1-\varepsilon)} f(k,\ell)).$$

Recall Proposition 14 and let us assume that SETH holds. The above run time contradicts the non-existence of an algorithm for k-Sum with run time $T^{1-\varepsilon} n^{\delta k}$ if $\ell(1-\varepsilon) < \delta k$. As δ depends only on ε, the inequality can be satisfied by "big enough" k. To be more precise, it holds for $k \geq \ell(1-\varepsilon)/\delta$. Thus, we get that the existence of an algorithm for ℓ-Variate k-Satisfaction in time $O(|G|^{\ell(1-\varepsilon)} f(l,k))$ contradicts SETH. ◀

From this result, we get directly a lower bound for model checking of formulas with inequality, that is, $\exists x_1 \dots \exists x_k \bigwedge_{i=1}^{\ell} \#y\, \varphi_i(y\bar{x}) \geq N_i$. Note that by Lemma 11 we have an algorithm with a running time of $f(k,\ell)n^\ell \operatorname{polylog} n$.

▶ **Corollary 19.** *Assuming SETH, for any $\varepsilon > 0$ and $\ell \in \mathbf{N}$ the ℓ-Variate k-Satisfaction with inequalities on star forests is not in time $O(f(\ell,k)|G|^{1/2\ell-\varepsilon})$ for all k and functions f.*

Proof. Note that an equality $\#y\, \varphi(y\bar{x}) = N$ can be expressed by two inequalities $\#y\, \varphi(y\bar{x}) \leq N \wedge \#y\, \varphi(y\bar{x}) \geq N$, and equivalently by $\#y\, \neg\varphi(y\bar{x}) \geq |G| - N \wedge \#y\, \varphi(y\bar{x}) \geq N$. A formula $\exists x_1 \dots \exists x_k \bigwedge_{i\in[\ell]} \#y\, \varphi_i(y\bar{x}) = N_i$ from ℓ-Variate k-Satisfaction with ℓ counting terms is then equivalent to the formula $\exists x_1 \dots \exists x_k \bigwedge_{i\in[\ell]} \#y\, \varphi_i(y\bar{x}) \leq N_i \wedge \#y\, \neg\varphi_i(y\bar{x}) \leq |G| - N_i$ with 2ℓ counting terms using inequalities. The result is weaker in the degree of the polynomial by a factor of $1/2$, as the number ℓ of counting terms doubles when going from equalities to inequalities as discussed above. ◀

Lower Bounds for Other Fragments

In the following we show that our algorithmic results cannot be generalized to slightly more general fragments. Firstly, consider the existential fragment with nested counting, that is, we add another counter formula *inside* the counting formulas considered in Section 4. Secondly, we show that one cannot add even one universal quantifier between the existential quantifiers and the counting quantifier.

Here, we regard the equality predicate $= t$ only (where t is a constant which depends on the graph). This can easily be extended to inequality predicates by replacing $\#y\, \varphi = t$ with both $\#y\, \varphi \geq t \wedge \#y\, \varphi \leq t$. Also note that the constant t depends on the graph. It is also possible to construct a formula which is independent of the graph. However, this formula needs a comparison between two counting terms instead.

The reduction is based on the reduction from [17, Theorem 4.1] and [9, Lemma 16]. The proof is omitted and can be found in the appendix.

▶ **Theorem 20.** *The model checking problem on forests of depth 2 is* W[1]*-hard for FO*$(\{=\})$*-formulas* ψ *of the form*

1. $\exists x_1 \dots \exists x_k \forall z \# y\, \varphi(\bar{x}yz) = t$
2. $\exists x_1 \dots \exists x_k \# z\, [\# y\, \varphi(\bar{x}yz) = t] = s$
3. $\exists x_1 \dots \exists x_k \forall z \forall z' \# y\, \varphi(\bar{x}yzz') = t$

where φ *is a quantifier-free FO-formula.*

Moreover, under ETH there is no algorithm with a running time of $n^{o(\sqrt{|\psi|})}$ *for formulas of the form 1 and 2, and* $n^{o(|\psi|)}$ *for formulas of form 3.*

6 Concluding Remarks

We have shown that on classes of bounded expansion, we can solve the query evaluation and query counting problem of formulas of the form $\exists x_1 \dots \exists x_k\, P(\# y\, \varphi_1(\bar{x}, y), \dots, \# y\, \varphi_\ell(\bar{x}, y))$ in time $f(|\psi|)n^{\ell+1}\operatorname{polylog} n$, and it cannot be improved to $f(|\psi|)n^{\ell-\varepsilon}$ for any $\varepsilon > 0$. For the case of $\leq$-constraints we improved the running time for the query evaluation problem to $f(|\psi|)n^{\ell}\operatorname{polylog} n$.

It would be interesting to close the gap between the lower bound of Theorem 18 and the upper bound of Theorem 6. One approach could be improving the algorithm of Lemma 5. For such approaches it is certainly needed to maintain a table of size $\widetilde{O}(n^{\ell})$ in the worst case. However, it could be possible that in many nodes of the treedepth decomposition, the maintained tables have considerably smaller size, e.g., at the leaves and that together with output sensitive FFT algorithms, e.g., ones that have almost linear run time in the size of the output [23], one could achieve a better run time amortized across all n nodes of the treedepth decomposition. But we do not know if the table size during the dynamic program on the treedepth decomposition permits an improved run time analysis.

Also, for the case that **P** consists of a boolean combination of constraints lower bounding the counting terms by a constant, the gap between lower and upper bound is quite large. As the Pareto-front for those should have size $\Theta(n^{\ell-1})$ it seems that the lower bound could be improved. However, the numerical problems known to us which have strong conditional lower bounds as subset sum are not based on such inequalities.

Moreover, extending our results to the class of nowhere dense graphs would be certainly interesting. The results of [10] for FO$(\{>0\})$ are already partially lifted to such classes [8]. They used very different techniques, however. Especially, they did not use (or show) an equivalent result to Lemma 3. On top of that , recently the non-existence of quantifier elimination for FO on nowhere dense classes was shown [15] which implies that different techniques are *needed.*

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. `doi:10.1145/3450524`.

2 B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1):49–82, 1993. `doi:10.1016/0304-3975(93)90064-Z`.

3 Bruno Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. `doi:10.1016/0890-5401(90)90043-H`.

4 Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 270–279. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.31.

7 Jan Dreier, Nikolas Mählmann, and Sebastian Siebertz. First-order model checking on structurally sparse graph classes. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 567–580. ACM, 2023. doi:10.1145/3564246.3585186.

8 Jan Dreier, Daniel Mock, and Peter Rossmanith. Evaluating restricted first-order counting properties on nowhere dense classes and beyond. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.43.

9 Jan Dreier and Peter Rossmanith. Approximate evaluation of first-order counting queries. *CoRR*, abs/2010.14814, 2020. doi:10.48550/arXiv.2010.14814.

10 Jan Dreier and Peter Rossmanith. Approximate evaluation of first-order counting queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1720–1739. SIAM, 2021. doi:10.1137/1.9781611976465.104.

11 Zdeněk Dvořák, Daniel Král', and Robin Thomas. Deciding first-order properties for sparse graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 133–142. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.20.

12 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

13 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

14 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO_1 model-checking. *J. Comput. Syst. Sci.*, 80(1):180–194, 2014. doi:10.1016/j.jcss.2013.07.005.

15 Mario Grobler, Yiting Jiang, Patrice Ossona de Mendez, Sebastian Siebertz, and Alexandre Vigny. Discrepancy and sparsity, 2021. doi:10.48550/arXiv.2105.03693.

16 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

17 Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 253–266. ACM, 2018. doi:10.1145/3196959.3196970.

18 Wojciech Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:25)2020.

19 Joachim Kneis, Daniel Mölle, and Peter Rossmanith. Partial vs. complete domination: t-dominating set. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, volume 4362 of *Lecture Notes in Computer Science*, pages 367–376. Springer, 2007. doi:10.1007/978-3-540-69507-3_31.

20 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 189–198. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.39.

21 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005133.

22 Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Comput. Sci. Rev.*, 13-14:39–74, 2014. doi:10.1016/j.cosrev.2014.08.001.

23 Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020. doi:10.1109/TIT.2020.2989385.

24 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

25 Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Modulo-counting first-order logic on bounded expansion classes. *Discrete Mathematics*, page 113700, 2023. doi:10.1016/j.disc.2023.113700.

26 Detlef Seese. Linear time computable problems and first-order descriptions. *Math. Struct. Comput. Sci.*, 6(6):505–526, 1996. doi:10.1017/s0960129500070079.

27 Szymon Toruńczyk. Aggregate queries on sparse databases. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 427–443. ACM, 2020. doi:10.1145/3375395.3387660.

28 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.

Path-Reporting Distance Oracles with Linear Size

Ofer Neiman
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Idan Shabat
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Given an undirected weighted graph, an (approximate) distance oracle is a data structure that can (approximately) answer distance queries. A *Path-Reporting Distance Oracle*, or *PRDO*, is a distance oracle that must also return a path between the queried vertices. Given a graph on n vertices and an integer parameter $k \geq 1$, Thorup and Zwick [22] showed a PRDO with stretch $2k-1$, size $O(k \cdot n^{1+1/k})$ and query time $O(k)$ (for the query time of PRDOs, we omit the time needed to report the path itself). Subsequent works [20, 7, 8] improved the size to $O(n^{1+1/k})$ and the query time to $O(1)$. However, these improvements produce distance oracles which are not path-reporting. Several other works [12, 13] focused on small size PRDO for general graphs, but all known results on distance oracles with linear size suffer from polynomial stretch, polynomial query time, or not being path-reporting.

In this paper we devise the first linear size PRDO with poly-logarithmic stretch and low query time $O(\log\log n)$. More generally, for any integer $k \geq 1$, we obtain a PRDO with stretch at most $O(k^{4.82})$, size $O(n^{1+1/k})$, and query time $O(\log k)$. In addition, we can make the size of our PRDO as small as $n + o(n)$, at the cost of increasing the query time to poly-logarithmic. For unweighted graphs, we improve the stretch to $O(k^2)$.

We also consider *pairwise PRDO*, which is a PRDO that is only required to answer queries from a given set of pairs $\mathcal{P}$. An exact PRDO of size $O(n + |\mathcal{P}|^2)$ and constant query time was provided in [13]. In this work we dramatically improve the size, at the cost of slightly increasing the stretch. Specifically, given any $\epsilon > 0$, we devise a pairwise PRDO with stretch $1 + \epsilon$, constant query time, and near optimal size $n^{o(1)} \cdot (n + |\mathcal{P}|)$.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Graph Algorithms, Shortest Paths, Distance Oracles

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.36

Related Version *Full Version*: `https://arxiv.org/abs/2405.14254`

Funding Partially supported by the Lynn and William Frankel Center for Computer Sciences and ISF grant 970/21.

Acknowledgements The second-name author would like to thank Michael Elkin, who also suggested to the second-name author that the pairwise spanner of [19] might be extended to a pairwise PRDO, and that a PRDO for unweighted graphs with size $O\left(n^{1+\frac{1}{k}}\right)$ and stretch $O(k^2)$ can be achieved.

1 Introduction

Given an undirected weighted graph $G = (V, E)$ with n vertices and positive weights on the edges $w : E \to \mathbb{R}_+$, the distance $d_G(u, v)$ between two vertices $u, v \in V$ is the minimal weight of a path between them in G. For a parameter $\alpha \geq 1$, a *distance oracle* with stretch α is a data structure, that given a query for a pair of vertices (u, v), returns an estimated distance $\hat{d}(u, v)$ such that

$$d_G(u,v) \leq \hat{d}(u,v) \leq \alpha \cdot d_G(u,v) \ .$$

A *Path-Reporting Distance Oracle*, or *PRDO*, is a distance oracle that must also return a path in G of weight $\hat{d}(u, v)$ between the queried vertices u, v.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 36; pp. 36:1–36:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Distance oracles have been the subject of extensive research in the last few decades. They are fundamental objects in Graph Algorithms, due to their both practical and theoretical usefulness. The main interest is in the triple tradeoff between the stretch of a distance oracle, its size, and its query time. In some cases, the preprocessing time, (that is, the time needed to construct the distance oracle) is also considered. Note that for every query, a PRDO must return a path P, thus the running time of the query algorithm is always in the general form of $O(q + |P|)$. We usually omit the term[1] $|P|$ from the query time and write only $O(q)$.

This work focuses on path-reporting distance oracles for general graphs. The path-reporting property is more appealing for certain applications that require navigation or routing [17, 11, 24]. See the survey [21] and the references therein for additional applications of distance oracles.

1.1 Linear Size Path-Reporting Distance Oracles

In [22], a PRDO with stretch $2k - 1$, size $O(kn^{1+1/k})$ and query time $O(k)$ was shown. Assuming the girth conjecture of Erdős [15], this result is best possible, up to the factor of k in the size and query time.[2] The query time was improved to $O(\log k)$ by [23]. Observe that $kn^{1/k} \geq \log n$ for any $k \geq 1$, so the PRDO of [22, 23] cannot be sparser than $\Theta(n \log n)$. Subsequent works [20, 7, 8] obtained distance oracles with stretch $2k - 1$, improved size $O(n^{1+1/k})$ and constant query time. However, these distance oracles are not path-reporting.

Of particular interest is trying to achieve a PRDO of linear size. The first such result [12] obtained a PRDO with size $O(tn)$, for any parameter $t \geq 1$, and $O(\log t)$ query time, but had a polynomial stretch $O(tn^{2/\sqrt{t}})$, and required that the aspect ratio of the weights is polynomially bounded. This result was improved by [13], who showed a PRDO with stretch $O(k)$ and size $O(n^{1+1/k})$, but at the cost of increasing the query time to $O(n^{1/k+\epsilon})$, where $\epsilon > 0$ is a *constant*. Note that the query time $O(n^{1/k+\epsilon})$ is prohibitively large - this term dominates the length of many of the output paths, so the PRDO suffers from large query time for these paths. For this reason, it is of special interest to construct linear size PRDOs with query time that is far less than polynomial in n, say polylogarithmic in n. Another variant of [13] does achieve a PRDO with query time $O(\log \log n)$ and stretch $\log^{O(1)} n$, however its size $O(n \log \log n)$ is no longer linear.

We conclude that in all previous results, every linear size distance oracle suffers from a polynomial stretch [12], has polynomial query time [13], or simply cannot report paths [8].

1.1.1 Our Results

In this work we devise the first linear size PRDO for general graphs with polylogarithmic stretch and low query time. Specifically, for any integer $k \geq 1$ our PRDO has stretch $O(k^{4.82})$, size $O(n^{1+1/k})$, and query time $O(\log k)$. Indeed, setting $k = \log n$ yields linear size, $\log^{O(1)} n$ stretch and $O(\log \log n)$ query time. Our main result is for the case $k = \log n$, where we get a linear size PRDO with low query time. In fact, for any $k > \frac{\log n}{\log \log \log n}$, our new PRDO improves all previous bounds.

Note that since the query time is low, in most cases it is dominated by the length of the reported path. Therefore, the strength of this result is not in the precise expression for the query time, but in the fact that the query time is far less than polynomial in n.

[1] Throughout this paper, $|P|$ denotes the number of *edges* in a path P.

[2] In fact, Erdős girth conjecture only implies that to achieve stretch $2k - 1$, any distance oracle must use $\Omega(n^{1+\frac{1}{k}})$ *bits*. However, in [8] a lower bound of $\Omega(n^{1+\frac{1}{k}})$ *words* (each of size $\log n$ bits) for PRDOs is proved.

We can refine our result to obtain an *ultra-compact* PRDO, whose size is as small as $n + o(n)$ (we measure the size by *words*, that is, the oracle uses storage of $n \log n + o(n \log n)$ bits), at the cost of increasing the query time to $\log^{O(1)} n$. In view of the lower bound of [8], this space usage is optimal, up to additive lower order terms. If the graph is unweighted, we offer a simpler construction with improved stretch $O(k^2)$.

1.2 Pairwise Path-Reporting Distance Oracles

A *pairwise distance oracle* is a distance oracle that is also given as input a set of pairs $\mathcal{P}$, and is required to answer queries only for pairs in $\mathcal{P}$. The problem of designing such oracles is related to the extensive research on *distance preservers*: these are subgraphs that preserve exactly all distances between pairs in $\mathcal{P}$. When allowing some stretch, these are sometimes called *pairwise spanners*.[3] Distance preservers were introduced in [9], and pairwise spanners have been studied in [10, 18, 4, 3, 6, 19].

In [13], an exact pairwise PRDO was shown with constant query time and size $O(n+|\mathcal{P}|^2)$. For distance preservers, [9] showed a lower bound of $\Omega(n^{2/3}|\mathcal{P}|^{2/3})$. Note that for $|\mathcal{P}| = n^{2-\delta}$, the lower bound implies that any distance preserver must have size $\Omega(|\mathcal{P}| \cdot n^{\delta/3})$, so there are no distance preservers with near-linear size (except for the trivial case when $\mathcal{P}$ contains a constant fraction of all pairs). In [5], it was proved that exact pairwise PRDOs suffer from the same lower bounds of exact preservers (see Theorem 14 in [5]). Thus, it is very natural to ask if the size of a pairwise PRDO can be reduced when allowing a small stretch. Specifically, we would like to obtain a very small stretch (e.g., $1 + \epsilon$ for any $\epsilon > 0$), and size that is proportional to $|\mathcal{P}| + n$ (which is the basic lower bound).

1.2.1 Our Results

In this work we devise a pairwise PRDO with near optimal size $n^{o(1)} \cdot (|\mathcal{P}| + n)$, constant query time, and stretch $1 + \epsilon$, for any $\epsilon > 0$ (the $o(1)$ term in the size depends logarithmically on ϵ). This result uses the techniques of [19] on hopsets and spanners, and extends them for pairwise path-reporting distance oracles.

1.3 Our Techniques

Our main result on linear size (and ultra-compact) PRDO uses a conceptually simple idea: we partition the graph into $O(n/k)$ clusters, and define the *cluster-graph* by contracting every cluster to a single vertex (keeping the lightest edge among parallel edges). Next, we apply the [22] distance oracle on this cluster-graph. In addition, we store a certain spanning tree for every cluster. Given a query (u, v), the algorithm first finds a path in the cluster-graph between the clusters containing u, v, and for each cluster in this path, it finds an inner-cluster path between the entry vertex and the exit vertex of the cluster, using the spanning tree.

In order to implement this framework, it is required to find a clustering so that the overhead created by going through the spanning tree of every cluster is small enough. For unweighted graphs, we apply a simple clustering with radius $O(k)$, and maintain a BFS tree for each cluster. However, for weighted graphs a more intricate clustering is required. Note that we cannot enforce a small diameter bound on all clusters, since by only restricting the diameter, each cluster can be very small, and we need to have at most $O(n/k)$ clusters. Instead, we use a variant of Borůvka clustering [1].

[3] In [19] pairwise spanners with small $1 + \epsilon$ stretch are called *near-exact preservers*.

In Borůvka's algorithm for minimum spanning tree, in each phase, every vertex adds its adjacent edge of minimal weight to a forest F (breaking ties consistently), and the connected components of F are contracted to yield the vertices of the next phase. If we truncate this process after t phases, we get a clustering, analyzed in [1]. Unfortunately, any phase in this clustering may produce long chains, in which case the stretch cannot be controlled. To rectify this, we delete certain edges in F, so that every cluster is a star, while ensuring that the number of non-singleton clusters is large enough. These stars are also the basis for the spanning tree of each cluster. The main technical part is analyzing the stretch induced by this clustering on the paths returned by calling the distance oracle on the cluster-graph.

1.4 Organization

After some preliminaries in Section 2, we show our PRDO for unweighted graph is in Section 3, and for weighted graphs in Section 4. Our result for pairwise PRDO appears in Section 5.

1.5 Bibliographic Note

Following this work, [14] showed (among other results) a PRDO of linear size with stretch $\tilde{O}(\log n)$ and query time $O(\log\log\log n)$.

2 Preliminaries

Let $G = (V, E)$ be an undirected weighted graph. In all that follows we assume that G is connected.

Spanners. For a parameter $\alpha \geq 1$, an *α-spanner* is a subgraph S of G, such that for every two vertices $u, v \in V$,

$$d_S(u, v) \leq \alpha \cdot d_G(u, v) \ . \tag{1}$$

The spanner is called a *pairwise* spanner, if for a given a set of pairs $\mathcal{P}$, we only require (1) to hold for all $(u, v) \in \mathcal{P}$.

Trees. Let x, y be two vertices in a rooted tree. We denote by $p(x)$ the *parent* of x, which is the unique neighbor of x that lies on the path from x to the root, and by $h(x)$ its *height*, which is the number of edges on the path from x to the root. Denote by $lca(x, y)$ the lowest common ancestor of x, y, which is a vertex z such that x, y are both in its sub-tree, but not both in the sub-tree of any child of z. Note that the unique path between x, y in the tree is the concatenation of the unique paths from x to $lca(x, y)$, and from $lca(x, y)$ to y.

The following lemma let us easily find paths within a tree T.

▶ **Lemma 1.** *Let T be a rooted tree, and assume we are given $h(x)$ and $p(x)$ for every vertex x in T. There is an algorithm that given two distinct vertices a, b in a tree T, finds the unique path between a, b in T. The running time of this algorithm is proportional to the number of edges in the output path (or $O(1)$ if the path is empty).*

Proof. First, if $a = b$, then the desired path is empty and we return it in $O(1)$ time. Otherwise, if $h(a) > h(b)$, recursively find the unique path $P_{p(a),b}$ in T between $p(a)$ and b, and return $\{a, p(a)\} \circ P_{p(a),b}$. Symmetrically, if $h(b) \geq h(a)$, return $\{b, p(b)\} \circ P_{p(b),a}$.

The correctness of this algorithm follows from the fact that if, for example, $h(a) \geq h(b)$, and $a \neq b$, then b cannot be in the sub-tree of a in T, hence the unique path between a, b must pass through $p(a)$. In each recursive call we reduce the sum $h(a) + h(b)$ by 1, and therefore the algorithm ends when $a = b = lca(a, b)$. Therefore the running time of this algorithm is proportional to the length of the unique paths from a to $lca(a, b)$ and from b to $lca(a, b)$. The concatenation of these paths is exactly the returned path by our algorithm, which is the unique path in T between a, b. Hence, the running time is proportional to the number of edges in the output path, as desired. ◀

2.1 Thorup-Zwick PRDO

A main component of our new PRDO relies on a well-known construction by Thorup and Zwick [22]. Given a weighted graph G with n vertices and an integer parameter $k \geq 1$, they constructed a PRDO with stretch $2k - 1$, query time $O(k)$ and size $O(kn^{1+1/k})$.

A useful property of the Thorup-Zwick (TZ) PRDO is that for every query, it returns a path that is contained in a sub-graph S of G, such that $|S| = O(kn^{1+1/k})$. Notice that since the stretch of this PRDO is $2k - 1$, then S must be a $(2k - 1)$-spanner of G. We call S the *underlying spanner* of the PRDO. One can compute the underlying spanner S either during the PRDO construction, or after its construction by querying the PRDO on every pair of vertices, and computing the union of the resulting paths.

A result of [23] improved the query time of the TZ PRDO to $O(\log k)$ instead of $O(k)$, while returning the same path that the TZ PRDO returns. Indeed, when we use here the TZ PRDO, we consider its query time to be $O(\log k)$. The following theorem concludes this discussion.

▶ **Theorem 2** (By [22] and [23]). *Let G be an undirected weighted graph with n vertices, and let $k \geq 1$ be an integer parameter. There is a PRDO for G with stretch $2k - 1$, query time $O(\log k)$ and size $O(kn^{1+1/k})$, with an underlying spanner of the same size.*

3 Path-Reporting Distance Oracle for Unweighted Graphs

In this section we introduce a simple variant of our construction, tailored for unweighted graphs. We first apply a simple clustering, and store a BFS (Breadth First Search) tree for each cluster. We next apply the TZ PRDO on the resulting cluster-graph. Finally, each query (u, v) is answered by taking the path in the cluster-graph between the clusters containing u and v, and completing it to a path in G using the BFS trees.

3.1 Clustering

We start by dividing the graph into clusters, using the following lemma.

▶ **Lemma 3.** *Let $G = (V, E)$ be an undirected unweighted graph with n vertices. Let $k \in [1, n]$ be some integer. There is an algorithm that finds a partition $V = \bigcup_{i=1}^{q} C_i$, such that every C_i has a spanning tree $T_i = (C_i, E_i)$ with root r_i, where $E_i \subseteq E$ and for every $v \in C_i$, $d_{T_i}(v, r_i) \leq k$. In addition, the number of sets in this partition, q, is at most $\frac{n}{k}$.*

Proof. Fix some $r \in V$, and let $T = (V, E_T)$ be the BFS tree with r as a root. The tree T is actually the shortest paths tree from r in G, and so the path from every $v \in V$ to r in T is of length exactly $d_G(v, r)$, i.e., $d_T(v, r) = d_G(v, r)$. If every vertex $v \in V$ satisfies $d_G(v, r) \leq k$, then we can return the trivial partition $\{V\}$, with the spanning tree T and root r.

Otherwise, let v be the furthest leaf of T from r, that is, v maximizes the length $d_T(v, r)$. We know that $d_T(v, r) > k$, and since G is unweighted, there is a vertex r' on the path in T from v to r, with $d_T(v, r') = k$. Denote by T' the sub-tree of T rooted at r'.

Let $u \in V$ be a vertex in T'. Since r' is on the path from u to r, and on the path from v to r, we have

$$d_{T'}(u,r') = d_T(u,r) - d_T(r',r) \leq d_T(v,r) - d_T(r',r) = d_T(v,r') = k \ .$$

Therefore, if C is the set of vertices of T', we can return C as one of the sets in the desired partition, where its spanning tree is T' and its root is r'. We then delete C from G and continue recursively.

Note that the tree T' contains the path from v to r', which is of length k. Since G is unweighted, that means that T' contains at least k vertices, and so does C. Hence, the number of vertices in the graph, after the deletion of C, is at most $n-k$. Notice also that the tree T is still a tree, after the removal of T', thus the remaining graph is still connected. As a result, we can assume that our algorithm recursively partitions the remaining graph into at most $\frac{n-k}{k} = \frac{n}{k} - 1$ sets, with spanning trees and roots as desired. Together with the last set C, we obtain a partition into at most $\frac{n}{k}$ parts, with the wanted properties. ◀

Given the unweighted graph $G = (V,E)$ and the integer k, let $\mathcal{C}$ be a partition as in Lemma 3. For every $C \in \mathcal{C}$, let $T[C]$ and $r[C]$ be the spanning tree of C and its root. We define a new graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ as follows.

▶ **Definition 4.** *The graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ is defined as follows. The set $\mathcal{E}$ consists of all the pairs $\{C, C'\}$, where $C, C' \in \mathcal{C}$, such that there is an edge in G between C, C'.*

Given an edge $\{C, C'\} \in \mathcal{E}$, we denote by $e(C, C')$ the edge $\{x, y\}$ of G, where $\{x, y\}$ is some choice of an edge that satisfies $x \in C$, $y \in C'$.

We denote by F the forest that consists of the disjoint union of the trees $T[C]$, for every $C \in \mathcal{C}$. For a vertex $x \in V$, define $h(x)$ to be the height of x in the tree $T[C]$ such that $x \in C$, and $p(x)$ its parent in this tree.

3.2 Stretch Analysis

Fix any cluster C, let $T = T[C]$ be its spanning tree with root $r = r[C]$. For any two vertices $a, b \in T$, the unique path between them is a sub-path of the union between the two paths from a to r and from b to r. Both of these paths are of length at most k. Hence, the resulting path is of length at most $2k$, and this path is exactly the one that the algorithm from Lemma 1 returns.

▶ **Lemma 5.** *There is an algorithm that given two vertices $u, v \in V$, and a simple path $Q = (C_1, C_2, ..., C_t)$ in the graph $\mathcal{H}$, such that u is in C_1 and v is in C_t, returns a path P in G between u and v, with number of edges*

$$|P| \leq t \cdot (2k+1) \ .$$

The running time of the algorithm is proportional to the number of edges in the output path. The required information for the algorithm is the set $\{h(x), p(x)\}_{x \in V}$, and the set $\{e(C_j, C_{j+1})\}_{j=1}^{t-1}$.

Proof. Given the edges $\{C_j, C_{j+1}\}$, the set $\{e(C_j, C_{j+1})\}_{j=1}^{t-1}$ can be used to find $x_j, y_j \in V$ (vertices of the original graph $G = (V,E)$), such that $x_j \in C_j$, $y_j \in C_{j+1}$ and $\{x_j, y_j\} \in E$. Define also $y_0 = u, x_t = v$. For every $j \in [1,t]$, using the set $\{h(x), p(x)\}_{x \in V}$, we can use Lemma 1 to find a path P_j in G between y_{j-1} and x_j, with length at most $2k$. Finding all of these paths takes time that is proportional to the sum of lengths of these paths.

The returned path by this algorithm is

$$P = P_1 \circ \{x_1, y_1\} \circ P_2 \circ \{x_2, y_2\} \circ \cdots \circ \{x_{t-1}, y_{t-1}\} \circ P_t \ .$$

The time needed to report this path is $O(\sum_{j=1}^{t} |P_j|) = O(|P|)$. The length of this path is

$$t - 1 + \sum_{j=1}^{t} |P_j| \leq t - 1 + t \cdot 2k < t \cdot (2k+1) \ .$$

This concludes the proof of the lemma. ◀

3.3 A PRDO for Unweighted Graphs

We are now ready to introduce the construction of our small size PRDO.

▶ **Theorem 6.** *Let $G = (V, E)$ be an undirected unweighted graph with n vertices, and let $k \in [1, \log n]$ be some integer parameter. There is a path-reporting distance oracle for G with stretch $2k(2k+1) = O(k^2)$, query time $O(\log k)$ and size $O(n^{1+\frac{1}{k}})$.*

Proof. Denote by TZ the PRDO from Theorem 2 with the parameter k, when constructed over the graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ (the clustering $\mathcal{C}$ is constructed with k as the radius[4]). Let $S_{TZ} \subseteq \mathcal{E}$ be the set of the edges of the underlying spanner of TZ. In addition, for a given vertex $x \in V$, denote by $C(x)$ the vertex of $\mathcal{H}$ (i.e., cluster) that contains x. Recall also that $h(x)$ is the height of x in the tree spanning $C(x)$ and $p(x)$ denotes the parent of x in this tree.

We define our new PRDO for the undirected unweighted graph $G = (V, E)$. This PRDO contains the following information.

1. The TZ PRDO.
2. The set $\{e(C, C') \mid \{C, C'\} \in S_{TZ}\}$.
3. The variables $\{h(x), p(x)\}_{x \in V}$.
4. The variables $\{C(x)\}_{x \in V}$.

Given a query $(u, v) \in V^2$, our PRDO queries TZ on the vertices $C(u), C(v)$ of $\mathcal{H}$. Let $Q = (C(u) = C_1, C_2, ..., C_t = C(v))$ be the resulting path, and note that all of its edges are in S_{TZ}. Then, using the sets $\{e(C_j, C_{j+1})\}_{j=1}^{t-1} \subseteq \{e(C, C') \mid \{C, C'\} \in S_{TZ}\}$ and $\{h(x), p(x)\}_{x \in V}$, we find a path P in G between u, v using the algorithm from Lemma 5. The resulting path P has length of

$$|P| \leq (|Q| + 1)(2k+1) = t \cdot (2k+1) \ ,$$

and it is returned as an output to the query.

Note that the path Q that TZ returned satisfies $|Q| = t - 1 \leq (2k-1)|R|$, where R is the shortest path in $\mathcal{H}$ between $C(u)$ and $C(v)$. Let $P_{u,v}$ be the actual shortest path in G between u and v. Suppose that the vertices of $\mathcal{H}$ that $P_{u,v}$ passes through, by the order that it passes through them, are $(T_1, T_2, ...T_q)$. By the definition of $\mathcal{H}$, there must be an edge $\{T_j, T_{j+1}\}$ in $\mathcal{H}$ for every $j \in [1, q-1]$. Hence, $R' = (T_1, T_2, ..., T_q)$ is a path in $\mathcal{H}$, between $T_1 = C(u)$ and $T_q = C(v)$, with length of at most $|P_{u,v}| = d_G(u, v)$. Since R is the shortest path in $\mathcal{H}$ between $C(u)$ and $C(v)$, we have $|R| \leq d_G(u, v)$.

[4] Actually, by constructing the clustering $\mathcal{C}$ with radius $\frac{k}{8}$ instead of k, the stretch of our new PRDO decreases from $4k^2$ to k^2. In the same way, one can achieve an arbitrarily small leading constant in the stretch.

As a result,

$$\begin{aligned}|P| &\leq t\cdot(2k+1)\\ &\leq ((2k-1)|R|+1)(2k+1)\\ &\leq ((2k-1)d_G(u,v)+1)(2k+1)\\ &= (4k^2-1)d_G(u,v)+2k+1\\ &\leq (4k^2+2k)d_G(u,v)=2k(2k+1)d_G(u,v)\ .\end{aligned}$$

Thus, the stretch of our PRDO is at most $2k(2k+1)$.

The query time of our oracle consists of the time required for running a query of TZ, and of the time required for finding the path P. By Theorem 2 and Lemma 5, the total time for these two computations is $O(\log k+|P|)$ which is $O(\log k)$ by our conventional PRDO notations.

As for the size of our PRDO, note that the variables $\{h(x),p(x)\}_{x\in V}$ (item 3 in the description of the oracle) can be stored using only $O(n)$ space. The size of the set $\{e(C,C')\mid\{C,C'\}\in S_{TZ}\}$ equals to the size of S_{TZ}. Therefore, by Theorem 2, the size of TZ, as well as the size of this set (items 1 and 2), is

$$O(k|\mathcal{C}|^{1+\frac{1}{k}})\ .$$

Recall that by Lemma 3, the size of $\mathcal{C}$ is at most $\frac{n}{k}$. We conclude that the total size of our new PRDO is

$$O(n+k\cdot(\frac{n}{k})^{1+\frac{1}{k}})=O(n^{1+\frac{1}{k}})\ .\qquad\blacktriangleleft$$

An Ultra-Compact PRDO for Unweighted Graphs. We can modify our PRDO for unweighted graphs, and get a PRDO of size $n+o(n)$. Here, the required storage for our PRDO is measured by *words* - each of size at most $\log n$ bits. Decreasing the size of our PRDO is done at the cost of increasing the query time and (slightly) the stretch. The details are deferred to the full version of this paper.

4 Path-Reporting Distance Oracle for Weighted Graphs

In this section we devise our PRDO for weighted graphs. The basic framework is similar to the unweighted case: create a clustering of the graph, select a spanning tree for each cluster, and then apply the TZ PRDO over the cluster-graph. To answer a query (u,v), we use the path in the cluster-graph between the clusters containing u,v, and complete it inside each cluster via the spanning trees edges.

The main differences from the unweighted setting are: 1) we use a more intricate clustering, *Borůvka's clustering*, and 2) the trees spanning each cluster are not BFS trees, but are a subset of the MST (Minimum Spanning Tree) of the graph. These changes are needed in order to achieve the desired properties - that the number of clusters is small enough, while the stretch caused by going through the spanning trees of the clusters is controlled.

4.1 Clustering via Borůvka Forests

In this section we construct a clustering via a spanning forest of a graph. This construction is based on the well-known algorithm by Borůvka for finding a minimum spanning tree in a graph. Similar constructions can be found in [16, 1, 2].

▶ **Definition 7.** *Given an undirected weighted graph $G = (V, E)$, and a vertex $v \in V$, we denote by e_v the minimum-weight edge among the adjacent edges to v in the graph G. If there is more than one edge with this minimum weight, e_v is chosen to be the one that is the smallest lexicographically.*

▶ **Definition 8.** *Given an undirected weighted graph $G = (V, E)$, the* Borůvka Forest *of G is the sub-graph $G' = (V, E')$ of G, where*

$$E' = \{e_v \mid v \in V\} \ .$$

Each connected component T of G' is called a Borůvka Tree. *The* root *of T is chosen to be one of the adjacent vertices to the minimum-weight edge in T (if there are several such minimum-weight edges, we pick the smallest one lexicographically, and the choice between its two adjacent vertices is arbitrary).*

To justify the use of the words “forest” and “tree”, we prove the following lemma.

▶ **Lemma 9.** *The graph G' is a forest. Moreover, if T is a tree in G', x is a vertex of T, and $p(x)$ is x's parent in T (that is, the next vertex on the unique path from x to the root of T), then $\{x, p(x)\} = e_x$.*

Proof. First, we prove that G' is a forest. Seeking contradiction, assume that G' contains a cycle C, and let $\{u, v\}$ be the heaviest edge in C (if there are several edges with the largest weight, choose the one that is largest lexicographically). Note that since u has at least one adjacent edge in C, that is lighter than $\{u, v\}$, then it cannot be that $e_u = \{u, v\}$ (recall that e_u is the lightest edge adjacent to u). Similarly, it cannot be that $e_v = \{u, v\}$. Hence, we get a contradiction to the fact that $\{u, v\}$ is an edge of G' - since every such edge must be the edge e_v of one of its endpoints v.

Next, Let T be a tree in G', denote its root by v, and let $x \neq v$ be a vertex of T. We prove by induction over the height of x, $h(x)$, which is the number of edges in the unique path between x and v in T.

When $h(x) = 1$, we have $p(x) = v$. We consider two cases. If $\{x, v\}$ is the minimum-weight edge in T, then by definition e_x must be this edge, i.e., $e_x = \{x, p(x)\}$. If $\{x, v\}$ is not the minimum-weight edge in T, then e_v must be some other adjacent edge to v, thus $e_v \neq \{x, v\}$. But then, the reason that $\{x, v\}$ is in E' must be that $\{x, p(x)\} = \{x, v\} = e_x$.

For $h(x) > 1$, notice that $h(p(x)) = h(x) - 1$, and therefore by the induction hypothesis, $\{p(x), p(p(x))\} = e_{p(x)}$. But then, the edge $\{x, p(x)\}$ cannot be equal to $e_{p(x)}$, so it must be equal to e_x. ◀

The following lemma bounds the number of connected components (i.e., trees) in G'.

▶ **Lemma 10.** *The number of connected components in G' is at most $\frac{1}{2}|V|$.*

Proof. Let $C = (V_C, E'_C)$ be a connected component of G', and let $x \in V_C$. The edge $e_x = \{x, y\}$ is in C, hence y is also a vertex of C. In particular, $|V_C| \geq 2$. Hence, if $\{C_i\}_{i=1}^t$ are the connected components of G', then $|V_{C_i}| \geq 2$ for every $i \in [1, t]$. Thus,

$$\frac{1}{2}|V| = \frac{1}{2}\sum_{i=1}^{t}|V_{C_i}| \geq \frac{1}{2} \cdot 2t = t \ . \qquad ◀$$

Next, we trim the trees in the Borůvka forest so that each of them will be a *star*, instead of a general tree. For this purpose, we will need the following definitions.

▶ **Definition 11.** *Let G' be the Borůvka forest of G. For a vertex $x \in V$, denote by $h(x)$ the height of x in the Borůvka tree containing it. Define*

$$E'_0 = \{\{a,b\} \in E' \mid \min\{h(a),h(b)\} = 0 \mod 2\} ,$$

$$E'_1 = \{\{a,b\} \in E' \mid \min\{h(a),h(b)\} = 1 \mod 2\} .$$

We denote by E'' the largest set among these two.

Given an undirected weighted graph $G = (V,E)$, the Partial Borůvka Forest *of G is the graph $G'' = (V,E'')$.*

▶ **Definition 12.** *A* Star *is a rooted tree $S = (V_S, E_S)$ with root v such that for every $x \in V_S \setminus \{v\}$, $\{x,v\} \in E_S$.*

▶ **Lemma 13.** *The partial Borůvka forest $G'' = (V,E'')$ is a forest, where every tree is a star. In addition, if S is a star in G'', x is its root and $z \neq x$ is some other vertex of S, then $\{z,x\} = e_z$.*

Proof. Notice that G'' is a sub-graph of the Borůvka forest G', hence G'' is also a forest. We assume that $E'' = E'_0$, and the proof for the case where $E'' = E'_1$ is symmetric.

Let T be a tree in G', with root r. Note that for any vertex $x \neq r$ we always have $h(x) = h(p(x)) + 1$, and thus $\min\{h(x), h(p(x))\} = h(p(x)) = h(x) - 1$. We conclude that if $h(x)$ is even, then $\{x,p(x)\} \notin E''$, and if $h(x)$ is odd, then $\{x,p(x)\} \in E''$.

Now let S be a tree in G'', and let x be the vertex in S that has minimal $h(x)$. It cannot be that $h(x)$ is odd, otherwise $p(x)$ is connected to x in E'', thus $p(x)$ is also in S and has a smaller value of $h(p(x)) = h(x) - 1$. Therefore, $h(x)$ is even. By the discussion above we know that all of the children of x in T (y's that satisfy $p(y) = x$) have an edge in E'' to x, but their children have no such edge. That is, S is a star with x as a root, where all the other vertices in S are the children of x.

The last part of the lemma follows from the fact that we just proved, that the only other vertices in a star S with a root x, are the children of x. By Lemma 9, for every such child z, the edge $\{z,x\} = \{z,p(z)\} = e_z$. ◀

The following lemma bounds the number of trees in the partial Borůvka forest of a graph.

▶ **Lemma 14.** *The number of stars in G'' is at most $\frac{3}{4}|V|$.*

Proof. Recall the Borůvka forest $G' = (V,E')$. In every spanning forest (V,F) of a graph $G = (V,E)$, the number of trees is exactly $|V| - |F|$. Thus, by Lemma 10, we get

$$|V| - |E'| \leq \frac{1}{2}|V| ,$$

and therefore $|E'| \geq \frac{1}{2}|V|$. By the definition of E'', it contains at least half of these edges (since it equals to the larger set among two sets that cover the entire set E'). We conclude that $|E''| \geq \frac{1}{4}|V|$, and the number of trees in G'', which are stars, is

$$|V| - |E''| \leq |V| - \frac{1}{4}|V| = \frac{3}{4}|V| .$$ ◀

4.1.1 A Hierarchy of Forests

Given an undirected weighted graph $G = (V,E)$, we construct a sequence of forests $\{F_i = (V,E_i)\}_{i=0}^{l}$, where the integer parameter $l \geq 0$ will be determined later. For $i = 0$, define $E_0 = \emptyset$. Then, for every $i \in [0,l]$, define the cluster-graph $\mathcal{H}_i = (\mathcal{C}_i, \mathcal{E}_i)$ as follows.

The set $\mathcal{C}_i$ is defined to be the set of the disjoint trees of the forest F_i. For every $T, T' \in \mathcal{C}_i$, denote by $e(T, T')$ the minimum-weight edge in E among the edges between T and T'. If there is no such edge, denote $e(T, T') = \perp$. Then define

$$\mathcal{E}_i = \{\{T, T'\} \mid e(T, T') \neq \perp\} .$$

The weight of an edge $\{T, T'\} \in \mathcal{E}_i$ is defined to be the same as the weight of the edge $e(T, T') \in E$.

For any $i \in [0, l]$, given the graph $\mathcal{H}_i$, let $\mathcal{H}''_i = (\mathcal{C}_i, \mathcal{E}''_i)$ be the partial Borůvka forest of $\mathcal{H}_i$. The graph $\mathcal{H}''_i$ is a disjoint union of stars. Let $\mathcal{S}$ be such star and let T_0 be its root.

Define the tree Z in G to be the tree that is formed by the union of the trees in $\mathcal{S}$ and the edges $e(T, T_0)$, for every $T \neq T_0$ in $\mathcal{S}$. The root of the tree Z is defined to be the root of T_0. Finally, for any $i \in [0, l-1]$, the forest $F_{i+1} = (V, E_{i+1})$ is defined to be the disjoint union of the rooted trees Z that are formed as was described, for all stars in $\mathcal{H}''_i$.

▶ **Lemma 15.** *For every $i \in [0, l]$, the forest F_i has at most $(\frac{3}{4})^i |V|$ trees.*

Proof. We prove the lemma by induction over $i \in [0, l]$. For $i = 0$, F_0 is defined to be the graph $(V, \emptyset)$, so the number of trees in F_0 is $|V|$ and the claim holds.

For $i > 0$, recall the graphs $\mathcal{H}_{i-1}$ and $\mathcal{H}''_{i-1}$ that were used in the definition of F_i. By the induction hypothesis, F_{i-1} consists of at most $(\frac{3}{4})^{i-1}|V|$ trees, which are exactly the vertices of $\mathcal{H}_{i-1}$. Then, by Lemma 14, the graph $\mathcal{H}''_{i-1}$ has at most $\frac{3}{4} \cdot (\frac{3}{4})^{i-1}|V| = (\frac{3}{4})^i |V|$ stars. The forest F_i consists of a single tree Z for every star $\mathcal{S}$ in $\mathcal{H}''_{i-1}$, thus the number of trees in F_i is at most $(\frac{3}{4})^i |V|$, as desired. ◀

4.2 Stretch Analysis

Due to space considerations, we only state here the main lemma that will be used for bounding the stretch of our PRDO, without proof. The proof of this lemma, as well as some other lemmata , appears in the full version of this paper.

In the next lemma, we use the notations $p_i(x)$ and $h_i(x)$ to denote the parent and the height of x in the tree of F_i that contains x.

▶ **Lemma 16.** *There is an algorithm that given two vertices $u, v \in V$, and a simple path $Q = (S_1, S_2, ..., S_t)$ in the graph $\mathcal{H}_i$, such that u is in S_1 and v is in S_t, returns a path P in G between u and v, with*

$$w(P) \leq 3^{i+1}(d_G(u, v) + w(Q)) .$$

The running time of the algorithm is proportional to the number of edges in the output path P. The required information for the algorithm is the set $\{h_i(x), p_i(x)\}_{x \in V}$, and the set $\{e(S_j, S_{j+1})\}_{j=1}^{t-1}$.

4.3 A PRDO for Weighted Graphs

We are now ready to introduce our small size path-reporting distance oracle.

▶ **Theorem 17.** *Let $G = (V, E)$ be an undirected weighted graph with n vertices, and let $k \geq 1$ be an integer parameter. There is a path-reporting distance oracle for G with stretch $k^{\log_{4/3} 4} < k^{4.82}$, query time $O(\log k)$ and size $O(n^{1+\frac{1}{k}})$.*

Proof. Given the graph $G = (V, E)$, we construct the hierarchy of forests $\{F_i\}_{i=0}^{l}$ from Section 4.1.1, where $l = \lfloor \log_{4/3} k \rfloor - 2$. Consider the graph $\mathcal{H}_l = (\mathcal{C}_l, \mathcal{E}_l)$ that is defined in Section 4.1.1. For every $x \in V$, denote by $h_l(x)$ the number of edges in the unique path from x to the root of the tree of F_l that x belongs to. Let $p_l(x)$ be the parent of x in that tree. Lastly, let $S(x)$ be the vertex of $\mathcal{H}_l$ (i.e., tree) that contains x.

Denote by TZ the PRDO from Theorem 2 with the parameter k, when constructed over the graph $\mathcal{H}_l$. Let $S_{TZ} \subseteq \mathcal{E}_l$ be the set of edges of the underlying spanner of TZ.

Our new PRDO D stores the following information.

1. The oracle TZ.
2. The set $\{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$.
3. The variables $\{h_l(x), p_l(x)\}_{x \in V}$.
4. The variables $\{S(x)\}_{x \in V}$.

Given a query $(u, v) \in V^2$, the oracle D queries TZ on the vertices $S(u), S(v)$ of $\mathcal{H}_l$. Let $Q = (S(u) = S_1, S_2, ..., S_t = S(v))$ be the resulting path, and note that all of its edges are in S_{TZ}. Then, using the sets $\{e(S_j, S_{j+1})\}_{j=1}^{t-1} \subseteq \{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$ and $\{h_l(x), p_l(x)\}_{x \in V}$, the oracle D uses the algorithm from Lemma 16 to find a path P in G between u, v with

$$w(P) \leq 3^{l+1}(d_G(u, v) + w(Q)) \ .$$

The path P is returned as an output to the query. Note that the path Q that TZ returned satisfies

$$w(Q) \leq (2k - 1)w(R) \ ,$$

where R is the shortest path in $\mathcal{H}_l$ between $S(u)$ and $S(v)$. Similarly to the proof of Theorem 17, it is easy to verify that $w(R) \leq d_G(u, v)$.

As a result,

$$\begin{aligned} w(P) &\leq 3^{l+1}(d_G(u, v) + w(Q)) \\ &\leq 3^{l+1}(d_G(u, v) + (2k - 1)w(R)) \\ &\leq 3^{l+1}(d_G(u, v) + (2k - 1)d_G(u, v)) \\ &= 2k \cdot 3^{l+1} d_G(u, v) \\ &\leq 2k \cdot 3^{\log_{4/3} k - 1} d_G(u, v) \\ &< k^{1+\log_{4/3} 3} d_G(u, v) = k^{\log_{4/3} 4} d_G(u, v) \ . \end{aligned}$$

Thus the stretch of our PRDO is smaller than $k^{\log_{4/3} 4}$.

The query time of our oracle consists of the time required for running a query of TZ, and of the time required for computing the resulting path P by Lemma 16. By Theorem 2 and Lemma 16, the total time for these two computations is $O(\log k + |P|)$, which is $O(\log k)$ by our conventional PRDO notations.

As for the size of the PRDO D, note that the variables $\{h_l(x), p_l(x), S(x)\}_{x \in V}$ (items 3 and 4 in the description of D) can be stored using only $O(n)$ space. The size of the set $\{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$ equals to the size of S_{TZ}. Therefore, by Theorem 2, the size of TZ, as well as the size of this set (items 1 and 2 in the description of D), is

$$O(k|\mathcal{C}_l|^{1+\frac{1}{k}}) \ .$$

Recall that $\mathcal{C}_l$ is the set of vertices of $\mathcal{H}_l$. This set consists of the trees in the forest F_l. By Lemma 15, the number of these trees is at most

$$(\frac{3}{4})^l|V| = (\frac{3}{4})^{\lfloor \log_{4/3} k \rfloor - 2}n \leq (\frac{3}{4})^{\log_{4/3} k - 3}n = \frac{64n}{27k} \ .$$

Hence, the total size of our PRDO is

$$O(n + k \cdot (\frac{64n}{27k})^{1+\frac{1}{k}}) = O(n + n^{1+\frac{1}{k}}) = O(n^{1+\frac{1}{k}}) \ . \qquad \blacktriangleleft$$

An Ultra-Compact PRDO for Weighted Graphs. As in the unweighted version, the PRDO presented above can be fine-tuned into an ultra-compact PRDO (with size $n + o(n)$), at the cost of increasing the stretch and the query time. The details are deferred to the full version of this paper.

5 Pairwise Path-Reporting Distance Oracle

Our construction of a pairwise PRDO relies on the pairwise spanner of Kogan and Parter, from their recent paper [19] (in which the pairwise spanner is called a "near-exact preserver"). One of their useful results, that they also relied on for constructing their pairwise spanners, is the following lemma on hopsets. We first recall the definition of hopsets.

Let $G = (V, E)$ be a weighted undirected graph. For vertices $u, v \in V$ and some positive integer β, $d_G^{(\beta)}(u, v)$, denotes the weight of the lightest path between u and v in G, among the paths that have at most β edges. An (α, β)-*hopset* is a set $H \subseteq \binom{V}{2}$, such that for every two vertices $u, v \in V$,

$$d_G(u, v) \leq d_{G \cup H}^{(\beta)}(u, v) \leq \alpha \cdot d_G(u, v) \ ,$$

where the weight of an edge $(x, y) \in H$ is defined to be $d_G(x, y)$.

The proof of the following lemma can be found in [19].

▶ **Lemma 18** (Lemma 4.4 from [19]). *Let $G = (V, E)$ be an undirected weighted graph on n vertices, and let $k, D \geq 1$ be integer parameters. For every $0 < \epsilon < 1$, there exists a $(1 + \epsilon, \beta)$-hopset H for G, where $\beta = O(\frac{\log k}{\epsilon})^{\log k} \cdot D$ and*

$$|H| = O\left(\left(\frac{n \log n}{D}\right)^{1+\frac{1}{k}}\right) \ .$$

Similarly to the constructions in [19], we now show how a pairwise PRDO can be produced, using the hopsets from Lemma 18. We will use the notation $\beta(\epsilon, k) = O(\frac{\log k}{\epsilon})^{\log k}$ for brevity.

▶ **Theorem 19.** *Let $G = (V, E)$ be an undirected weighted graph on n vertices and let $\mathcal{P} \subseteq V^2$ be a set of pairs of vertices. For every $\epsilon \in (0, 1)$, there exists a pairwise path-reporting distance oracle with stretch $1 + \epsilon$, query time $O(1)$ and size*

$$O\left(\frac{\log n \cdot (\log \log n)^2}{\epsilon}\right)^{\log \log n} \cdot \tilde{O}\big(|\mathcal{P}| + n\big) = n^{o_\epsilon(1)} \cdot O(n + |\mathcal{P}|) \ .$$

Proof. Let $n = D_0 > D_1 > \cdots > D_l = 2$ be some sequence of integer parameters that will be determined later. Denote $k = \log n$, and for a given $\epsilon \in (0, 1)$, denote $\epsilon' = \frac{\epsilon}{2(l+1)}$. Let $H_0, H_1, ..., H_l$ be the resulting hopsets when applying Lemma 18 on $\epsilon', k = \log n$ and $D_0, D_1, ..., D_l$ respectively. That is, H_i is a $(1+\epsilon', \beta_i)$-hopset with size $O((\frac{n \log n}{D_i})^{1+\frac{1}{k}})$, where $\beta_i = \beta(\epsilon', k) \cdot D_i$. For $i = 0$, note that $\beta_i \geq n$, thus we can simply assume that $H_0 = \emptyset$ (if it is not the case, we *define* H_0 to be $\emptyset$, which is a $(1, n)$-hopset).

We now define our oracle D to contain the following information. For every $i \in [1, l]$ and for every $(x, y) \in H_i$, let $Q_{x,y}$ be the shortest path in $G \cup H_{i-1}$ between x, y, among the paths that contain at most β_{i-1} edges. In addition, for every $(x, y) \in \mathcal{P}$, let $P_{x,y}$ be the shortest path in $G \cup H_l$ between x, y, among the paths with at most β_l edges. Our oracle D stores all of these paths: $\bigcup_{i=1}^{l} \{Q_{x,y}\}_{(x,y)\in H_i} \cup \{P_{x,y}\}_{(x,y)\in\mathcal{P}}$.

Given a query $(u, v) \in \mathcal{P}$, we find the path $P_l = P_{u,v} \subseteq G \cup H_l$ that is stored in D. Then, we replace every edge $(x, y) \in H_l$ on P_l by the corresponding path $Q_{x,y} \subseteq G \cup H_{l-1}$. The result is a path P_{l-1} between u, v in $G \cup H_{l-1}$. Every edge $(x, y) \in H_{l-1}$ on P_{l-1} is then replaced by the path $Q_{x,y} \subseteq G \cup H_{l-2}$, to get a path P_{l-2} between u, v in $G \cup H_{l-2}$. We continue in the same way, until finally reaching to a path P_0 between u, v in the graph $G \cup H_0 = G$. We return P_0 as an output to the query.

By the hopset property, we know that

$$w(P_l) = w(P_{u,v}) = d^{(\beta_l)}_{G\cup H_l}(u, v) \leq (1 + \epsilon')d_G(u, v) \ .$$

Similarly, every $(x, y) \in P_l$ that is also in H_l, is replaced with the path $Q_{x,y}$, that has a weight of

$$w(Q_{x,y}) = d^{(\beta_{l-1})}_{G\cup H_{l-1}}(x, y) \leq (1 + \epsilon')d_G(x, y) = (1 + \epsilon')w(x, y) \ .$$

Thus, the resulting path P_{l-1} has a weight of at most $1 + \epsilon'$ times the weight of P_l, that is

$$w(P_{l-1}) \leq (1 + \epsilon')w(P_l) \leq (1 + \epsilon')^2 d_G(u, v) \ .$$

Proceeding in the same way, we conclude that $w(P_0) \leq (1+\epsilon')^{l+1}d_G(u, v)$. Hence, the stretch of our distance oracle is

$$(1 + \epsilon')^{l+1} = (1 + \frac{\epsilon}{2(l + 1)})^{l+1} \leq e^{\frac{\epsilon}{2}} \leq 1 + \epsilon \ .$$

For analysing the query time of our distance oracle, we can think of the query algorithm as a single pass on the path P_l, where every time that an edge of H_l is reached, we replace it with the appropriate path $Q_{x,y}$, and continue inside $Q_{x,y}$ recursively. Since every step produces an edge that will appear in the output path, the query time is proportional to this output path. Observe, however, that the resulting path is actually a walk, and not necessarily a simple path. By our convention of writing the query time of PRDOs, this query time is $O(1)$.

Lastly, we analyse the size of our pairwise PRDO. Note that by their definitions, the paths $P_{x,y}$, for every $(x, y) \in \mathcal{P}$ are of length at most β_l. Similarly, the length of $Q_{x,y}$, for $(x, y) \in H_i$ is at most β_{i-1}. Therefore, the total space required for storing these paths is at most

$$\begin{aligned}
|\mathcal{P}| \cdot \beta_l + \sum_{i=1}^{l} |H_i| \cdot \beta_{i-1} &= |\mathcal{P}| \cdot \beta(\epsilon', k) \cdot D_l + \sum_{i=1}^{l} O\left(\left(\frac{n \log n}{D_i}\right)^{1+\frac{1}{k}}\right) \cdot \beta(\epsilon', k) \cdot D_{i-1} \\
&= \beta(\epsilon', k) \cdot \left(|\mathcal{P}| \cdot 2 + \sum_{i=1}^{l} O\left(\left(\frac{n \log n}{D_i}\right)^{1+\frac{1}{k}}\right) \cdot D_{i-1}\right) \\
&= \beta(\epsilon', k) \cdot O\big(|\mathcal{P}| + (n \log n)^{1+\frac{1}{k}} \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big) \\
&= O\left(\frac{\log k}{\epsilon/2l}\right)^{\log k} \cdot O\big(|\mathcal{P}| + (n \log n)^{1+\frac{1}{\log n}} \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big)
\end{aligned}$$

$$= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot O\big(|\mathcal{P}| + n \log n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big)$$

$$= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big) \ .$$

For making the last term small, we choose $D_i = \left\lceil n^{(\frac{k}{k+1})^i} \right\rceil$, and thus $\frac{D_{i-1}}{D_i^{1+\frac{1}{k}}} \leq \frac{n^{(\frac{k}{k+1})^{i-1}}+1}{n^{(\frac{k}{k+1})^i \cdot (1+\frac{1}{k})}} = \frac{n^{(\frac{k}{k+1})^{i-1}}+1}{n^{(\frac{k}{k+1})^{i-1}}} \leq 2$. For this choice of D_i, since we want D_l to be 2, we must have $n^{(\frac{k}{k+1})^l} \leq 2$, that is, $l \geq \log_{\frac{k+1}{k}}(\log n)$. Notice that $\log_{\frac{k+1}{k}}(\log n) = \frac{\log\log n}{\log(1+\frac{1}{k})} \leq \frac{\log\log n}{\log(2^{\frac{1}{k}})} = k \log\log n$, thus we can choose $l = \lceil k \log\log n \rceil = \lceil \log n \cdot \log\log n \rceil$.

In conclusion, the size of our pairwise PRDO is at most

$$\begin{aligned} O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big) &= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} 2\big) \\ &= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + l \cdot n\big) \\ &= O\left(\frac{\log n \cdot (\log\log n)^2}{\epsilon}\right)^{\log\log n} \cdot \tilde{O}\big(|\mathcal{P}| + n\big) \\ &= n^{o_\epsilon(1)} \cdot O\big(|\mathcal{P}| + n\big) \end{aligned}$$ ◀

References

1 MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6864–6874, 2017. URL: `https://proceedings.neurips.cc/paper/2017/hash/2e1b24a664f5e9c18f407b2f9c73e821-Abstract.html`.

2 Amartya Shankha Biswas, Michal Dory, Mohsen Ghaffari, Slobodan Mitrović, and Yasamin Nazari. Massively parallel algorithms for distance approximation and spanners. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 118–128, 2021.

3 Greg Bodwin. New results on linear size distance preservers. *SIAM J. Comput.*, 50(2):662–673, 2021. `doi:10.1137/19M123662X`.

4 Greg Bodwin, Keerti Choudhary, Merav Parter, and Noa Shahar. New fault tolerant subset preservers. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.15`.

5 Greg Bodwin, Gary Hoppenworth, and Ohad Trabelsi. Bridge girth: A unifying notion in network design. *arXiv preprint*, 2022. `arXiv:2212.11944`.

6 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. *ACM Trans. Algorithms*, 17(4):36:1–36:24, 2021. `doi:10.1145/3490147`.

7 Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 654–663. ACM, 2014. `doi:10.1145/2591796.2591801`.

8 Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015. doi:10.1145/2746539.2746562.

9 D. Coppersmith and M. Elkin. Sparse source-wise and pair-wise distance preservers. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 660–669, 2005.

10 Marek Cygan, Fabrizio Grandoni, and Telikepalli Kavitha. On pairwise spanners. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 209–220. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.209.

11 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina Anna Zweig, editors, *Algorithmics of Large and Complex Networks – Design, Analysis, and Simulation [DFG priority program 1126]*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009. doi:10.1007/978-3-642-02094-0_7.

12 Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016. doi:10.1016/j.tcs.2016.07.038.

13 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016. doi:10.1145/2888397.

14 Michael Elkin and Idan Shabat. Path-reporting distance oracles with near-logarithmic stretch and linear size. *CoRR*, abs/2304.04445, 2023. doi:10.48550/arXiv.2304.04445.

15 P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and Applications (Proc. Sympos. Smolenice)*, pages 29–36, 1964.

16 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 202–219. SIAM, 2016.

17 Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12–16, 1996, Rockville, Maryland, USA*, pages 261–268. ACM, 1996. doi:10.1145/238355.238550.

18 Telikepalli Kavitha. New pairwise spanners. *Theory Comput. Syst.*, 61(4):1011–1036, 2017. doi:10.1007/s00224-016-9736-7.

19 Shimon Kogan and Merav Parter. Having hope in hops: New spanners, preservers and lower bounds for hopsets. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 766–777. IEEE, 2022. doi:10.1109/FOCS54457.2022.00078.

20 Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007.

21 Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46:45:1–31, 2014. doi:10.1145/2530531.

22 M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 183–192, 2001.

23 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–549. SIAM, 2013.

24 Christos D. Zaroliagis. Engineering algorithms for large network applications. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms – 2008 Edition*. Springer, 2008. doi:10.1007/978-0-387-30162-4_125.

Toward Grünbaum's Conjecture

Christian Ortlieb
Institute of Computer Science, University of Rostock, Germany

Jens M. Schmidt
Institute of Computer Science, University of Rostock, Germany

Abstract

Given a spanning tree T of a planar graph G, the *co-tree* of T is the spanning tree of the dual graph G^* with edge set $(E(G) - E(T))^*$. Grünbaum conjectured in 1970 that every planar 3-connected graph G contains a spanning tree T such that both T and its co-tree have maximum degree at most 3.

While Grünbaum's conjecture remains open, Biedl proved that there is a spanning tree T such that T and its co-tree have maximum degree at most 5. By using new structural insights into Schnyder woods, we prove that there is a spanning tree T such that T and its co-tree have maximum degree at most 4. This tree can be computed in linear time.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Planar graph, spanning tree, maximum degree, Schnyder wood

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.37

Funding This research is supported by the grant SCHM 3186/2-1 (401348462) from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation).

1 Introduction

Let a *k-tree* be a spanning tree whose maximum degree is at most k. In 1966, Barnette proved the fundamental theorem that every planar 3-connected graph contains a 3-tree [3]. Both assumptions in this theorem are essential in the sense that the statement fails for arbitrary non-planar graphs (as the arbitrarily high degree in any spanning tree of the complete bipartite graphs $K_{3,n-3}$ show) as well as for graphs that are not 3-connected (as the planar graphs $K_{2,n-2}$ show).

Since then, Barnette's theorem has been extended and generalized in several directions. First, one may try to relax the 3-connectedness assumption: Indeed, Barnette's original proof holds for the slightly more general class of *circuit graphs*[1], and may also be extended to arbitrary planar graphs G in form of a local version that guarantees for every 3-connected[2] vertex set X of G a (not necessarily spanning) tree of G that has maximum degree at most 3 and contains X [6]. Alternatively, one may relax the planarity assumption. Ota and Ozeki [22] proved that for every $k \geq 3$, every 3-connected graph with no $K_{3,k}$-minor contains a $(k-1)$-tree if k is even and a k-tree if k is odd. Further sufficient conditions for the existence of k-trees may be found in the survey [23].

Second, one may see spanning trees as 1-connected spanning subgraphs and generalize these to k-connected spanning subgraphs for any $k > 1$. In this direction, Barnette [4] proved that every planar 3-connected planar graph contains a 2-connected spanning subgraph whose maximum degree is at most 15, and Gao [17] improved this result subsequently to the tight bound of maximum degree at most 6. Interestingly, Gao showed that his result holds as well for the 3-connected graphs that are embeddable on the projective plane, the torus or the Klein bottle.

[1] that is, planar internally 3-connected graphs with a designated outer face

[2] $X \subseteq V(G)$ such that G contains three internally vertex-disjoint paths between every two vertices of X

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 37; pp. 37:1–37:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Third, one may try to strengthen the 3-tree in question. A recent alternative proof of Barnette's theorem based on canonical orderings by Biedl [5, Corollary 1] (which was also mentioned by Chrobak and Kant) reveals that further degree constraints may be imposed on the 3-tree for prescribed vertices (for example, two vertices of a common face may be forced to be leaves of the tree). To strengthen this further, Barnette's theorem can be seen as a side-result of a structure obtained in Hamiltonicity studies from generalizing the theory of Tutte paths and Tutte cycles: Gao and Richter [18] proved that every planar 3-connected graph contains a 2-*walk*, which is a walk that visits every vertex exactly once or twice. By going along such 2-walks and omitting the last edge whenever a vertex is revisited, these 2-walks imply the existence of 3-trees. Here, planar 3-connected graphs may again be replaced with circuit graphs, and all results have been successfully lifted to higher surfaces. Even more, the surfaces on which every embedded 3-connected graph contains a 2-walk have been classified [7].

Perhaps one of the most severe strengthenings of the 3-tree in question is a long-standing and to the best of our knowledge still open conjecture made by Grünbaum in 1970. Since the planar dual $G^* = (V^*, E^*)$ of every (simple) planar 3-connected graph G is again planar and 3-connected, G^* contains a 3-tree as well. By the well-known cut-cycle duality, any spanning tree T of G implies that also $\neg T^* := (V^*, (E(G) - E(T))^*)$ is a spanning tree of G^*; we call $\neg T^*$ the *co-tree* of T. Taking the best of these two worlds, Grünbaum made the following conjecture.

▶ **Conjecture** (Grünbaum [19, p. 1148], 1970). *Every planar 3-connected graph G contains a 3-tree T whose co-tree $\neg T^*$ is also a 3-tree.*

While Grünbaum's conjecture is to the best of our knowledge still unsolved, progress has been made by Biedl [5], who proved the existence of a 5-tree, whose co-tree is a 5-tree. We prove the existence of a 4-tree, whose co-tree is a 4-tree. Our methods exploit insights into the structure of Schnyder woods. We discuss Schnyder woods, their lattice structure and ordered path partitions in Section 2, our main result in Section 3 and computational aspects of this main result in Section 5.

2 Schnyder Woods and Ordered Path Partitions

We only consider simple undirected graphs. A graph is *plane* if it is planar and embedded into the Euclidean plane without intersecting edges. The *neighborhood of a vertex set A* is the union of the neighborhoods of vertices in A. Although parts of this paper use orientation on edges, we will always let vw denote the undirected edge $\{v, w\}$.

2.1 Schnyder Woods

Let $\sigma := \{r_1, r_2, r_3\}$ be a set of three vertices of the outer face boundary of a plane graph G in clockwise order (but not necessarily consecutive). We call r_1, r_2 and r_3 *roots*. The *suspension G^σ* of G is the graph obtained from G by adding at each root of σ a half-edge pointing into the outer face. With a little abuse of notation, we define a *half-edge* as an arc that has a startvertex but no endvertex. A plane graph G is *σ-internally 3-connected* if the graph obtained from the suspension G^σ of G by making the three half-edges incident to a common new vertex inside the outer face is 3-connected. Note that the class of σ-internally 3-connected plane graphs properly contains all 3-connected plane graphs.

▶ **Definition 1** (Felsner [11]). *Let $\sigma = \{r_1, r_2, r_3\}$ and G^σ be the suspension of a σ-internally 3-connected plane graph G. A* Schnyder wood *of G^σ is an orientation and coloring of the edges of G^σ (including the half-edges) with the colors 1,2,3 (red, green, blue) such that*

(a) *Every edge e is oriented in one direction (we say e is* unidirected*) or in two opposite directions (we say e is* bidirected*). Every direction of an edge is colored with one of the three colors 1,2,3 (we say an edge is i-*colored *if one of its directions has color i) such that the two colors i and j of every bidirected edge are distinct (we call such an edge i-j-*colored*). Similarly, a unidirected edge whose direction has color i is called i-*colored*. Throughout the paper, we assume modular arithmetic on the colors 1,2,3 in such a way that $i+1$ and $i-1$ for a color i are defined as $(i \mod 3)+1$ and $(i+1 \mod 3)+1$. For a vertex v, a uni- or bidirected edge is* incoming *(i-colored) in v if it has a direction (of color i) that is directed toward v, and* outgoing *(i-colored)* of *v if it has a direction (of color i) that is directed away from v.*

(b) *For every color i, the half-edge at r_i is unidirected, outgoing and i-colored.*

(c) *Every vertex v has exactly one outgoing edge of every color. The outgoing 1-, 2-, 3-colored edges e_1, e_2, e_3 of v occur in clockwise order around v. For every color i, every incoming i-colored edge of v is contained in the clockwise sector around v from e_{i+1} to e_{i-1} (see Figure 1).*

(d) *No inner face boundary contains a directed cycle (disregarding possible opposite edge directions) in one color.*

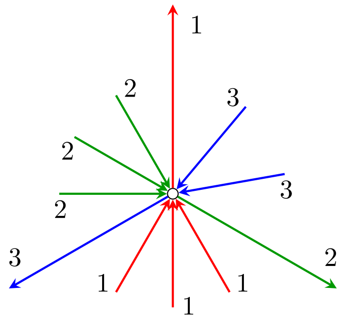


Figure 1 Properties of Schnyder woods. Condition 1c at a vertex.

For a Schnyder wood and color i, let T_i be the directed graph that is induced by the directed edges of color i. The following result justifies the name of Schnyder woods.

▶ **Lemma 2** ([12, 24]). *For every color i of a Schnyder wood of G^σ, T_i is a directed spanning tree of G in which all edges are oriented to the root r_i.*

For a directed graph H, we denote by H^{-1} the graph obtained from H by reversing the direction of all its edges.

▶ **Lemma 3** (Felsner [14]). *For every $i \in \{1, \ldots, 3\}$, $T_i^{-1} \cup T_{i+1}^{-1} \cup T_{i+2}$ is acyclic.*

2.2 Dual Schnyder Woods

Let G be a σ-internally 3-connected plane graph. Any Schnyder wood of G^σ induces a Schnyder wood of a slightly modified planar dual of G^σ in the following way [9, 13] (see [21, p. 30] for an earlier variant of this result given without proof). As common for plane duality, we will use the plane dual operator * to switch between primal and dual objects (also on sets of objects).

Extend the three half-edges of G^σ to non-crossing infinite rays and consider the planar dual of this plane graph. Since the infinite rays partition the outer face f of G into three parts, this dual contains a triangle with vertices b_1, b_2 and b_3 instead of the outer face vertex f^* such that b_i^* is not incident to r_i for every i (see Figure 2). Let the *suspended dual* G^{σ^*} of G^σ be the graph obtained from this dual by adding at each vertex of $\{b_1, b_2, b_3\}$ a half-edge pointing into the outer face.

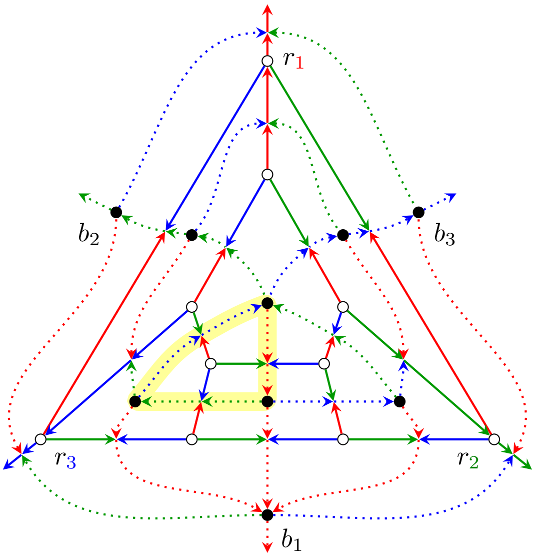


Figure 2 The completion of G obtained by superimposing G^σ and its suspended dual G^{σ^*} (the latter depicted with dotted edges). The primal Schnyder wood is not the minimal element of the lattice of Schnyder woods of G, as this completion contains a clockwise directed cycle (marked in yellow).

Consider the superposition of G^σ and its suspended dual G^{σ^*} such that exactly the primal dual pairs of edges cross (here, for every $1 \le i \le 3$, the half-edge at r_i crosses the dual edge $b_{i-1}b_{i+1}$).

▶ **Definition 4.** *For any Schnyder wood S of G^σ, define the orientation and coloring S^* of the suspended dual G^{σ^*} as follows (see Figure 2):*

(a) *For every unidirected $(i-1)$-colored edge or half-edge e of G^σ, color e^* with the two colors i and $i+1$ such that e points to the right of the i-colored direction.*

(b) *Vice versa, for every i-$(i+1)$-colored edge e of G^σ, $(i-1)$-color e^* unidirected such that e^* points to the right of the i-colored direction.*

(c) *For every color i, make the half-edge at b_i unidirected, outgoing and i-colored.*

The following lemma states that S^* is indeed a Schnyder wood of the suspended dual. The vertices b_1, b_2 and b_3 are called the *roots* of S^*.

▶ **Lemma 5** ([20], [13, Prop. 3]). *For every Schnyder wood S of G^σ, S^* is a Schnyder wood of G^{σ^*}.*

Since $S^{*^*} = S$, Lemma 5 gives a bijection between the Schnyder woods of G^σ and the ones of G^{σ^*}. Let the *completion* $\widetilde{G}$ of G be the plane graph obtained from the superposition of G^σ and G^{σ^*} by subdividing each pair of crossing (half-)edges with a new vertex, which we call a *crossing vertex* (see Figure 2). The completion has six half-edges pointing into its outer face.

Any Schyder wood S of G^σ implies the following natural orientation and coloring $\widetilde{G}_S$ of its completion $\widetilde{G}$: For any edge $vw \in E(G^\sigma) \cup E(G^{\sigma^*})$, let z be the crossing vertex of G^σ that subdivides vw and consider the coloring of vw in either S or S^*. If vw is outgoing of v and i-colored, we direct $vz \in E(\widetilde{G})$ toward z and i-color it; analogously, if vw is outgoing of w and j-colored, we direct $wz \in E(\widetilde{G})$ toward z and j-color it. In the case that vw is unidirected, say without loss of generality incoming at v and i-colored, we direct $zv \in E(\widetilde{G})$ toward v and i-color it. The three half-edges of G^{σ^*} inherit the orientation and coloring of S^* for $\widetilde{G}_S$. By Definition 4, the construction of $\widetilde{G}_S$ implies immediately the following corollary.

▶ **Corollary 6.** *Every crossing vertex of $\widetilde{G}_S$ has one outgoing edge and three incoming edges and the latter are colored 1, 2 and 3 in counterclockwise direction.*

Using results on orientations with prescribed outdegrees on the respective completions, Felsner and Mendez [8, 12] showed that the set of Schnyder woods of a planar suspension G^σ forms a distributive lattice. The order relation of this lattice relates a Schnyder wood of G^σ to a second Schnyder wood if the former can be obtained from the latter by reversing the orientation of a directed clockwise cycle in the completion. This gives the following lemma, of which the computational part is due to Fusy [15].

▶ **Lemma 7** ([8, 12, 15]). *For the minimal element S of the lattice of all Schnyder woods of G^σ, $\widetilde{G}_S$ contains no clockwise directed cycle. Also, S and $\widetilde{G}_S$ can be computed in linear time.*

We call the minimal element of the lattice of all Schnyder woods of G^σ the *minimal Schnyder wood* of G^σ.

2.3 Ordered Path Partitions

▶ **Definition 8.** *For any $j \in \{1, 2, 3\}$ and any $\{r_1, r_2, r_3\}$-internally 3-connected plane graph G, an* ordered path partition *$\mathcal{P} = (P_0, \dots, P_s)$ of G* with base-pair *(r_j, r_{j+1}) is an ordered partition of $V(G)$ into the vertex sets of induced paths such that the following holds for every $i \in \{0, \dots, s-1\}$, where $V_i := \bigcup_{q=0}^{i} P_q$ and the* contour *C_i is the clockwise walk from r_{j+1} to r_j on the outer face of $G[V_i]$.*

(a) *P_0 is the vertex set of the clockwise path from r_j to r_{j+1} on the outer face boundary of G, and $P_s = \{r_{j+2}\}$.*
(b) *Every vertex in P_i has a neighbor in $V(G) \setminus V_i$.*
(c) *C_i is a path.*
(d) *Every vertex in C_i has at most one neighbor in P_{i+1}.*

For the ease of notation we often refer to vertex sets of paths as paths.

▶ Remark 9. Our definition of an ordered path partition $\mathcal{P} = (P_0, \dots, P_s)$ is essentially the definition of Badent et al. [2], in which the vertex sets P_i have to induce paths (this is not explicitly stated in [2], but used in the proof of their Theorem 5). Because a part of the proof of Theorem 5 in [2] (correspondence of ordered path partitions and Schnyder woods) was incomplete, Alam et al. [1, Lemma 1] corrected the result, but unfortunately outsourced the corrected proof into the extended abstract [1, arXiv version, Section 2.2] only. In this correction however, Alam et al. [1] give an incomplete definition[3] of ordered path partitions that misses Condition b. This incompleteness does however not affect the proof of their Lemma 4 [1, arXiv version], as this only gives a correction of [2, Theorem 5] regarding the order of the paths. In this paper, we only use Lemma 4 of [1, arXiv version] which is identical to [1, Lemma 1].

[3] Confirmed by personal communication with the authors of [1].

By Definition 8a and 8b, G contains for every i and every vertex $v \in P_i$ a path from v to r_{j+2} that intersects V_i only in v. Since G is plane, we conclude the following.

▶ **Lemma 10.** *Every path P_i of an ordered path partition is embedded into the outer face of $G[V_{i-1}]$ for every $1 \leq i \leq s$.*

Compatible Ordered Path Partitions

We describe a connection between Schnyder woods and ordered path partitions that was first given by Badent et al. [2, Theorem 5] and then revisited by Alam et al. [1, Lemma 1].

▶ **Definition 11.** *Let $j \in \{1, 2, 3\}$ and S be any Schnyder wood of the suspension G^σ of G. As proven in [1, arXiv version, Section 2.2], the vertex sets of the inclusion-wise maximal j-$(j+1)$-colored paths of S then form an ordered path partition of G with base pair (r_j, r_{j+1}), whose order is a linear extension of the partial order given by reachability in the acyclic graph $T_j^{-1} \cup T_{j+1}^{-1} \cup T_{j+2}$; we call this special ordered path partition* compatible *with S and denote it by $\mathcal{P}^{j,j+1}$.*

For example, for the Schnyder wood given in Figure 2, $\mathcal{P}^{2,3}$ consists of the vertex sets of six maximal 2-3-colored paths, of which four are single vertices. We denote each path $P_i \in \mathcal{P}^{j,j+1}$ by $P_i := \{v_1^i, \dots, v_k^i\}$ such that $v_1^i v_2^i$ is outgoing j-colored at v_1^i and, for every $l \in \{1, \dots, k-1\}$, $v_l^i v_{l+1}^i$ is a j-$(j+1)$-colored edge.

Let C_i be as in Definition 8. By Definition 8c and Lemma 10, every path $P_i = \{v_1^i, \dots, v_k^i\}$ of an ordered path partition satisfying $i \in \{1, \dots, s\}$ has a neighbor $v_0^i \in C_{i-1}$ that is closest to r_{j+1} and a different neighbor $v_{k+1}^i \in C_{i-1}$ that is closest to r_j (see Figure 3). We call v_0^i the *left neighbor* of P_i, v_{k+1}^i the *right neighbor* of P_i and $P_i^e := \{v_0^i\} \cup P_i \cup \{v_{k+1}^i\}$ the *extension* of P_i; we omit superscripts if these are clear from the context. For $0 < i \leq s$, let the path P_i *cover* an edge e or a vertex x if e or x is contained in C_{i-1}, but not in C_i, respectively.

▶ **Lemma 12.** *Every path $P_i \neq P_0$ of a compatible ordered path partition $\mathcal{P}^{j,j+1}$ satisfies the following (see Figure 3):*

(a) *Every neighbor of P_i that is in V_{i-1} is contained in the path of C_{i-1} between v_0^i and v_{k+1}^i.*

(b) *$v_0^i v_1^i$ and $v_k^i v_{k+1}^i$ are edges of $G[V_i]$.*

(c) *$v_0^i v_1^i$ is $(j+1)$-colored outgoing at v_1^i and $v_k^i v_{k+1}^i$ is j-colored outgoing at v_k^i.*

(d) *Every edge $v_l^i x$ incident to P_i and V_{i-1} except for $v_0^i v_1^i$ and $v_k^i v_{k+1}^i$ is unidirected toward P_i, $(j+2)$-colored and satisfies $x \notin \{v_0^i, v_{k+1}^i\}$.*

Proof. The statement a follows directly from Lemma 10 and the definition of left and right neighbor of P_i.

Now, we prove statements b and c. According to Definition 11, the order of $\mathcal{P}^{j,j+1}$ on the vertex sets of paths is a linear extension of the partial order given by reachability in the acyclic graph $T_j^{-1} \cup T_{j+1}^{-1} \cup T_{j+2}$. This allows us to characterize the color of the edges that join P_i with vertices of V_{i-1} and $V - V_i$, respectively. Edges that join P_i with vertices of V_{i-1} are incoming $(j+2)$-colored, unidirected outgoing j-colored or unidirected outgoing $(j+1)$-colored at a vertex of P_i. Edges that join P_i with vertices of $V - V_i$ are outgoing $(j+2)$-colored, unidirected incoming j-colored or unidirected incoming $(j+1)$-colored at a vertex of P_i.

Recall that all edges of $G[P_i]$ are j-$(j+1)$-colored. Let wv_1^i be the outgoing $(j+1)$-colored edge at v_1^i and $v_k^i u$ be the outgoing j-colored edge at v_k^i. If $k > 1$, $v_1^i v_2^i$ is outgoing j-colored by definition. Thus, as $G[P_i]$ is induced, $w \notin P_i$. If $k = 1$, P_i consists of only one vertex and hence $w \notin P_i$. Thus, as $G[P_i]$ is a maximal j-$(j+1)$-colored path, wv_1^i is either unidirected $(j+1)$-colored or $(j+1)$-$(j+2)$-colored. As observed above, this implies $w \in V_{i-1}$ and by a $w \in C_{i-1}$. Similarly, we obtain $u \in C_{i-1}$.

Assume to the contrary that u is closer to r_{j+1} on C_{i-1} than w is. By definition of P_i, for every vertex of P_i, the outgoing j-colored edge is directed toward u and the outgoing $(j+1)$-colored edge points toward w on $G[P_i] \cup \{wv_1^i, v_k^i u\}$. By Definition 1c, the outgoing $(j+2)$-colored edge e of a vertex of P_i occurs in the counterclockwise sector from the outgoing j-colored to the outgoing $(j+1)$-colored edge excluding both. As we assumed that u is closer to r_{j+1} on C_{i-1} than w is, this sector is in the interior of the region bounded by $G[P_i] \cup \{wv_1^i, v_k^i u\}$ and the path from u to w on C_{i-1}. Hence, by planarity, e joins P_i with a vertex of $C_{i-1} \subseteq V_{i-1}$, contradicting our above characterization of edges that join P_i with vertices of V_{i-1}. Thus, w is closer to r_{j+1} on C_{i-1} than u is or we have $w = u$. If $u = w$, then Lemma 3 is violated by the cycle formed by $P_i \cup u$ in $T_j \cup T_{j+1}^{-1} \cup T_{j+2}^{-1}$, which is a contradiction. Thus, w is closer to r_{j+1} on C_{i-1} than u.

Since P_i is a maximal j-$(j+1)$-colored path, the outgoing j-colored and $(j+1)$-colored edges at every of its vertices are either in P_i or in $\{wv_1^i, v_k^i u\}$. Hence, by our above characterization, the edges that join P_i with vertices of $C_{i-1} \subseteq V_{i-1}$ are exactly $v_k^i u$, $v_1^i w$ and the unidirected incoming $(j+2)$-colored edges at vertices of P_i. Let vx be such an unidirected incoming $(j+2)$-colored edge with $v \in P_i$. By Definition 1c, vx occurs in the clockwise sector from the outgoing j-colored edge to the outgoing $(j+1)$-colored edge around v excluding both. By planarity and the fact that w is closer to r_{j+1} on C_{i-1} than u, x is contained in the path of C_{i-1} between w and u. By definition of the left and right neighbor v_0^i and v_{k+1}^i of P_i, we thus have $v_0^i = w$ and $v_{k+1}^i = u$, which proves b and c.

For d, let $v_l^i x \notin \{v_k^i v_{k+1}^i, v_1^i v_0^i\}$ be an edge that joins P_i with a vertex x of V_{i-1}. By a, $x \in C_{i-1}$. In the last paragraph, we observed that $v_l^i x$ is incoming $(j+2)$-colored at a vertex of P_i. We showed also that the outgoing j-colored and the outgoing $(j+1)$-edge of any vertex in P_i is either in P_i or $v_1^i v_0^i$ or $v_k^i v_{k+1}^i$. Thus, we obtain that $v_l^i x$ is unidirected incoming $(j+2)$-colored at a vertex of P_i. Assume, for the sake of contradiction, that $x = v_0^i$. Then the path from v_l^i to v_1^i on P_i, $v_0^i v_1^i$ and $v_l^i v_0^i$ form an oriented cycle in $T_j \cup T_{j+1}^{-1} \cup T_{j+2}^{-1}$, which contradicts Lemma 3. A similar argument shows $x \neq v_{k+1}^i$. ◀

3 Spanning Trees with Maximum Degree at Most 4

In this section, we prove our main result. The following new lemma on the structure of minimal Schnyder woods and their compatible ordered path partitions is crucial for this proof. For $0 < i \leq s$, let the path P_i *cover* an edge e or a vertex x if e or x is contained in C_{i-1}, but not in C_i, respectively.

▶ **Lemma 13.** *Let G be a σ-internally 3-connected plane graph, S be the minimal Schnyder wood of G^σ and $\mathcal{P}^{2,3} = (P_0, \ldots, P_s)$ be the ordered path partition that is compatible with S. Let $P_i := \{v_1, \ldots, v_k\} \neq P_0$ be a path of $\mathcal{P}^{2,3}$ and v_0 and v_{k+1} be its left and right neighbor. Then every edge $v_l w \notin \{v_0 v_1, v_k v_{k+1}\}$ with $v_l \in P_i$ and $w \in V_{i-1}$ is unidirected, 1-colored and incoming at v_k and $w \notin \{v_0, v_{k+1}\}$.*

Proof. Consider any edge $v_l w \notin \{v_0 v_1, v_k v_{k+1}\}$ that is incident to $v_l \in P_i$ and $w \in V_{i-1}$ (see Figure 3). By Lemma 12a, w is either v_0, v_{k+1} or a vertex that is covered by P_i. As $v_l w \notin \{v_0 v_1, v_k v_{k+1}\}$, $v_l w$ must be 1-colored incoming at v_l such that $w \notin \{v_0, v_{k+1}\}$ by Lemma 12d. It thus remains to show that $l = k$.

Assume to the contrary that $l \neq k$. Observe that, by Definition 1c, all edges in the clockwise sector from $v_l v_{l+1}$ to $v_l v_{l-1}$ are incoming 1-colored. Choose w such that $v_l w$ is the clockwise first incoming 1-colored edge at v_l (see Figure 3). By Corollary 6, the dual edge of $v_l v_{l+1}$ is unidirected 1-colored in the completion $\widetilde{G}_S$ of G and the dual edge of $v_l w$ is 2-3-colored. Hence, $\widetilde{G}_S$ contains the clockwise cycle shown in Figure 3, which contradicts the assumption that S is the minimal Schnyder wood. ◀

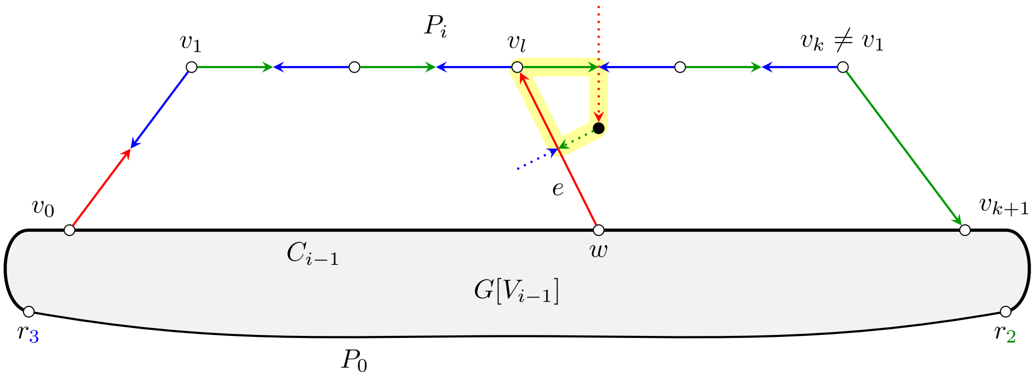


Figure 3 The clockwise cycle of $\widetilde{G}_S$ of the proof of Lemma 13, depicted in yellow.

For a spanning subgraph T of a plane graph G, let the *co-graph* $\neg T^*$ be the spanning subgraph $(V^*, (E(G) - E(T))^*)$ of G^*. As stated in the introduction, $\neg T^*$ is a spanning tree if T is one and in that case called a co-tree.

▶ **Theorem 14.** *Every $\{r_1, r_2, r_3\}$-internally 3-connected plane graph G contains a 4-tree T whose co-tree $\neg T^*$ is a 4-tree.*

Proof. We first sketch the general idea of the proof: First, we identify a spanning candidate graph $H \subseteq G$ such that $\neg H^*$ is a subgraph of G^* that has the same structural properties as H. We then define a subset D of the edges of H such that $H - D$ is acyclic and $\neg H^* + D^*$ has maximum degree 4. We use the same arguments to define a similar subset D' for $\neg H^*$. In the end, we need to show that D'^* and D^* do not create new cycles in $\neg H^*$ and H, respectively. That way we obtain that the co-graph of $H - D + D'^*$ is $\neg H^* - D' + D^*$, and both graphs are acyclic and of maximum degree 4. Since a spanning subgraph G' of G is connected if and only if $G - E(G')$ does not contain any edge cut of G, the cut-cycle duality [10, Prop. 4.6.1] proves that those two graphs are both connected, which gives the claim.

Let S be the minimal Schnyder wood of G^σ. By Lemma 7, the completion $\widetilde{G}_S$ of G contains no clockwise directed cycle. Since $\widetilde{G}_S$ contains the completion of the suspended dual G^{σ^*} except for its three outer vertices (which do not affect clockwise cycles), S^* is a minimal Schnyder wood of G^{σ^*}.

Let H be the spanning subgraph of G whose edge set consists of the bidirected edges of S. Recall that an edge $e \in E(G)$ is not in H if and only if e^* is in $\neg H^*$. By Definition 4, $\neg H^*$ contains therefore exactly the bidirected edges of S^*, except for the three bidirected edges on the outer face boundary of G^{σ^*}, as these are not dual edges of G (in fact, these three edges appear only in the suspended dual G^{σ^*} and were necessary to define dual Schnyder woods).

Since every vertex is incident to at most three bidirected edges by Definition 1c for S and as well for S^*, both H and $\neg H^*$ have maximum degree at most three. However, H and $\neg H^*$ may neither be connected nor acyclic. In fact, H contains always the outer face boundary of G as a cycle, as all edges are bidirected by the definition of the first paths of the compatible ordered path partitions $\mathcal{P}^{1,2}$, $\mathcal{P}^{2,3}$ and $\mathcal{P}^{3,1}$.

We will therefore iteratively identify edges of cycles of H such that $\neg H^*$ still has maximum degree at most four when those cycles are deleted in H. In order to do this, we iteratively define edge sets D and D' that are deleted from H and $\neg H^*$, starting with $D := D' := \emptyset$.

Let C be a cycle of H and let $(P_0, \dots, P_s)$ be the paths of the compatible ordered path partition $\mathcal{P}^{2,3}$ of S. Let P be the path of maximal length in C such that $P \subseteq P_M$ with $M := \max\{i \mid P_i \cap V(C) \neq \emptyset\}$; we call P the *index maximal subpath* of C, as it is the fraction of C highest up in the order of $\mathcal{P}^{2,3}$. Since C has only bidirected edges, the statement of Lemma 13 about e being unidirected implies that $P = P_M$ and that C contains the extension of P; in particular, $P \in \mathcal{P}^{2,3}$.

Denote by $\mathcal{P}_{max}$ the set of index maximal subpaths of all cycles of H. For a path $P \in \mathcal{P}_{max} \setminus \{P_s\}$, let P_L with $L := \min\{i \mid P_i$ covers an edge of the extension of $P\}$ be the *minimal-covering path* of P (recall that this extension is part of the cycle and the minimal-covering path exists, as P_s is excluded). Denote by $\mathcal{P}_{cover}$ the set of the minimal-covering paths of all index maximal subpaths in $\mathcal{P}_{max} \setminus \{P_s\}$. In particular, $P_s = r_1$ is the index maximal subpath of the outer face boundary of G, which is a bidirected cycle, as shown before. Since no edge of the extension of P_s is covered by another path of $\mathcal{P}^{2,3}$, we add the outgoing 2-colored edge of r_1 to D in order to destroy the outer face cycle.

Next, we process the paths of $\mathcal{P}_{cover}$ in reverse order of $\mathcal{P}^{2,3}$, i.e., from highest to lowest index. Let $P_c = \{v_1, \dots, v_k\} \in \mathcal{P}_{cover}$ for some $c \in \{1, \dots, s\}$ be the path under consideration. Let $P'_1, \dots, P'_l$ be the index maximal paths for which P_c is the minimal-covering path, ordered clockwise around the outer face of $G[V_{c-1}]$ (see Figure 4); note that there may also be other paths covered by P_c that are not index maximal. Let $f_1, \dots, f_a$ be the faces incident to v_k in counterclockwise order from the outgoing 3-colored edge to the outgoing 2-colored edge; we say that $f_1, \dots, f_a$ are *below* P_c. For every path of $\{P'_1, \dots, P'_l\}$, we will add an edge to D that is on the extension of that path. Thus, after having processed every path in $\mathcal{P}_{cover}$ in this way, a cycle in H does not exist in $H - D$ anymore.

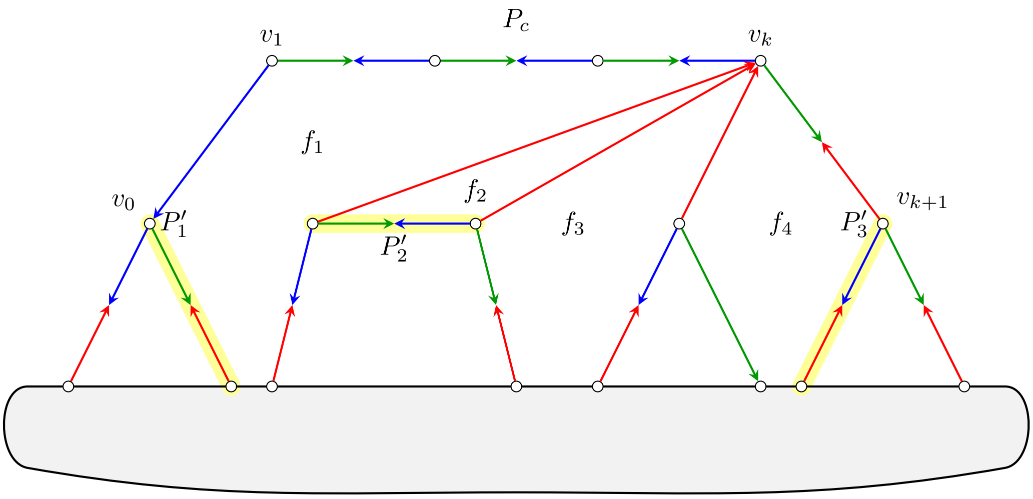


Figure 4 Illustration for some of the definitions used in the proof of Theorem 14. If Case 1 applies to P_c, we add the edges marked in yellow to D.

Consider the case that $v_{k+1} = w_1$ for a path $P'_l = \{w_1, \dots, w_t\}$. Assume for the sake of contradiction that then $v_k v_{k+1}$ is not 1-2-colored. Since P'_l is an index maximal subpath, $w_0 w_1$ is 1-3-colored. By Lemma 12c, then $v_k v_{k+1}$ is unidirected 2-colored. By Corollary 6,

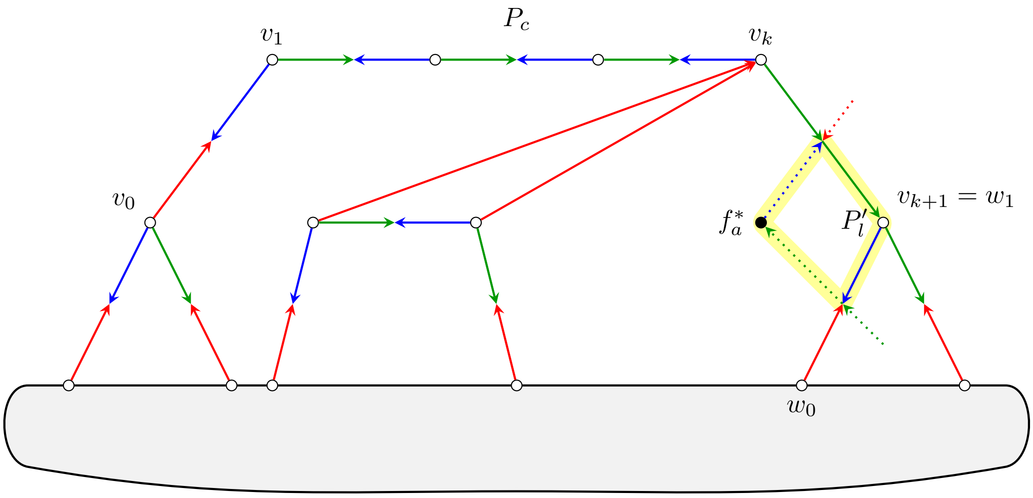


Figure 5 If $v_k v_{k+1}$ is unidirected 2-colored, then $\widetilde{G}_S$ contains the clockwise cycle depicted in yellow.

this implies that $(v_k v_{k+1})^*$ is 1-3-colored. Hence, $\widetilde{G}_S$ contains the clockwise cycle in Figure 5, which contradicts the assumption that S is the minimal Schnyder wood. We conclude that $v_k v_{k+1}$ is 1-2-colored in that case.

We will now select one edge from each of the extensions of the paths $P_1', \ldots, P_l'$ and add it to D. We generally aim for selecting those edges that have smallest possible impact on the maximum degree of the dual graph: we prefer always edges of the paths $P_1', \ldots, P_l'$ that are covered by P_c. For example, for P_2' in Figure 4, adding its edge to D causes a higher degree at the dual vertex f_2^* while connecting the dual to it; this is fine, as f_2 is a triangle by the mandatory outgoing 1-colored edges and thus the degree of f_2^* never exceeds 3 anyway. In detail, we distinguish the following two cases.

Augmentation procedure of D for the path P_c

Case 1: P_c is not an index maximal subpath (see Figure 4).

For every $i \in \{1, \ldots, l\}$, if P_c covers an edge of $G[P_i']$, then we add one such edge to D. If for $P_l' = \{w_1, \ldots, w_t\}$, we have $w_1 = v_{k+1}$ (note that this excludes the previous condition), then we add $w_0 w_1$ to D. For all remaining $i \in \{1, \ldots, l\}$ for which none of the above conditions apply, we set $P_i' = \{u_1, \ldots, u_t\}$ and add the edge $u_t u_{t+1}$ to D.

Case 2: P_c is an index maximal subpath.

Since the minimal-covering path of P_c has higher index than P_c itself, there already is either an edge of $G[P_c]$, $v_0 v_1$ or $v_k v_{k+1}$ in D.

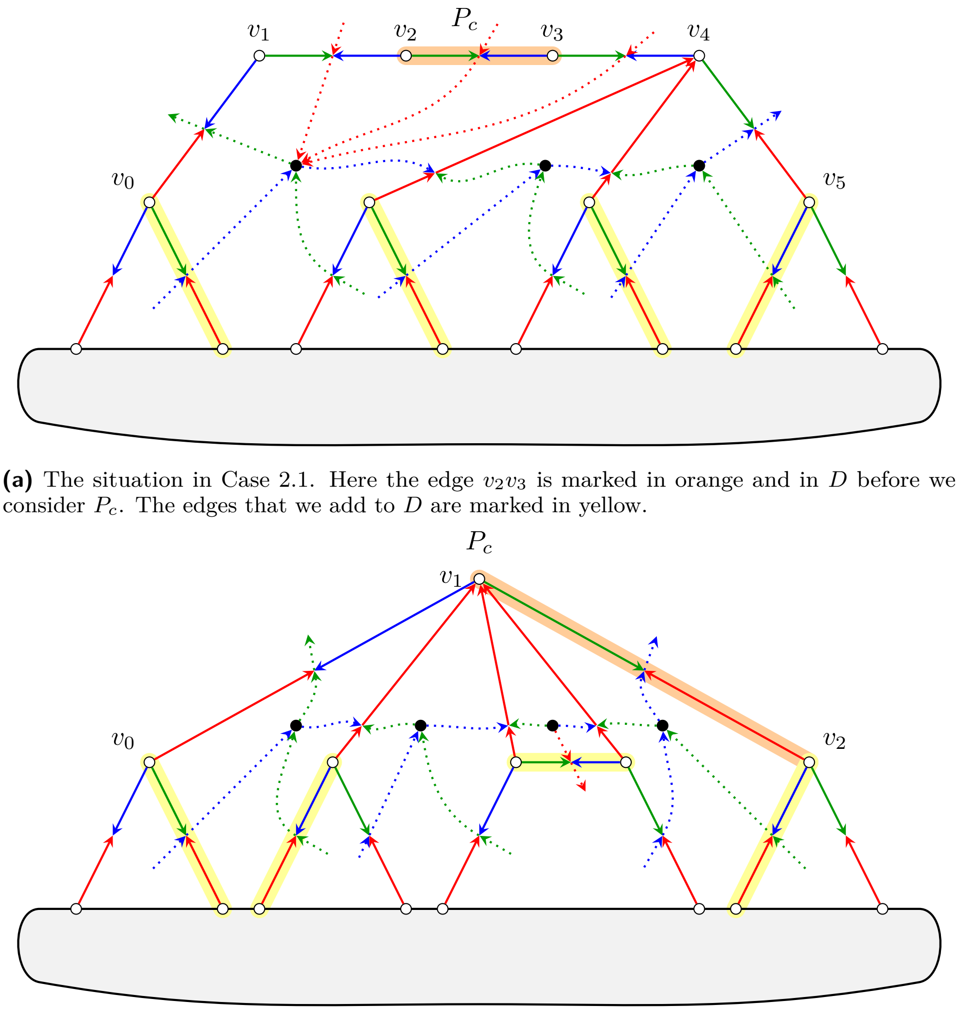
Case 2.1: An edge of $G[P_c]$ or $v_0 v_1$ is in D (see Figure 6a).

We proceed as in Case 1.

Case 2.2: $v_k v_{k+1} \in D$ (see Figure 6b)

For every $i \in \{1, \ldots, l\}$, if P_c covers an edge of $G[P_i']$, then we add one such edge to D. If for $P_1' = \{p_1, \ldots, p_b\}$, we have $p_b = v_0$ (note that this excludes the previous condition), then we add $p_b p_{b+1}$ to D. For all remaining $i \in \{1, \ldots, l\}$ for which none of the above conditions apply, we set $P_i' = \{u_1, \ldots, u_t\}$ and add the edge $u_0 u_1$ to D.

We now need to show that the maximum degree of $\neg H^* + D^*$ does not exceed 4. We prove that, after having processed P_c, no further boundary edge of any $f \in \{f_1, \ldots, f_a\}$ is added to D: Assume to the contrary that there is a face $f \in \{f_1, \ldots, f_a\}$ and an edge e



(a) The situation in Case 2.1. Here the edge v_2v_3 is marked in orange and in D before we consider P_c. The edges that we add to D are marked in yellow.

(b) The situation in Case 2.2. The edge v_1v_2 is marked in orange and in D before we consider P_c. The edges that we then add to D are marked in yellow.

Figure 6 Subcases for which P_c is an index maximal subpath in Theorem 14.

on the boundary of f such that e is not in D after having processed P_c but will be added later. Let $P_i \in \mathcal{P}^{2,3}$ be the path whose extension contains e. Then the minimal-covering path $P_{c'} \in \mathcal{P}^{2,3}$ of P_i needs to have lower index than P_c, i.e., $c' < c$. As e is covered by P_c, it is not covered by the minimal-covering path of P_i. Hence e will not be added to D, which is a contradiction.

First, consider the case $a > 1$, in which there at least two faces below P_c. By Definition 8b, the boundary of every f_j with $j \in \{1, \dots, a\}$ contains at most two edges that are in the union of the extensions of paths in $\{P'_1, \dots, P'_l\}$. For $j \in \{2, \dots, a-1\}$, the augmentation procedure adds at most one of those edges to D, which implies that $\deg_{\neg H^*+D^*}(f^*_j) \leq 4$ for every $j \in \{2, \dots, a-1\}$ (see Figure 6).

Now, consider $j = 1$, i.e., the face f_1 in the case $a > 1$. Let $P'_1 = \{p_1, \dots, p_b\}$. In Case 1 of the augmentation procedure, we add at most one edge of the boundary of f_1 to D, hence $\deg_{\neg H^*+D^*}(f^*_1) \leq 4$. In Case 2, v_0v_1 is 1-3-colored, since P_c is an index maximal subpath (see Figure 6). By Corollary 6, $(v_0v_1)^*$ is unidirected 2-colored and outgoing at f^*_1. This implies $\deg_{\neg H^*}(f_1^*) \leq 2$, as f_1^* is incident to at most two bidirected edges. In Case 2.1, there is an edge of P_c or v_0v_1 in D. And if $p_b = v_0$, the edge p_bp_{b+1} is in D. Those are the only edges of the boundary of f_1 in D in Case 2.1 and hence $\deg_{\neg H^*+D^*}(f_1^*) \leq 4$. In Case 2.2, there is neither an edge of P_c nor v_0v_1 in D. As above, if $p_b = v_0$, the edge p_bp_{b+1} is in D. And if the left neighbor of P_2' is no the boundary of f_1, then also the edge from P_2' to its left neighbor is in D. Thus, also in Case 2.2, the augmentation procedure adds at most two edges of the boundary of f_1 to D and hence $\deg_{\neg H^*+D^*}(f_1^*) \leq 4$.

Now consider $j = a$, i.e., the face f_a in the case $a > 1$. Let $P_l' = \{w_1, \dots, w_t\}$. If v_kv_{k+1} is 1-2-colored, then $(v_kv_{k+1})^*$ is unidirected 3-colored and outgoing at f_a^* by Corollary 6 and hence $\deg_{\neg H^*}(f_a^*) \leq 2$. The augmentation procedure adds at most two edges of the boundary of f_a to D and hence $\deg_{\neg H^*+D^*}(f_a^*) \leq 4$. Assume now that v_kv_{k+1} is unidirected 2-colored. Then P_c is not an index maximal subpath and we are in Case 1. As we observed above, then $w_1 \neq v_{k+1}$. The augmentation procedure adds at most one edge of the boundary of f_a to D and we have $\deg_{\neg H^*+D^*}(f_a^*) \leq 4$.

In the remaining case $a = 1$, there is exactly one face below P_c. If P_c is not an index maximal subpath, we use exactly the same arguments as we used to show that $\deg_{\neg H^*+D^*}(f_a^*) \leq 4$ for $a \neq 1$. If P_c is an index maximal subpath, then, by the same arguments as above, we know that $(v_kv_{k+1})^*$ and $(v_1v_0)^*$ are unidirected and outgoing at f_1^*. This implies $\deg_{\neg H^*}(f_1^*) \leq 1$. There at most three edges of the boundary of f_1 in D. Those potential edges are an edge of the extension of P_c, the outgoing 2-colored edge of v_0 and the outgoing 3-colored edge of v_{k+1}.

In addition, there are faces that are never below a path of $\mathcal{P}_{cover}$. Those faces have at most one edge of their boundary in D. Thus, their dual vertices in $\neg H^* + D^*$ have degree at most 4 (see Figure 6).

The clockwise path from r_2 to r_3 on the outer face boundary is not an index maximal subpath. Hence, the augmentation procedure does not add any edge of the clockwise path from r_2 to r_3 on the outer face boundary to D. However, by our assumption, D includes the outgoing 2-colored edge at r_1, which is the only edge of D that is on the boundary of the outer face of G.

So far we showed that $H - D$ is acyclic and $\neg H^* + D^*$ has maximum degree at most 4. We now apply the same arguments that we used for H to $\neg H^* \cup \{b_1b_2, b_2b_3, b_3b_1\}$ and obtain D'. Hence, we have that $\neg H^* \cup \{b_1b_2, b_2b_3, b_3b_1\} - D'$ is acyclic and $H + D'^* \setminus \{b_1b_2, b_2b_3, b_3b_1\}^*$ has maximum degree at most 4.

The edges b_1b_2, b_2b_3 and b_3b_1 are not in G^* and there is only one edge on the boundary of the outer face of G that is also in D. We may thus ignore b_1b_2, b_2b_3 and b_3b_1 in the following and freely switch from $\neg H^* \cup \{b_1b_2, b_2b_3, b_3b_1\}$ to $\neg H^*$. Hence, we also remove any of the edges b_1b_2, b_2b_3, b_3b_1 from D'.

Then the graphs $H - D + D'^*$ and $\neg H^* - D' + D^*$ have maximum degree at most 4 and by construction $\neg(H - D + D'^*)^* = \neg H^* - D' + D^*$. An edge set $E \subseteq E(G)$ is the edge set of a cycle in G if and only if the edge set E^* is a minimal edge cut in G^* [10, Prop. 4.6.1]. So in order to show that $\neg H^* - D' + D^*$ and $H - D + D'^*$ are both trees it suffices to show that they are both acyclic. We show that $\neg H^* - D' + D^*$ is acyclic. Applying the same arguments then shows that $H - D + D'^*$ is acyclic.

Assume to the contrary that there is a cycle C in $\neg H^* - D' + D^*$. Remember that for each index maximal subpath in $\mathcal{P}_{max}$ we pick exactly one edge of the extension and add it to D. This will finally lead to a contradiction. By construction, every cycle in $\neg H^*$ has at least one edge that is also in D'. Hence, C has at least one edge of D^*. Since every edge of D is in a cycle of H, by [10, Prop. 4.6.1], every edge in D^* joins two vertices of two different connected components of $\neg H^*$.

For a connected component K of $\neg H^*$, let $E_K \subseteq E(G^*)$ be the minimal edge cut separating K and $G^* - K$. Let C_K be the cycle of G with $E(C_K) = E_K^*$ and let $P^{C_K} = P_i \in \mathcal{P}^{2,3}$ be the index maximal subpath of C_K (see Figure 7). Choose K such that K shares a vertex with C and $P^{C_K} = P_i$ has smallest index. Since C is a cycle and intersects at least two connected components of $\neg H^*$, there are two edges $e, e' \in E_K$ that are also in C. Observe that these edges need to be in D^*.

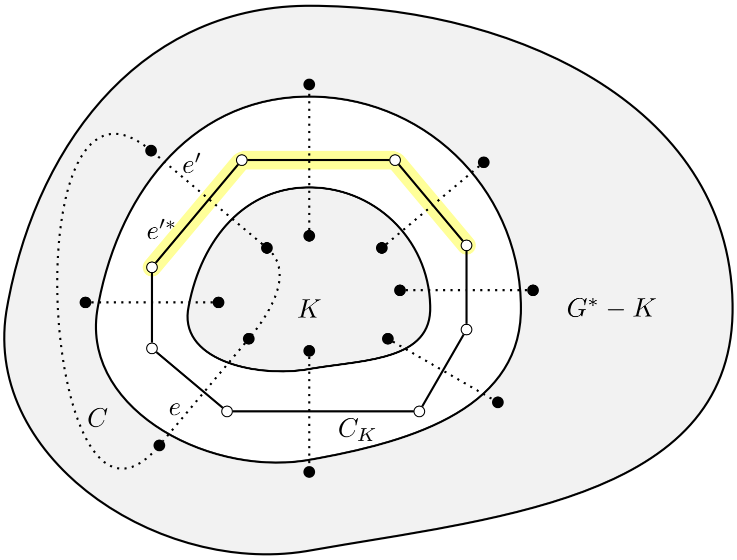


Figure 7 Illustration for the proof of Theorem 14. The extension of the path P^{C_K} is highlighted in yellow.

Then either e^* or e'^* is not in the extension of the index maximal subpath P^{C_K}. Assume w.l.o.g. that e^* is not in the extension of P^{C_K}. Let $P' = P_j \in \mathcal{P}^{2,3}$ for some $j \in \{1, \dots, s\}$ be the path such that e^* is in the extension of P'. Since P^{C_K} is the index maximal subpath of C_K, we have $j < i$. So there exists a connected component K' of $\neg H^*$ such that K' and C have a vertex in common and P' is the index maximal subpath of the cycle $C_{K'}$ with $(E(C_{K'}))^*$ being the minimal cut separating K' and $G^* - K'$. This contradicts the definition of K. So $\neg H^* - D' + D^*$ and $H - D + D'^*$ are our desired trees. ◀

▶ **Corollary 15.** *Every 3-connected planar graph G contains a 4-tree T whose co-tree $\neg T^*$ is also a 4-tree.*

▶ **Corollary 16.** *The root r_1 is a leaf in $H - D + D'^*$ and all edges on the outer face of G except for the outgoing 2-colored edge at r_1 are in $H - D + D'^*$. We have $\deg_{H-D+D'^*}(r_3) = 2$ and $\deg_{H-D+D'^*}(r_2) \leq 3$. Also, the dual vertex of the outer face of G is a leaf in $\neg H^* - D' + D^*$.*

Proof. The proof of Theorem 14 yields that all edges on the outer face of G except for the outgoing 2-colored edge at r_1 are in $H - D + D'^*$. In $G^{\sigma *}$, the path $P_1 \in \mathcal{P}^{2,3}$ is given by the duals of the unidirected incoming 1-colored edges at r_1 (see Figure 2). Since the outgoing 2-colored and the outgoing 3-colored edge at r_1 are bidirected, P_1 is not an index maximal subpath and hence none of the duals of the unidirected incoming 1-colored edges at r_1 is added to D'. Thus, r_1 is a leaf in $H - D + D'^*$.

The dual edges of the incoming unidirected edges at r_2 and r_3 are all covered by the last singleton b_1 of $\mathcal{P}^{2,3}$ of $\neg H^* \cup \{b_1b_2, b_2b_3, b_3b_1\}$ (see Figure 2). Let e_2 be the dual of the clockwise first unidirected 2-colored incoming edge at r_2 and e_3 be the dual of the counterclockwise first unidirected 3-colored incoming edge at r_3. Let I_i be the set of the duals of the unidirected i-colored incoming edges at r_i, $i = 2, 3$. For $e \in I_i$, $i = 2, 3$ let $P_e \in \mathcal{P}^{2,3}$ be the path such that e belongs to the extension of P_e. Observe that, for all edges $e \in (I_2 \setminus \{e_2\}) \cup (I_3 \setminus \{e_3\})$, b_1 is not the minimal-covering path of P_e. Hence, those edges are not added to D'. On the other hand b_1 might be the minimal-covering path of P_{e_2} and/or P_{e_3}. Since we added b_1b_2 to D', we do not add e_3 to D' but might do so for e_2 (compare Case 2.2 in the proof of Theorem 14). Hence, $\deg_{H-D+D'^*}(r_3) = 2$ and $\deg_{H-D+D'^*}(r_2) \leq 3$.

Since the outgoing 2-colored edge at r_1 is the only edge on the boundary of the outer face f that is not in $H - D + D'^*$, we know that the vertex f^* is a leaf in $\neg H^* - D' + D^*$. ◀

4 Relaxing Connectivity Assumptions

In this section, we relax the connectivity condition. A common relaxation of σ-internal 3-connectedness is internal 3-connectedness. A plane graph G is *internally 3-connected* if adding a vertex, the *apex vertex*, in the outer face and connecting this new vertex with all the vertices on the outer face of G results in a 3-connected graph. Observe that every σ-internally 3-connected graph is also internally 3-connected.

▶ Remark 17. The statement of Theorem 14 does not hold for internally 3-connected graphs. There exist internally 3-connected plane graphs G_k on $2k$ vertices such that every spanning tree of the dual graph has maximum degree at least $\lceil k/2 \rceil$.

Proof. In order to define G_k, fix an embedding of the cycle C_k on k vertices. Let $w_0, \ldots, w_{k-1}$ be the vertices of this cycle in clockwise order. For every $i = 0, \ldots, k-1$, add a vertex p_i in the outer face and add edges p_iw_i and p_iw_{i+1} (indices taken modulo k) such that the resulting graph G_k is still plane (see Figure 8). Clearly, G_k is internally 3-connected. The dual of G_k contains parallel edges. Its underlying graph, in which all those vertex pairs joined by parallel edges are only joined by one edge, is the complete bipartite graph $K_{2,k}$. By pigeonhole principle, every spanning tree of $K_{2,k}$ has maximum degree at least $\lceil k/2 \rceil$. ◀

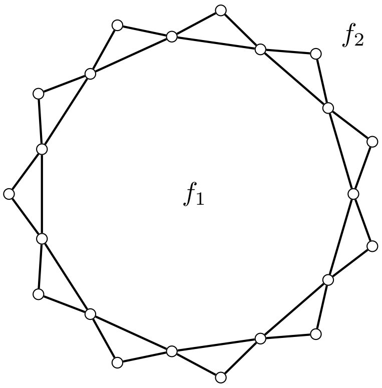


Figure 8 The graph G_{11} of Remark 17. In every spanning tree of the dual graph, f_1^* or f_2^* has degree at least 6.

However, we can apply Theorem 14 to $G + x$ for an internally 3-connected graph G with an apex vertex x. Then, we obtain after small modifications a 4-tree of G and a tree of G^* such that all vertices except for the dual of the outer face have degree at most 4. This motivates the notion of *k-internally 3-connected* graphs. G is *k-internally 3-connected* if there are k vertices $w_1, \dots, w_k$ on the outer face of G such that adding an apex vertex x in the outer face and the edges xw_i for all $i \in \{1, \dots, k\}$ yields a 3-connected graph. Observe that every σ-internally 3-connected graph is k-internally 3-connected for $k \geq 3$ and every k-internally 3-connected graph is also internally 3-connected.

▶ **Lemma 18.** *For every k-internally 3-connected plane graph G there exists a 4-tree such that all vertices of its co-tree except for the dual of the outer face have degree at most 4. The dual of the outer face has degree at most $2k - 2$.*

Proof. Let G^x be the plane graph obtained by adding and connecting the apex vertex as described in the statement. Define $r_1 := x$ and r_2 and r_3 to be its clockwise and counterclockwise neighbor on the outer face of G^x, respectively. Let $w_1, \dots, w_k$ be ordered clockwise around the outer face of G such that $w_1 = r_3$, $w_k = r_2$ and $w_i x \in E(G^x)$ for all $i \in \{1, \dots, k\}$ (Figure 9). We now apply Theorem 14 to G^x with this choice of roots. We obtain a 4-tree T of G^x such that $\neg T^*$ is a 4-tree of G^{x*}. Observe that by Corollary 16 all edges on the outer face of G^x except for $r_1 r_2$ are in T, $\deg_T(r_1) = 1$ and $\deg_T(r_3) = 2$ (see Figure 9). Thus, we have that $w_1 x \in E(T)$ and $w_i x \notin E(T)$ for all $i \in \{2, \dots, k\}$. Hence, $T - w_1 x$ is a 4-tree of G. We consider the dual graph. As $T - w_1 x$ is a 4-tree of G, its co-tree is also a spanning tree of G^*. As $\neg T^*$ is a 4-tree of G^{x*}, we obtain that in the co-tree of $T - w_1 x$ every vertex except for the dual of the outer face has degree at most 4.

We consider the outer face. Take a Schnyder wood as in the proof of Theorem 14. Let d_i be the dual vertex of the face incident to $w_i x$ and $w_{i+1} x$ for $i \in \{1, \dots, k-1\}$ in G^x. In $\neg T^*$, those vertices have degree at most 4. Consider d_1. The dual edge $(r_1 r_3)^*$ is outgoing at d_1. The edge e preceding $r_1 r_3$ on the face d_1^* in clockwise order is unidirected 3-colored and incoming at r_3. Thus, there is no index maximal subpath that contains e. And hence, in the algorithm of the proof of Theorem 14, we add at most one edge to D that is incident to d_1. Furthermore, $(xw_2)^*$ is incident to d_1 and in $E(\neg T^*)$. Therefore, there are at most two edges on the clockwise path from r_3 to w_2 on the outer face of G that are not in T.

As $(w_i x)^*$ and $(w_{i+1} x)^*$ are incident to d_i and $(w_i x)^*, (w_{i+1} x)^* \in E(\neg T^*)$ for all $i \in \{2, \dots, k-1\}$, we obtain, that there are at most two edges on the clockwise path from w_i to w_{i+1} on the outer face of G that are not in T. And hence, the dual vertex of the outer face of G has degree at most $2k - 2$ in the co-tree of $T - w_1 x$. ◀

5 Computational Aspects

Let G^σ be the suspension of a σ-internally 3-connected plane graph and let S be the minimal Schnyder wood of G^σ. Badent et al. showed that an ordered path partition $\mathcal{P}^{2,3}$ that is compatible to S can be computed in time $O(n)$ [2, Theorem 7]. This $\mathcal{P}^{2,3}$ can also be used to compute S itself in the same time [2, Theorem 5], which in turn allows to compute the dual S^* and thus also the candidate graphs H and $\neg H^*$in linear time.

For $i := 1, \dots, s$, we detect whether $H \cap G[V_i]$ has a cycle that contains the extension of P_i by maintaining the connected components of the previous graph $H \cap G[V_{i-1}]$ and querying whether the left and right neighbor of P_i are in the same connected component of $H \cap G[V_{i-1}]$. This can be done in amortized constant time per step using the special union-find data structure in [16], since the structure of possible union operations is a tree. This gives the set $\mathcal{P}_{max}$ of all index maximal subpaths in $\mathcal{P}^{2,3}$ and their minimial-covering paths.

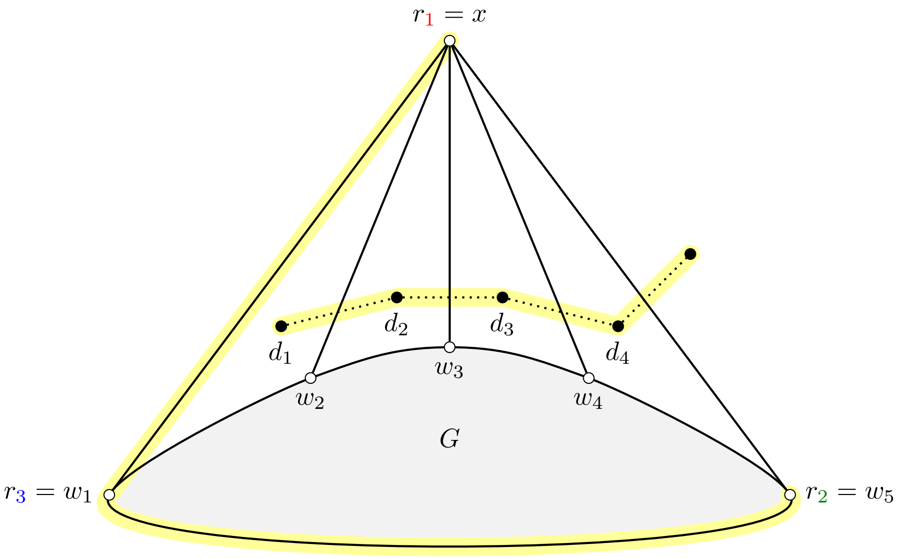


Figure 9 Situation as in Lemma 18. A 5-internally 3-connected graph G with its apex vertex x. Some edges of the 4-tree T of G^x and its co-tree are highlighted in yellow.

Since the case distinction and every step of the augmentation procedure for every minimal-covering path P_c can be computed in constant time per index-maximal subpath, we obtain an algorithm with running time $O(n)$ to compute a 4-tree of G whose co-tree is also a 4-tree.

6 Conclusion

We used Schnyder woods in order to prove that every (σ-)internally 3-connected graph has a 4-tree such that its co-tree is also a 4-tree. Also, we showed that there is a linear time algorithm computing such a tree. If we further relax the connectivity condition to (k-)internal 3-connectedness, then we cannot expect a 4-tree on the dual anymore. However, we always manage to find a tree such that at most one vertex of its co-tree has degree larger than 4.

Grünbaum's conjecture still remains open. We believe that it could prove worthwhile to assume further restrictions on the graph in order to decrease the maximum degree in both the tree and its co-tree or only one of them.

References

1 Md. J. Alam, W. Evans, S. G. Kobourov, S. Pupyrev, J. Toeniskoetter, and T. Ueckerdt. Contact representations of graphs in 3D. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS '15)*, volume 9214 of *Lecture Notes in Computer Science*, pages 14–27, 2015. Technical Report accessible on arXiv: `arXiv:1501.00304`. `doi:10.1007/978-3-319-21840-3_2`.

2 M. Badent, U. Brandes, and S. Cornelsen. More canonical ordering. *Journal of Graph Algorithms and Applications*, 15(1):97–126, 2011. `doi:10.7155/JGAA.00219`.

3 D. Barnette. Trees in polyhedral graphs. *Canadian Journal of Mathematics*, 18:731–736, 1966. `doi:10.4153/CJM-1966-073-4`.

4 D. W. Barnette. 2-Connected spanning subgraphs of planar 3-connected graphs. *Journal of Combinatorial Theory, Series B*, 61:210–216, 1994. `doi:10.1006/jctb.1994.1045`.

5 T. Biedl. Trees and co-trees with bounded degrees in planar 3-connected graphs. In *14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'14)*, pages 62–73, 2014. doi:10.1007/978-3-319-08404-6_6.

6 T. Böhme, J. Harant, M. Kriesell, S. Mohr, and J. M. Schmidt. Rooted minors and locally spanning subgraphs. *Journal of Graph Theory*, 105(2):209–229, 2024. doi:10.1002/jgt.23012.

7 R. Brunet, M. N. Ellingham, Z. Gao, A. Metzlar, and R. B. Richter. Spanning planar subgraphs of graphs in the torus and Klein bottle. *Journal of Combinatorial Theory, Series B*, 65(1):7–22, 1995. doi:10.1006/jctb.1995.1041.

8 P. O. de Mendez. *Orientations bipolaires.* PhD thesis, École des Hautes Études en Sciences Sociales, Paris, 1994.

9 G. Di Battista, R. Tamassia, and L. Vismara. Output-sensitive reporting of disjoint paths. *Algorithmica*, 23(4):302–340, 1999. doi:10.1007/PL00009264.

10 R. Diestel. *Graph theory.* Graduate texts in mathematics 173. Springer, Berlin, 4th edition edition, 2012. URL: http://swbplus.bsz-bw.de/bsz377230375cov.htm.

11 S. Felsner. Geodesic embeddings and planar graphs. *Order*, 20:135–150, 2003.

12 S. Felsner. *Geometric Graphs and Arrangements.* Advanced Lectures in Mathematics. Vieweg+Teubner, Wiesbaden, 2004. doi:10.1007/978-3-322-80303-0.

13 S. Felsner. Lattice structures from planar graphs. *Electronic Journal of Combinatorics*, 11(1):R15, 1–24, 2004. doi:10.37236/1768.

14 Stefan Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18(1):19–37, 2001. doi:10.1023/A:1010604726900.

15 É. Fusy. *Combinatorics of planar maps and algorithmic applications (Combinatoire des cartes planaires et applications algorithmiques).* PhD thesis, École Polytechnique, Palaiseau, France, 2007. URL: https://tel.archives-ouvertes.fr/pastel-00002931.

16 H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985.

17 Z. Gao. 2-Connected coverings of bounded degree in 3-connected graphs. *Journal of Graph Theory*, 20(3):327–338, 1995. doi:10.1002/JGT.3190200309.

18 Z. Gao and R. B. Richter. 2-Walks in circuit graphs. *Journal of Combinatorial Theory, Series B*, 62(2):259–267, 1994. doi:10.1006/JCTB.1994.1068.

19 B. Grünbaum. Polytopes, graphs, and complexes. *Bulletin of the American Mathematical Society*, 76(6):1131–1201, 1970. doi:10.1090/S0002-9904-1970-12601-5.

20 G. Kant. Drawing planar graphs using the lmc-ordering. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 101–110, 1992. doi:10.1109/SFCS.1992.267814.

21 G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996. doi:10.1007/BF02086606.

22 K. Ota and K. Ozeki. Spanning trees in 3-connected $K_{3,t}$-minor-free graphs. *Journal of Combinatorial Theory, Series B*, 102:1179–1188, 2012. doi:10.1016/J.JCTB.2012.07.002.

23 K. Ozeki and T. Yamashita. Spanning trees: A survey. *Graphs and Combinatorics*, 27:1–26, 2011. doi:10.1007/S00373-010-0973-2.

24 W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320191.

Finding Induced Subgraphs from Graphs with Small Mim-Width

Yota Otachi
Graduate School of Informatics, Nagoya University, Japan

Akira Suzuki
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Yuma Tamura
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract

In the last decade, algorithmic frameworks based on a structural graph parameter called mim-width have been developed to solve generally NP-hard problems. However, it is known that the frameworks cannot be applied to the Clique problem, and the complexity status of many problems of finding dense induced subgraphs remains open when parameterized by mim-width. In this paper, we investigate the complexity of the problem of finding a maximum induced subgraph that satisfies prescribed properties from a given graph with small mim-width. We first give a meta-theorem implying that various induced subgraph problems are NP-hard for bounded mim-width graphs. Moreover, we show that some problems, including Clique and Induced Cluster Subgraph, remain NP-hard even for graphs with (linear) mim-width at most 2. In contrast to the intractability, we provide an algorithm that, given a graph and its branch decomposition with mim-width at most 1, solves Induced Cluster Subgraph in polynomial time. We emphasize that our algorithmic technique is applicable to other problems such as Induced Polar Subgraph and Induced Split Subgraph. Since a branch decomposition with mim-width at most 1 can be constructed in polynomial time for block graphs, interval graphs, permutation graphs, cographs, distance-hereditary graphs, convex graphs, and their complement graphs, our positive results reveal the polynomial-time solvability of various problems for these graph classes.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases mim-width, graph algorithm, NP-hardness, induced subgraph problem, cluster vertex deletion

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.38

Related Version *Full Version*: `https://arxiv.org/abs/2405.15492`

Funding *Yota Otachi*: JSPS KAKENHI Grant Numbers JP18H04091, JP20H05793, JP21K11752, JP22H00513.
Akira Suzuki: JSPS KAKENHI Grant Numbers JP20K11666, JP20H05794.
Yuma Tamura: JSPS KAKENHI Grant Number JP21K21278.

1 Introduction

Efficiently solving intractable graph problems by using structural graph parameters has been extensively studied over the past few decades. Tree-width is arguably one of the most successful parameters in this research direction. Courcelle's celebrated result indicates that every problem expressible in MSO_2 logic is solvable in linear time for bounded tree-width graphs [12]. Various graph problems, including Independent Set, Clique, Dominating Set, Independent Dominating Set, k-Coloring for a fixed k, Feedback Vertex Set, and Hamiltonian Cycle, can be written in MSO_2 logic, and hence Courcelle's theorem covers a wide range of problems. Later, Courcelle et al. also gave an analogous result for a

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 38; pp. 38:1–38:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

more general parameter than tree-width, namely, clique-width: every problem expressible in MSO_1 logic is solvable in linear time for bounded clique-width graphs (under the assumption that a k-expression for a fixed k of an input graph is given) [13]. However, these results are not applicable directly to problems on interval graphs and permutation graphs, because these graph classes have unbounded clique-width (and thus unbounded tree-width).

In 2012, Vatshelle introduced mim-width [33], and recently, algorithms based on mim-width have been widely developed [1, 2, 3, 4, 5, 7, 8, 9, 15, 16, 21, 23, 24]. Roughly speaking, mim-width is an upper bound on the size of maximum induced matching along a branch decomposition of a graph. (In Section 2, its formal definition will be given.) Mim-width is a more general structural parameter than clique-width in the sense that the class of bounded mim-width graphs properly contains the class of bounded clique-width graphs. Furthermore, many graph classes of unbounded clique-width have bounded mim-width: for example, interval graphs, permutation graphs, convex graphs, k-polygon graphs for a fixed k, circular k-trapezoid graphs for a fixed k, and H-graphs for a fixed graph H. (See [1, 14] for more details.) Bergougnoux et al. gave an algorithmic meta-theorem [2], which states that every problem expressible in A&C DN logic is solvable in polynomial time for bounded mim-width graphs (under the assumption that a suitable branch decomposition of an input graph is given). Independent Set, Dominating Set, Independent Dominating Set, k-Coloring for a fixed k, Feedback Vertex Set etc. can be expressed in A&C DN logic. Thus, Bergougnoux et al. showed that many problems are solvable in polynomial time for a much wider range of graph classes than the class of bounded clique-width graphs.

Unfortunately, A&C DN logic does not cover all problems expressible in MSO_2 logic. Clique and Hamiltonian Cycle cannot be written in A&C DN logic, whereas they can be expressed in MSO_1 logic and MSO_2 logic, respectively. This means that the meta-theorem by Bergougnoux et al. is not applicable to these problems. In fact, it is known that Clique is NP-hard for graphs with linear mim-width[1] at most 6 [33] and Hamiltonian Cycle is NP-hard for graphs with linear mim-width 1 [23]. Note that by combining some known facts, we can show that Clique on graphs with mim-width at most 1 can be solved in polynomial time (see the discussion in the second paragraph of Section 4). These results lead us to ask the following questions:

- What kind of problems expressible in MSO_2 logic are NP-hard for bounded mim-width graphs?
- Is Clique NP-hard for graphs with mim-width less than 6?
- Given a graph with mim-width at most 1, which MSO_2-expressible problems are polynomial-time solvable?

1.1 Our contributions

To answer the questions above, in this paper, we systematically study the complexity of the Induced Π Subgraph problems and their complementary problems, called the Π Vertex Deletion problems, on bounded (linear) mim-width graphs. We first show that for any nontrivial hereditary graph property Π that admits all cliques, there is a constant w such that Induced Π Subgraph and Π Vertex Deletion are NP-hard for graphs with (linear) mim-width at most w. For example, Clique, Induced Cluster Subgraph, Induced Polar Subgraph, and Induced Split Subgraph satisfy the aforementioned conditions, and hence all of them are NP-hard for bounded (linear) mim-width graphs. As a

[1] The linear mim-width of a graph G is the mim-width when a branch decomposition of G is restricted to a caterpillar. The formal definition will be given in Section 2.

byproduct, we also show that connected and dominating variants of them are NP-hard for bounded (linear) mim-width graphs. Moreover, we give sufficient conditions for Induced Π Subgraph and Π Vertex Deletion to be NP-hard for graphs with (linear) mim-width at most 2. Clique, Induced Cluster Subgraph, Induced Polar Subgraph, and Induced Split Subgraph are proven to be in fact NP-hard even for graphs with (linear) mim-width at most 2. We thus reveal that there are various NP-hard problems for bounded mim-width graphs, although they can be expressed in MSO_2 logic. Especially, our result for Clique strengthens the known result that Clique is NP-hard for graphs with mim-width at most 6 [33].

To complement the intractability, we next seek polynomial-time solvable cases for graphs with mim-width at most 1. Here we focus on Induced Cluster Subgraph, also known as Cluster Vertex Deletion. Induced Cluster Subgraph is known to be NP-hard for bipartite graphs [19, 34], while it is solvable in polynomial time for split graphs, block graphs, interval graphs [10], cographs [27], bounded clique-width graphs [13], and convex graphs[2]. Surprisingly, the complexity status of Induced Cluster Subgraph on chordal graphs is still open. We show that, given a graph G with mim-width at most 1 accompanied by its branch decomposition with mim-width at most 1, Induced Cluster Subgraph is solvable in polynomial time. Although the complexity of computing a branch decomposition with mim-width at most 1 of a given graph is still open in general, our result yields a unified polynomial-time algorithm for Induced Cluster Subgraph that works on block graphs, interval graphs, permutation graphs, cographs, distance-hereditary graphs, convex graphs, and their complement graphs because all these graphs have mim-width at most 1 and their branch decompositions of mim-width at most 1 can be obtained in polynomial time [1, 20, 33][3]. Consequently, we give independent proofs for some of the results in [10, 27] via mim-width. Moreover, to the best of our knowledge, this is the first polynomial-time algorithm for Induced Cluster Subgraph on permutation graphs. We also emphasize that our algorithmic technique can be applied to other problems such as Induced Polar Subgraph, Induced Split Subgraph, and so on. Combining our results, we give the complexity dichotomy of the above problems with respect to mim-width.

Due to the space limitation, the proofs of claims marked ♠ are omitted in this paper, which can be found in the full version.

1.2 Previous work on mim-width

Mim-width is a relatively new graph structural parameter introduced by Vatshelle [33] and it has attracted much attention in recent years to design efficient algorithms of problems on graph classes that have unbounded tree-width and clique-width. Combined with the result of Belmonte and Vatshelle [1], Bui-Xuan et al. provided XP algorithms of Locally Checkable Vertex Subset and Vertex Partitioning problems (LC-VSVP for short) parameterized by mim-width w, assuming that a branch decomposition with mim-width w of a given graph can be computed in polynomial time [9]. Many problems, including Independent Set, Dominating Set, Independent Dominating Set, and k-Coloring, are expressible in the form of LC-VSVP. Jaffke et al. later generalized the result to the

[2] If a given graph is convex (more generally K_3-free), Induced Cluster Subgraph is equivalent to Induced Π Subgraph such that Π is the class of graphs with maximum degree at most 1, which is solvable in polynomial time for convex graphs [9].

[3] As far as we know, it was not explicitly stated in any literature that block graphs and distance-hereditary graphs have mim-width at most 1. This follows from the facts that a graph is distance-hereditary if and only if its rank-width is at most 1 [20], and block graphs are distance-hereditary graphs.

distance versions of LC-VSVP [21]. As the name suggests, LC-VSVP can capture problems whose solutions are defined only by local constraints. Longest Induced Path [23] and Feedback Vertex Set [24] are the first problems with global constraints for which it was shown that there exist XP algorithms parameterized by mim-width. Bergougnoux and Kanté designed a framework to deal with problems with global constraints for bounded mim-width graphs [3]. The remarkable meta-theorem given by Bergougnoux et al. is not only a generalization of all the above results in this section, but also a powerful tool for solving more complicated problems on bounded mim-width graphs [2]. Subset Feedback Vertex Set is one of the few examples where there exists an XP algorithm parameterized by mim-width [4] although the meta-theorem does not work for it.

Unfortunately, computing the mim-width of a given graph is W[1]-hard, and there is no polynomial-time approximation algorithm within constant factor unless NP = ZPP [32]. Even the complexity of determining whether a given graph has mim-width at most 1 is a long-standing open problem. Fortunately, it is known that various graph classes have constant mim-width and their branch decompositions with constant mim-width are computable in polynomial time [1, 7, 8, 14, 26, 30]. In particular, some famous graphs, such as block graphs, interval graphs, permutation graphs, cographs, distance-hereditary graphs, and convex graphs, have mim-width at most 1 and their branch decomposition with mim-width at most 1 can be obtained in polynomial time [1, 20]. The class of leaf power graphs, which is the more general class than interval graphs and block graphs, also have mim-width at most 1 [22], although it is not known whether an optimal branch decomposition of a given leaf power graph can be obtained in polynomial time. On the other hand, the following graph classes have unbounded mim-width: strongly chordal split graphs [29], co-comparability graphs [26, 29], circle graphs [29], and chordal bipartite graphs [6].

In contrast to a wealth of research on developing XP algorithms parameterized by mim-width and establishing lower and upper bounds on mim-width for specific graph classes, there has been limited research on the NP-hardness of problems for graph classes with constant mim-width [23, 25, 33].

2 Preliminaries

Let $G = (V, E)$ be a graph. We assume that all the graphs in this paper are simple, undirected, and unweighted. We denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. For a vertex v of G, we denote by $N(G; v)$ the *(open) neighborhood* of v in G, that is, $N(G; v) = \{w \in V \mid vw \in E\}$. The *degree* of a vertex v of G is the size of $N(G; v)$. For a vertex subset $V' \subseteq V$, we denote by $G[V']$ the subgraph induced by V'. We use the shorthand $G - V'$ for $G[V \setminus V']$. For positive integers i and j with $i \leq j$, we write $[i, j]$ as the shorthand for the set $\{i, i+1, \ldots, j\}$ of integers. In particular, we write $[1, j] = [j]$.

For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$, the *disjoint union* of G_1 and G_2 is the graph whose vertex set is $V_1 \cup V_2$ and edge set is $E_1 \cup E_2$. For a graph H and a positive integer ℓ, ℓH means the disjoint union of ℓ copies of H. The *complement* of G, denoted by $\overline{G}$, is the graph on the same vertex set $V(G)$ with the edge set $\{uv \mid u, v \in V(G), uv \notin E(G)\}$. An *independent set* I of G is a vertex subset of G such that any two vertices in I are non-adjacent. A *clique* K of G is a vertex subset of G such that any two vertices in K are adjacent. Obviously, an independent set of G forms a clique of $\overline{G}$, and vice versa. A *dominating set* D of G is a vertex subset of G such that $N(G; v) \cap D \neq \emptyset$ for every vertex $v \in V(G) \setminus D$. A graph G is said to be *connected* if there is a path between any two vertices of G. A maximal connected subgraph of G is called a *connected component* of G. A *cut vertex* of G is a vertex whose removal from G increases the number of connected components.

2.1 Graph classes

A graph is *bipartite* if its vertex set can be partitioned into two independent sets. For disjoint vertex sets A and B of a graph G, we denote by $G[A, B]$ the *bipartite subgraph* with the vertex set $A \cup B$ and the edge set $\{ab \in E(G) \mid a \in A, b \in B\}$. A bipartite graph $G = (A \cup B, E)$ consisting of disjoint independent sets A and B is called a *chain graph* if there is an ordering $a_1, a_2, \ldots, a_{|A|}$ of vertices in A such that $N(G; a_1) \subseteq N(G; a_2) \subseteq \cdots \subseteq N(G; a_{|A|})$. Note that, if A has such an ordering, then B also has an ordering $b_1, b_2, \ldots, b_{|B|}$ of vertices in B such that $N(G; b_1) \subseteq N(G; b_2) \subseteq \cdots \subseteq N(G; b_{|B|})$.

A *tree* is a connected acyclic graph. A vertex of a tree is called a *leaf* if it has degree 1; otherwise, it is an *internal vertex*. A *rooted tree* T is a tree with a specific vertex r called the *root* of T. For a rooted tree T and two adjacent vertices x and y of T, we say that x is the *parent* of y, and conversely, y is a *child* of x if x lies on a path from y to r. A *full binary tree* is a rooted tree such that each vertex has zero or exactly two children. A tree T is a *caterpillar* if it contains a path P called a *spine* such that every leaf of T is adjacent to a vertex of P. In this paper, we assume that the spine P is maximum, that is, there is no path longer than P. The vertices of degree at most 1 in P are called the *endpoints* of P. A tree T is called *subcubic* if every internal vertex of T has degree exactly 3.

We denote by K_n and P_n the complete graph and the path graph with n vertices, respectively. We say that a graph G is *H-free* if G does not contain a graph isomorphic to H as an induced subgraph.

2.2 Mim-width

For an edge subset E' of a graph G, we denote $V(E') = \{v, w \in V(G) \mid vw \in E'\}$. An edge subset $M \subseteq E(G)$ is an *induced matching* of G if every vertex of $G[V(M)]$ has degree exactly 1. For a vertex subset $A \subseteq V(G)$, let $\mathsf{mim}(A)$ be the maximum size of an induced matching in the bipartite subgraph $G[A, \overline{A}]$, where $\overline{A} = V(G) \setminus A$.

A *branch decomposition* of a graph G is a pair (T, L), where T is a subcubic tree with $|V(G)|$ leaves and L is a bijection from $V(G)$ to the leaves of T. In particular, a branch decomposition (T, L) is called *linear* if T is a caterpillar. To distinguish vertices of T from those of the original graph G, we call the vertices of T *nodes*. For each edge e of T, as T is acyclic, removing e from T results in two trees T_1^e and T_2^e. Let (A_1^e, A_2^e) be a vertex bipartition of G, where $A_i^e = \{L^{-1}(\ell) \mid \ell \text{ is a leaf of } T_i^e\}$ for each $i \in \{1, 2\}$. The *mim-width* $\mathsf{mimw}(T, L)$ of a branch decomposition (T, L) of G is defined as $\max_{e \in E(T)} \mathsf{mim}(A_1^e)$. The *mim-width* $\mathsf{mimw}(G)$ of G is the minimum mim-width over all branch decompositions of G. Similarly, the *linear mim-width* $\mathsf{lmimw}(G)$ of G is the minimum mim-width over all linear branch decompositions of G. Note that $\mathsf{mimw}(G) \leq \mathsf{lmimw}(G)$ holds for any graph G.

In this paper, to make a branch decomposition easier to handle, we often consider its rooted variant. A *rooted layout* of a graph G is a pair (T', L), where T' is a rooted full binary tree with $|V(G)|$ leaves and L is a bijection from $V(G)$ to the leaves of T'. The mim-width of a rooted layout (T', L) is defined in the same way as a branch decomposition. A rooted layout of G is obtained from a branch decomposition (T, L) of G with the same mim-width by inserting a root r to an arbitrary edge of T. (If $|V(T)| = 1$, we regard the unique node of T as the root r of T'.)

Here we note propositions concerning mim-width. Vatshelle showed that for a graph G and a vertex $v \in V(G)$, it holds that $\mathsf{mimw}(G - v) \leq \mathsf{mimw}(G)$ [33]. One can see that the proof given by Vatshelle suggests the next proposition.

▶ **Proposition 1.** *For a graph G and an induced subgraph G' of G, it holds that* $\mathsf{mimw}(G') \leq \mathsf{mimw}(G)$ *and* $\mathsf{lmimw}(G') \leq \mathsf{lmimw}(G)$.

We here focus on graphs with mim-width at most 1. It is known that a graph G is a chain graph if and only if G is a bipartite graph with a maximum induced matching of size at most 1 [18]. Thus, we obtain the following proposition.

▶ **Proposition 2.** *Let (T, L) be a branch decomposition of a graph G. Then,* $\mathsf{mimw}(T, L) \leq 1$ *if and only if for any edge e of T, the bipartite subgraph $G[A_1^e, A_2^e]$ of G is a chain graph.*

Moreover, for a graph G and a vertex subset $A \subset V(G)$, it is not hard to see that $\mathsf{mim}(A) \leq 1$ on G if and only if $\mathsf{mim}(A) \leq 1$ on $\overline{G}$ from the definition of a chain graph. This implies the following proposition.

▶ **Proposition 3** ([33]). *Suppose that a graph G has mim-width at most 1. Then, any branch decomposition (T, L) of G with* $\mathsf{mimw}(T, L) \leq 1$ *is also the branch decomposition of $\overline{G}$ with* $\mathsf{mimw}(T, L) \leq 1$. *Consequently,* $\mathsf{mimw}(G) \leq 1$ *if and only if* $\mathsf{mimw}(\overline{G}) \leq 1$.

Combined with the observation that any cycle of length at least 5 has mim-width 2 and the strong perfect graph theorem [11], Proposition 3 leads to the following proposition.

▶ **Proposition 4** ([33]). *All graphs with mim-width at most 1 are perfect graphs.*

2.3 Graph properties and problems

Let Π be a fixed graph property. We often regard Π as a collection of graphs satisfying the graph property. A graph property Π is *nontrivial* if there exist infinitely many graphs satisfying Π and there exist infinitely many graphs that do not satisfy Π. A graph property Π is said to be *hereditary* if for any graph G satisfying Π, every induced subgraph of G also satisfies Π. We denote by $\overline{\Pi}$ the *complementary property* of Π, that is, $\overline{\Pi} = \{\overline{G} : G \in \Pi\}$.

For a graph G, a vertex subset $S \subseteq V(G)$ is called a Π-*set* of G if $G[S]$ satisfies Π. The Induced Π Subgraph problem asks for a Π-set S of maximum size for a given graph G. If $G[S]$ is also required to be connected, then the problem is called the Connected Induced Π Subgraph problem. For example, Independent Set is equivalent to Induced K_2-free Subgraph, and Clique is equivalent to Induced $2K_1$-free Subgraph and Connected Induced P_3-free Subgraph. Note that a vertex set S of G is a Π-set if and only if S is a $\overline{\Pi}$-set of $\overline{G}$. In Induced Π Subgraph, if the Π-set S is also required to be a dominating set of G, then the problem is called the Dominating Induced Π Subgraph problem.

Under the polynomial-time solvability, Induced Π Subgraph is equivalent to the Π Vertex Deletion problem, which asks for a minimum vertex subset S' of G such that $G - S'$ satisfies Π. The vertex subset S' is called a Π-*deletion set* of G. The Vertex Cover problem is equivalent to K_2-free Vertex Deletion. If $G[S']$ is also required to be connected, then the problem is called the Connected Π Vertex Deletion problem. In Π Vertex Deletion, if the Π-deletion set S' is also required to be a dominating set of G, then the problem is called the Dominating Π Vertex Deletion problem.

3 NP-hardness

In this section, we show the NP-hardness of Induced Π Subgraph and Π Vertex Deletion on graphs with linear mim-width at most w, where w is some constant.

▶ **Theorem 5.** *Let Π be a fixed nontrivial hereditary graph property that admits all cliques. Then there is a constant w such that* Induced Π Subgraph *and* Π Vertex Deletion*, as well as their connected variants and their dominating variants, are NP-hard for graphs with linear mim-width at most w, even if a branch decomposition with mim-width at most w of an input graph is given.*

Since Induced Π Subgraph is the complementary problem of Π Vertex Deletion, we only prove the hardness of Π Vertex Deletion.

The *girth* of a graph G is the length of a shortest cycle in G. We reduce Vertex Cover on graphs with girth at least 7, which is known to be NP-complete [31], to Π Vertex Deletion by following the classical reduction technique of Lewis and Yannakakis [28].

First, we define a sequence on a graph. Consider a graph H with p connected components $H_1, H_2, \dots, H_p$. Suppose that H_i for $i \in [p]$ has a cut vertex c and the removal of c from H_i results in q connected components $C_{i,1}, C_{i,2}, \dots, C_{i,q}$ with $|V(C_{i,1})| \geq |V(C_{i,2})| \geq \cdots \geq |V(C_{i,q})|$. For each $j \in [q]$, we denote by $H_{i,j}$ the subgraph induced by $V(C_{i,j}) \cup \{c\}$ and $n_{i,j} = |V(H_{i,j})|$. The cut vertex c gives a non-increasing sequence $\alpha_c = \langle n_{i,1}, n_{i,2}, \dots, n_{i,q} \rangle$. For two sequences α_c and $\alpha_{c'}$ according to cut vertices c and c' of H_i, we write $\alpha_{c'} <_L \alpha_c$ if $\alpha_{c'}$ is smaller than α_c in the sense of lexicographic order. Let α_i be the lexicographically smallest sequence among all sequences according to the cut vertices of H_i. If H_i has no cut vertex, we let $\alpha_i = \langle |V(H_i)| \rangle$. Define $\beta_H = \langle \alpha_1, \alpha_2, \dots, \alpha_p \rangle$, where we assume that $\alpha_1 \geq_L \alpha_2 \geq_L \cdots \geq_L \alpha_p$. For example, for the graph H depicted in Figure 1(a), we have $\beta_H = \langle \langle 4, 2 \rangle, \langle 2, 2 \rangle \rangle$. For two graphs H with p connected components and H' with q connected components, we write $\beta_{H'} <_R \beta_H$ if $\beta_{H'}$ is smaller than β_H in the sense of lexicographic order: more precisely, assuming that $\beta_H = \langle \alpha_1, \alpha_2, \dots, \alpha_p \rangle$ and $\beta_{H'} = \langle \alpha'_1, \alpha'_2, \dots, \alpha'_q \rangle$, there exists an integer $i \in [\min\{p, q\}]$ such that $\alpha'_j = \alpha_j$ for every $j \in [i-1]$ and $\alpha'_i <_L \alpha_i$; or $q < p$ and $\alpha'_i = \alpha_i$ for every $i \in [q]$.

Consider the complementary property $\overline{\Pi}$ of Π. Note that, since all cliques satisfy Π, all independent sets satisfy $\overline{\Pi}$. Let F be a graph satisfying the following two conditions:

1. there is an integer $\ell \geq 1$ such that ℓF violates $\overline{\Pi}$, whereas $(\ell - 1)F$ satisfies $\overline{\Pi}$; and
2. for any integer $\ell' \geq 1$ and any graph F' with $\beta_{F'} <_R \beta_F$, $\ell' F'$ satisfies $\overline{\Pi}$.

We call F the *base of* $\overline{\Pi}$*-forbidden subgraphs*. Notice that the existence of F is guaranteed because $\overline{\Pi}$ is nontrivial. Moreover, F and ℓ depend on Π solely and are independent of an instance of Vertex Cover, that is, F and ℓ are fixed.

Let $F_1, F_2, \dots, F_p$ be p connected components of F, where $\alpha_1 \geq_L \alpha_2 \geq_L \cdots \geq_L \alpha_p$. We denote by c_1 the cut vertex of F_1 that realizes α_1 (see Figure 1(a)) and by $F_{1,1}$ the induced subgraph of F_1 corresponding to $n_{1,1}$ (see Figure 1(b)). If F_1 has no cut vertex, then c_1 is any vertex of F_1. We then arbitrarily choose a vertex from $N(F_{1,1}; c_1)$ and label it as d. Notice that $N(F_{1,1}; c_1) \neq \emptyset$; otherwise, since $\alpha_1, \alpha_2, \dots, \alpha_p$ are lexicographically sorted, ℓF is an independent set and violates $\overline{\Pi}$, which contradicts that all independent sets satisfy $\overline{\Pi}$. Let F' be the graph obtained by removing $V(F_{1,1}) \setminus \{c_1\}$ from F (see Figure 1(c)).

We now construct an input graph G for Π Vertex Deletion from an input graph H with girth at least 7 for Vertex Cover. Let $n = |V(H)|$ and H^* be the disjoint union of ℓn copies of H. We assume that $n \geq 2$, $k < n - 1$, and H has at least one edge; otherwise, Vertex Cover is trivially solvable. For each vertex u of H^*, make a copy of F' and identify c_1 with u. For each edge uv of H^*, make a copy of $F_{1,1}$ and identify c_1 and d with u and v, respectively. (See Figure 2.) Let H' be the graph resulting from the above transformation. Finally, we let $G = \overline{H'}$. Since ℓ, F', and $F_{1,1}$ are fixed, G can be constructed in polynomial time in the size of H.

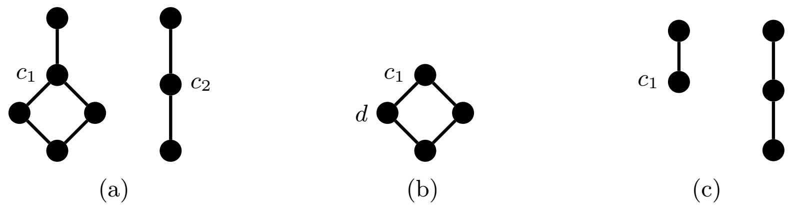


Figure 1 Let F be the graph depicted in (a). The cut vertex c_1 of the left connected component F_1 of F gives $\alpha_1 = \langle 4, 2\rangle$ and the cut vertex c_2 of the right connected component F_2 of F gives $\alpha_2 = \langle 2, 2\rangle$, where $\alpha_1 >_L \alpha_2$. Thus, if F is selected as the base of $\overline{\Pi}$-forbidden subgraphs, $F_{1,1}$ and F' are defined as the graphs depicted in (b) and (c), respectively.

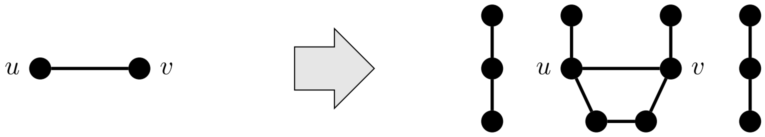


Figure 2 A transformation of an edge uv with $F_{1,1}$ and F', which are the graphs depicted in Figure 1(b) and (c), respectively.

In [28], it is shown that H has a vertex cover of size at most k if and only if H' has a $\overline{\Pi}$-deletion set S of size at most $k\ell n$. Notice that H' has $\ell n \geq 2$ connected components because H' is obtained from H^*, which is the disjoint union of ℓn copies of H. Moreover, we have the following lemma.

▶ **Lemma 6.** *Suppose that H' has a $\overline{\Pi}$-deletion set S of size at most $k\ell n$. Then the following two claims (a) and (b) are true:*

(a) *there are two connected components C_1 and C_2 of H' such that $V(C_1) \setminus S \neq \emptyset$ and $V(C_2) \setminus S \neq \emptyset$; and*

(b) *there are two connected components C_1' and C_2' of H' such that $V(C_1') \cap S \neq \emptyset$ and $V(C_2') \cap S \neq \emptyset$.*

Proof. In the claim (a), assume for a contradiction that there is at most one connected component C of H' such that $V(C) \setminus S \neq \emptyset$. In other words, $V(H' - C) \subseteq S$ holds. Recall that $k < n - 1$ and H' has $\ell n \geq 2$ connected components. Moreover, each connected component of $H' - C$ has at least n vertices from the construction of H'. Thus, we have

$$|S| \geq |V(H' - C)| \geq n(\ell n - 1) > (k + 1)(\ell n - 1) = k\ell n + \ell n - k - 1 > k\ell n,$$

a contradiction.

To prove the claim (b), assume for a contradiction that there is at most one connected component C' of H' such that $V(C') \cap S \neq \emptyset$. In other words, there are at least $\ell n - 1$ ($\geq \ell$ because $n \geq 2$) connected components of $H' - C'$ that contain no vertex in S. Consider ℓ connected components of $H' - C'$. Since each of them contains F as an induced subgraph, $H' - C'$ contains ℓF as an induced subgraph. However, ℓF violates $\overline{\Pi}$ because F is the base of $\overline{\Pi}$-forbidden subgraphs. This contradicts that S is a $\overline{\Pi}$-deletion set of H'. ◀

Observe that S is a $\overline{\Pi}$-deletion set of H' of size at most $k\ell n$ if and only if S is a Π-deletion set of $G = \overline{H'}$ of size at most $k\ell n$. Combined with Lemma 6, this implies that H has a vertex cover of size at most k if and only if G has a Π-deletion set S of size at most $k\ell n$ such that the induced subgraphs $G[S]$ and $G - S$ are both connected, and S and $V(G) \setminus S$ are dominating sets of G.

Our remaining task is to show that G has linear mim-width at most w for some constant w. To this end, we consider a sequence of subgraphs of H'. Let $V(H^*) = \{v_1, v_2, \dots, v_n\}$ and $E(H^*) = \{e_1, e_2, \dots, e_m\}$, where n and m are the numbers of vertices and edges of H^*, respectively. We make a sequence $\mathcal{H} = \langle H^* = H_0, H_1, \dots, H_{n+m} = H' \rangle$ such that H_i for $i \in [n]$ is obtained from H_{i-1} by attaching a copy of F' to v_i, and H_i for $i \in [n+1, n+m]$ is obtained from H_{i-1} by attaching a copy of $F_{1,1}$ to e_{i-n}. Since $G = \overline{H'}$, the following lemma completes the proof of Theorem 5. (Recall that F is fixed and hence $\mathsf{lmimw}(\overline{F})$ is a constant.)

▶ **Lemma 7.** *For any graph H_i in the sequence $\mathcal{H} = \langle H^* = H_0, H_1, \dots, H_{n+m} = H' \rangle$, a linear branch decomposition of H_i with mim-width at most $\mathsf{lmimw}(\overline{F}) + 2$ can be obtained in polynomial time in the size of H^*.*

Proof. We prove the lemma by induction, where the base case is $H_0 = H^*$. Note that H^* has girth at least 7 because H^* consists of copies of H whose girth is at least 7. Consider a linear branch decomposition (T_0, L_0) of $\overline{H_0}$, where L_0 is an arbitrary bijection from $V(H_0)$ to the leaves of T_0. To show that $\mathsf{mimw}(T_0, L_0) \le 2 \le \mathsf{lmimw}(\overline{F}) + 2$, assume for a contradiction that there is an edge e of T_0 such that $\mathsf{mim}(A_1^e) \ge 3$ for the bipartition (A_1^e, A_2^e). Let x_1x_2, y_1y_2, z_1z_2 be edges that form an induced matching in $G[A_1^e, A_2^e]$, where $x_1, y_1, z_1 \in A_1^e$ and $x_2, y_2, z_2 \in A_2^e$. Then, $x_1y_2, y_2z_1, z_1x_2, x_2y_1, y_1z_2, z_2x_1 \notin E(\overline{H})$ and hence they form a cycle of length 6 in H_0. This contradicts that the girth of H_0 is at least 7.

Consider the case of $i > 0$. We here define a *concatenation* of two linear branch decompositions. Let G_1 and G_2 be vertex-disjoint induced subgraphs of a graph G such that $V(G_1) \cup V(G_2) = V(G)$, and let (T_1, L_1) and (T_2, L_2) be linear branch decompositions of G_1 and G_2, respectively. A concatenation of (T_1, L_1) and (T_2, L_2) is to construct a new linear branch decomposition (T, L) of G as follows. For each $i \in \{1, 2\}$, let e_i be an edge incident to an endpoint of the spine of T_i. Insert nodes t_1 and t_2 into e_1 and e_2, respectively, and then connect t_1 and t_2 by an edge. (If $|V(T_i)| = 1$ for $i \in \{1, 2\}$, we define t_i as the unique node of T_i.) Observe that T is a subcubic caterpillar. Finally, set a bijection L from $V(G)$ to the leaves of T such that $L(v) = L_1(v)$ if $v \in V(G_1)$ and $L(v) = L_2(v)$ if $v \in V(G_2)$.

By the induction hypothesis, there exists a linear branch decomposition (T_{i-1}, L_{i-1}) of $\overline{H_{i-1}}$ such that $\mathsf{mimw}(T_{i-1}, L_{i-1}) \le \mathsf{lmimw}(\overline{F}) + 2$. Recall that H_i is constructed by attaching a copy of F' to v_i or a copy of $F_{1,1}$ to e_{i-n}. We denote by F_i the subgraph of H_i obtained by removing all vertices in $V(H_{i-1})$. We may assume that $|V(F_i)| \ge 1$; otherwise, $H_i = H_{i-1}$ and thus we immediately conclude that $\mathsf{lmimw}(\overline{H_i}) \le \mathsf{lmimw}(\overline{F}) + 2$. Let (T_i', L_i') be a linear branch decomposition of $\overline{F_i}$ such that $\mathsf{mimw}(T_i', L_i') \le \mathsf{lmimw}(\overline{F})$. Notice that, since $\overline{F_i}$ is an induced subgraph of $\overline{F}$, such a linear branch decomposition exists by Proposition 1. Moreover, it can be constructed in constant time because $\overline{F}$ is fixed. We define (T_i, L_i) as a linear branch decomposition obtained by a concatenation of (T_{i-1}, L_{i-1}) and (T_i', L_i'). Clearly, the construction of (T_i, L_i) can be done in polynomial time in the size of H^*.

To show that $\mathsf{mimw}(T_i, L_i) \le \mathsf{lmimw}(\overline{F}) + 2$, assume for a contradiction that there is an edge e of T_i such that the bipartite subgraph $G[A_1^e, A_2^e]$ of $\overline{H_i}$ has an induced matching M of size $\mathsf{lmimw}(\overline{F}) + 3$, where (A_1^e, A_2^e) is the bipartition of $V(H_i)$ given by e. From the construction of (T_i, L_i), the following two cases are considered: (I) $A_1^e \subseteq V(H_{i-1})$ and $V(F_i) \subseteq A_2^e$; and (II) $A_1^e \subseteq V(F_i)$ and $V(H_{i-1}) \subseteq A_2^e$.

Case (I). Let e' be an edge of T_{i-1} such that $A_1^{e'} = A_1^e$ and $A_2^{e'} = A_2^e \setminus V(F_i)$. If $V(M) \subseteq V(H_{i-1})$, then M is also an induced matching of the bipartite subgraph $G[A_1^{e'}, A_2^{e'}]$ defined by the linear branch decomposition (T_{i-1}, L_{i-1}). This implies that $\mathsf{mimw}(T_{i-1}, L_{i-1}) \ge |M| = \mathsf{lmimw}(\overline{F}) + 3$, which contradicts that $\mathsf{mimw}(T_{i-1}, L_{i-1}) \le \mathsf{lmimw}(\overline{F}) + 2$.

Without loss of generality, we assume that M has three distinct edges x_1x_2, y_1y_2, z_1z_2 such that $x_1, y_1, z_1 \in A_1^e \subseteq V(H_{i-1})$, $x_2 \in A_2^e \cap V(F_i)$, and $y_2, z_2 \in A_2^e$. Then, the sequence $\langle x_2, y_1, z_2, x_1, y_2, z_1, x_2 \rangle$ of vertices forms a cycle C of length 6 of H_i. If $i \in [n]$, as $x_2 \in V(F_i)$ is adjacent to at most one vertex in $V(H_{i-1})$ from the construction of H_i, then we have $y_1 = z_1$, a contradiction. Suppose that $i \in [n+1, n+m]$. Recall that, from the construction of H_i, each vertex of F_i is not adjacent to vertices in $V(H_{i-1})$ except for the endpoints of e_{i-n}. Since $x_2 \in V(F_i)$ is adjacent to the distinct vertices $y_1, z_1 \in V(H_{i-1})$, we have $e_{i-n} = y_1z_1$. Furthermore, $x_1 \in V(H_{i-1})$ is not adjacent to any vertex in $V(F_i)$ and hence we have $y_2, z_2 \in V(H_{i-1})$. Therefore, we obtain the cycle $C_1 = \langle y_1, z_2, x_1, y_2, z_1, y_1 \rangle$ with smaller length than that of C, where the vertices of C_1 are in $V(H_{i-1})$. Similarly, if C_1 contains vertices of F_j for $j \in [n+1, i]$, there exists a smaller cycle of H_{i-1} that contains no vertices of F_j. We eventually obtain a cycle C' of H of length less than 6, which contradicts that H has girth at least 7.

Case (II). Recall that at most two vertices in $V(H_{i-1})$, say u and w, are adjacent to some vertex in $V(F_i)$ on H_i and thus no vertex in $V(H_{i-1}) \setminus \{u, w\}$ is adjacent to any vertex in $V(F_i)$ on H_i. If some vertex in $V(M)$ is in $V(H_{i-1}) \setminus \{u, w\}$, then we can take three distinct edges $x_1x_2, y_1y_2, z_1z_2 \in M$ such that $x_1, y_1, z_1 \in A_1^e \subseteq V(F_i)$, $x_2 \in V(H_{i-1}) \setminus \{u, w\} \subseteq A_2^e$, and $y_2, z_2 \in A_2^e$. However, this implies that x_2 is adjacent to y_1 and z_1 on H_i, which contradicts that no vertex in $V(H_{i-1}) \setminus \{u, w\}$ is adjacent to any vertex in $V(F_i)$ on H_i.

If there is no vertex in $V(M)$ is in $V(H_{i-1}) \setminus \{u, w\}$, then there is an induced matching $M' \subseteq M$ of $G[A_1^e, A_2^e]$ such that $V(M') \subseteq V(F_i)$ and $|M'| \geq |M| - |V(M) \cap \{u, w\}| \geq \mathsf{lmimw}(\overline{F}) + 1$. For an edge e' of T_i' such that $A_1^{e'} = A_1^e$ and $A_2^{e'} = A_2^e \setminus V(H_{i-1})$, M' is also an induced matching of $G[A_1^{e'} A_2^{e'}]$ defined by the linear branch decomposition (T_i', L_i'). This implies that $\mathsf{mimw}(T_i', L_i') \geq \mathsf{lmimw}(\overline{F}) + 1$, which contradicts that $\mathsf{mimw}(T_i', L_i') \leq \mathsf{lmimw}(\overline{F})$. ◀

Refining the proof of Lemma 7 yields stronger claims for some problems. (See the full version of this paper.)

▶ **Theorem 8** (♠). *All the following problems, as well as their connected variants and their dominating variants, are NP-hard for graphs with linear mim-width 2: (i) Clique; (ii) Induced Cluster Subgraph; (iii) Induced Polar Subgraph (iv) Induced $\overline{P_3}$-free Subgraph; (v) Induced $\overline{K_3}$-free Subgraph; and (vi) Induced Split Subgraph. The NP-hardness for these problems holds even if a linear branch decomposition with mim-width at most 2 of an input graph is given.*

Theorem 8 strongly suggests that the complements of graphs with linear mim-width 2 have unbounded mim-width, because Independent Set, the complementary problem of Clique, is solvable in polynomial time for bounded mim-width graphs.

4 Polynomial-time algorithms for graphs with mim-width at most 1

A graph G is called a *cluster* if every connected component of G is a complete graph. Induced Cluster Subgraph is equivalent to Induced P_3-free Subgraph and Cluster Vertex Deletion (in terms of polynomial-time solvability). From Theorem 8, Induced Cluster Subgraph is NP-hard for graphs with linear mim-width at most 2.

Recall that all graphs with mim-width at most 1 are perfect graphs by Proposition 4. It is known that Clique is solvable in polynomial time for perfect graphs [17] and hence also for graphs with mim-width at most 1. In contrast, Induced Cluster Subgraph remains

NP-hard for bipartite graphs [19, 34], which are perfect graphs. Thus, the same argument as Clique is not applicable to Induced Cluster Subgraph. Nevertheless, assuming that a rooted layout (T, L) of an input graph with $\mathsf{mimw}(T, L) = 1$ is given, we design a polynomial-time algorithm for Induced Cluster Subgraph.

▶ **Theorem 9.** *Given a graph and its rooted layout of mim-width at most* 1*, Induced Cluster Subgraph is solvable in polynomial time.*

It is known that all interval graphs, permutation graphs, distance-hereditary graphs, and convex graphs have mim-width at most 1 and their rooted layout of mim-width at most 1 can be obtained in polynomial time [1, 20]. Moreover, by Proposition 3, rooted layouts with mim-width at most 1 for the complement of these graphs can also be obtained in polynomial time. Thus, our algorithm directly indicates the following corollary.

▶ **Corollary 10.** *There is an algorithm that solves Induced Cluster Subgraph in polynomial time for interval graphs, permutation graphs, distance-hereditary graphs, convex graphs, and their complements.*

Here we give an idea of our algorithm. For a rooted layout (T, L) of a given graph with mim-width at most 1, we compute an optimal solution by means of dynamic programming from the leaves to the root of T. To complete the computation in polynomial time, for each node t of T, we discard redundant partial solutions and store essential ones of polynomial size. This approach was also employed in the previous algorithmic work of mim-width [2, 3, 4, 9, 21, 23, 24]. Especially, an equivalence relation called the *d-neighbor equivalence* plays a key role in compressing partial solutions and designing XP algorithms parameterized by mim-width [2, 3, 4, 9, 21]. However, Theorem 8 suggests that the d-neighbor equivalence does not work for designing an algorithm for Induced Cluster Subgraph; otherwise, we would obtain an XP algorithm parameterized by mim-width, which is quite unlikely by Theorem 8. A rooted layout with mim-width at most 1 resolves the difficulty. Recall that $\mathsf{mimw}(T, L) \leq 1$ if and only if $G[A_1^e, A_2^e]$ for any edge e of T is a chain graph as in Proposition 2. This property allows us to give strict total orderings of vertices in A_1^e and A_2^e with respect to neighbors of vertices. We define new equivalence relations over the strict total orderings, which enables the dynamic programming to run in polynomial time.

Let G be a graph and $<_A$ be a strict total order on $A \subseteq V(G)$. For a vertex subset $C \subseteq A$, we denote by $\mathsf{head}(C, <_A)$ and $\mathsf{tail}(C, <_A)$ the largest and smallest vertices in C with respect to $<_A$, respectively. More precisely, for a vertex $u \in C$, $u = \mathsf{head}(C, <_A)$ if and only if $v <_A u$ for any vertex $v \in C \setminus \{u\}$, and $u = \mathsf{tail}(C, <_A)$ if and only if $u <_A w$ for any vertex $w \in C \setminus \{u\}$, respectively. (For the sake of convenience, we allow $C = \emptyset$ and in this case we let $\mathsf{head}(C, <_A) = \emptyset$ and $\mathsf{tail}(C, <_A) = \emptyset$.) For a subset $S \subseteq A$, a partition $(C_1, C_2, \ldots, C_p)$ of S into p disjoint subsets $C_1, C_2, \ldots, C_p$ is called a *component partition of* S *over* G if for every $i \in [p]$, the subgraph of G induced by C_i is a connected component of $G[S]$. We say that a partition $(C_1, C_2, \ldots, C_p)$ of $S \subseteq A$ is *indexed by* $<_A$ if it holds that $\mathsf{head}(C_j, <_A) <_A \mathsf{head}(C_i, <_A)$ for any pair of integers i, j with $1 \leq i < j \leq p$.

For a non-empty vertex subset A of G such that $\mathsf{mim}(A) \leq 1$, a strict total order $<_A$ of A is called a *chain order* if for any two distinct vertices v, w in A, $v <_A w$ means $N(G; v) \setminus A \subseteq N(G; w) \setminus A$. (If $|A| = 1$, we define that the trivial strict total order of A is also a chain order.) By Proposition 2 and the definition of chain graphs, there is a chain order of A if and only if $\mathsf{mim}(A) \leq 1$. Note that $\mathsf{mim}(\overline{A}) \leq 1$ also holds and thus there is a chain order $<_{\overline{A}}$ of $\overline{A}$.

For subsets S_A and S'_A of A, let $(C_1, C_2, \ldots, C_p)$ denote the component partition of S_A over G and let $(C'_1, C'_2, \ldots, C'_q)$ denote the component partition of S'_A over G, where p and q are positive integers and both the component partitions are indexed by a chain order

$<_A$. We write $S_A \equiv^{\mathsf{cl}}_{G,<_A} S'_A$ if $\mathsf{head}(C_1, <_A) = \mathsf{head}(C'_1, <_A)$, $\mathsf{tail}(C_1, <_A) = \mathsf{tail}(C'_1, <_A)$, and $\mathsf{head}(C_2, <_A) = \mathsf{head}(C'_2, <_A)$. If the graph G and the chain order $<_A$ involved in the component partitions of S_A and S'_A are clear from the context, then we use the shorthand $\equiv^{\mathsf{cl}}_A$. It is not hard to see that $\equiv^{\mathsf{cl}}_A$ is an equivalence relation over subsets of A. A *representative* of S_A, denoted by $\mathsf{rep}_A(S_A)$, is the set $R = \{\mathsf{head}(C_1, <_A), \mathsf{tail}(C_1, <_A), \mathsf{head}(C_2, <_A)\}$. In the same way, we define an equivalence relation $\equiv^{\mathsf{cl}}_{<_{\overline{A}}}$ over subsets of $\overline{A}$ according to a chain order $<_{\overline{A}}$ and a representative $\mathsf{rep}_{\overline{A}}(S_{\overline{A}})$ of $S_{\overline{A}} \subseteq \overline{A}$.

Consider two subsets $S_A, S'_A \subseteq A$ with $|S_A| \geq |S'_A|$. Assume that for any subset $S_{\overline{A}} \subseteq \overline{A}$, $S_A \cup S_{\overline{A}}$ is a cluster set of G if and only if $S'_A \cup S_{\overline{A}}$ is a cluster set of G. This suggests that there is no need to store S'_A during dynamic programming over T. Formally, we give the following lemma.

▶ **Lemma 11** (♠). *For a vertex subset A of a graph G such that $\mathsf{mim}(A) \leq 1$, let $S_A, S'_A \subseteq A$ be cluster sets of G with $S_A \equiv^{\mathsf{cl}}_A S'_A$ and let $S_{\overline{A}}$ be any subset of $\overline{A}$. Then, $S_A \cup S_{\overline{A}}$ is a cluster set of G if and only if $S'_A \cup S_{\overline{A}}$ is a cluster set of G.*

Lemma 11 asserts that the equivalence relation $\equiv^{\mathsf{cl}}_A$ allows us to determine vertex sets to be stored. However, Lemma 11 is not enough to construct a dynamic programming algorithm. If a chain order is arbitrarily given for each node t of T, then the ordering of the stored sets may change, which causes the algorithm to output an incorrect solution. To avoid the inconsistency, we need to define chain orders with additional constraints.

Let (T, L) be a rooted layout of a graph $G = (V, E)$. For a node t of T, we denote by T_t the subtree of T rooted at t. We define $V_t = \{L^{-1}(\ell) \mid \ell \text{ is a leaf of } T_t\}$, $\overline{V_t} = V \setminus V_t$, $G_t = G[V_t]$, and $G_{\overline{t}} = G[\overline{V_t}]$. We use the shorthand notations $G_{t,\overline{t}}$ for the bipartite subgraph $G[V_t, \overline{V_t}]$ and rep_t for the representative rep_{V_t}. We define a strict total order $<_t$ on vertices in V_t, called a *lower chain order*, that satisfies the two conditions below:

(ℓ-1) $<_t$ is a chain order of V_t; and

(ℓ-2) if t has a child c, then for any pair of distinct vertices v, w in V_c, it holds that $v <_c w$ if and only if $v <_t w$.

We also define an *upper chain order* $<_{\overline{t}}$ as a strict total order on vertices in $\overline{V_t}$ that holds the following three conditions:

(u-1) $<_{\overline{t}}$ is a chain order of $\overline{V_t}$;

(u-2) if t has a child c, then for any pair of distinct vertices v, w in $\overline{V_t}$, it holds that $v <_{\overline{t}} w$ if and only if $v <_{\overline{c}} w$; and

(u-3) if t has the parent p, then for any pair of distinct vertices v, w in $\overline{V_t} \cap V_p$, it holds that $v <_{\overline{t}} w$ if and only if $v <_p w$, where $<_p$ is a lower chain order on V_p.

Lemma 12 asserts that the above strict total orders can be found in polynomial time.

▶ **Lemma 12** (♠). *Let (T, L) be a rooted layout of a graph G with $\mathsf{mimw}(T, L) \leq 1$. For every node t of T, a lower chain order $<_t$ and an upper chain order $<_{\overline{t}}$ exist and can be obtained in polynomial time.*

We here give the following two lemmas, which are keys to show the correctness of our algorithm given later.

▶ **Lemma 13** (♠). *Let (T, L) be a rooted layout of a graph G with $\mathsf{mimw}(T, L) \leq 1$ and t be an internal node of T with a child c. For any subset $S \subseteq \overline{V_c} \cap V_t$ of G, it holds that $\mathsf{rep}_{\overline{c}}(S) = \mathsf{rep}_t(S)$.*

▶ **Lemma 14 (♠).** *Let (T, L) be a rooted layout of a graph G with $\mathsf{mimw}(T, L) \leq 1$ and let t be an internal node of T with children a and b. For disjoint cluster sets $X \subseteq V_a$ and $Y \subseteq V_b$, if $X \cup Y$ is a cluster set of G, then $\mathsf{rep}_t(X \cup Y) = \mathsf{rep}_t(\mathsf{rep}_t(X) \cup \mathsf{rep}_t(Y))$ holds. Moreover, for a cluster set $Z \subseteq \overline{V_t}$ of G, if $X \cup Z$ (resp. $Y \cup Z$) is a cluster set of G, then $\mathsf{rep}_{\overline{b}}(X \cup Z) = \mathsf{rep}_{\overline{b}}(\mathsf{rep}_{\overline{b}}(X) \cup \mathsf{rep}_{\overline{b}}(Z))$ (resp. $\mathsf{rep}_{\overline{a}}(Y \cup Z) = \mathsf{rep}_{\overline{a}}(\mathsf{rep}_{\overline{a}}(Y) \cup \mathsf{rep}_{\overline{a}}(Z))$) holds.*

We now provide a polynomial-time algorithm for Induced Cluster Subgraph. Suppose that (T, L) is a rooted layout of a graph G with $\mathsf{mimw}(T, L) \leq 1$ and t is a node of T. We let $\mathscr{R}_t = \{\mathsf{rep}_t(S_t) : S_t \subseteq V_t\}$ and $\mathscr{R}_{\overline{t}} = \{\mathsf{rep}_{\overline{t}}(S_{\overline{t}}) : S_{\overline{t}} \subseteq \overline{V_t}\}$. For two sets $R_t \in \mathscr{R}_t$ and $R_{\overline{t}} \in \mathscr{R}_{\overline{t}}$, we define $f_t(R_t, R_{\overline{t}})$ as the function that returns the largest size of a subset $S_t \subseteq V_t$ such that

1. $\mathsf{rep}_t(S_t) = R_t$; and
2. $S_t \cup R_{\overline{t}}$ is a cluster set of G.

We let $f_t(R_t, R_{\overline{t}}) = -\infty$ if there is no subset satisfying the above conditions. For each triple of $t \in V(T)$, $R_t \in \mathscr{R}_t$, and $R_{\overline{t}} \in \mathscr{R}_{\overline{t}}$, we compute $f_t(R_t, R_{\overline{t}})$ by means of dynamic programming from the leaves to the root r of T. As $G = G_r$, we obtain the maximum size of cluster sets of G by computing $\min\{f_r(R_r, \emptyset) : R_r \in \mathscr{R}_r\}$. Notice that, for simplicity, our algorithm computes the size of an optimal solution. One can easily modify our algorithm so that it finds the largest cluster set in the same time complexity.

The case where t is a leaf of T. Denote by v the unique vertex in V_t. Then, $\mathscr{R}_t = \{\emptyset, \{v\}\}$. If $R_t = \emptyset$, only $S_t = \emptyset$ satisfies the prescribed conditions for any $R_{\overline{t}} \in \mathscr{R}_{\overline{t}}$. If $R_t = \{v\}$, then $S_t = \{v\}$ and we have to check that $\{v\} \cup R_{\overline{t}}$ is a cluster set of G. In summary, we have

$$f_t(R_t, R_{\overline{t}}) = \begin{cases} 0 & \text{if } R_t = \emptyset \text{ and } R_{\overline{t}} \text{ is a cluster set of } G, \\ 1 & \text{if } R_t = \{v\} \text{ and } \{v\} \cup R_{\overline{t}} \text{ is a cluster set of } G, \\ -\infty & \text{otherwise.} \end{cases}$$

The case where t is an internal node of T. Suppose that t has children a and b, and $f_a(R_a, R_{\overline{a}})$ and $f_b(R_b, R_{\overline{b}})$ have already been computed for any $R_a \in \mathscr{R}_a$, $R_{\overline{a}} \in \mathscr{R}_{\overline{a}}$, $R_b \in \mathscr{R}_b$, and $R_{\overline{b}} \in \mathscr{R}_{\overline{b}}$. For the largest subset $S_t \subseteq V_t$ that satisfies the prescribed conditions, S_t can be partitioned into two cluster sets $S_t \cap V_a$ and $S_t \cap V_b$. In addition, $(S_t \cap V_b) \cup R_{\overline{t}}$ and $(S_t \cap V_a) \cup R_{\overline{t}}$ form cluster sets of $G[V_{\overline{a}}]$ and $G[V_{\overline{b}}]$, respectively. We guess that $\mathsf{rep}_t(S_t \cap V_a) = \mathsf{rep}_a(S_t \cap V_a) = R_a \in \mathscr{R}_a$ and $\mathsf{rep}_t(S_t \cap V_b) = \mathsf{rep}_b(S_t \cap V_b) = R_b \in \mathscr{R}_b$. By Lemma 14, R_t can be represented as follows:

$$\begin{aligned} R_t &= \mathsf{rep}_t(S_t) \\ &= \mathsf{rep}_t((S_t \cap V_a) \cup (S_t \cap V_b)) \\ &= \mathsf{rep}_t(\mathsf{rep}_t(S_t \cap V_a) \cup \mathsf{rep}_t(S_t \cap V_b)) \\ &= \mathsf{rep}_t(R_a \cup R_b). \end{aligned}$$

To obtain the value $f_t(R_t, R_{\overline{t}})$, we calculate the sum of $f_a(R_a, \mathsf{rep}_{\overline{a}}((S_t \cap V_b) \cup R_{\overline{t}}))$ and $f_b(R_b, \mathsf{rep}_{\overline{b}}((S_t \cap V_a) \cup R_{\overline{t}}))$ for each pair (R_a, R_b) such that $R_a \in \mathscr{R}_a$, $R_b \in \mathscr{R}_b$, and $R_t = \mathsf{rep}_t(R_a \cup R_b)$. Combining Lemmas 13 and 14 with $\mathsf{rep}_{\overline{a}}(R_{\overline{t}}) = \mathsf{rep}_{\overline{t}}(R_{\overline{t}})$, which is observed from the condition (u-2) for an upper chain order, it holds that

$$\begin{aligned} \mathsf{rep}_{\overline{a}}((S_t \cap V_b) \cup R_{\overline{t}}) &= \mathsf{rep}_{\overline{a}}(\mathsf{rep}_{\overline{a}}(S_t \cap V_b) \cup \mathsf{rep}_{\overline{a}}(R_{\overline{t}})) \\ &= \mathsf{rep}_{\overline{a}}(\mathsf{rep}_t(S_t \cap V_b) \cup \mathsf{rep}_{\overline{t}}(R_{\overline{t}})) \\ &= \mathsf{rep}_{\overline{a}}(R_b \cup R_{\overline{t}}). \end{aligned}$$

Similarly, we have $\mathsf{rep}_{\overline{b}}((S_t \cap V_a) \cup R_{\overline{t}}) = \mathsf{rep}_{\overline{b}}(R_a \cup R_{\overline{t}})$. We conclude that

$$f_t(R_t, R_{\overline{t}}) = \max_{R_a \in \mathscr{R}_a \wedge R_b \in \mathscr{R}_b} \{f_a(R_a, \mathsf{rep}_{\overline{a}}(R_b \cup R_{\overline{t}})) \\ + f_b(R_b, \mathsf{rep}_{\overline{b}}(R_a \cup R_{\overline{t}})) : R_t = \mathsf{rep}_t(R_a \cup R_b)\}.$$

Since $\mathscr{R}_t$ and $\mathscr{R}_{\overline{t}}$ are of polynomial size for every t of T, our algorithm runs in polynomial time. This completes the proof of Theorem 9.

We can extend the above algorithm to other several problems. (For more details, see the full version of this paper.) Combined with Theorem 8, we obtain the following dichotomy theorem.

▶ **Theorem 15** (♠). *All the following problems, as well as their connected variants and their dominating variants, are NP-hard for graphs with mim-width at most* 2*: (i)* Clique*; (ii)* Induced Cluster Subgraph*; (iii)* Induced Polar Subgraph*; (iv)* Induced $\overline{P_3}$-free Subgraph*; (v)* Induced Split Subgraph*; and (vi)* Induced $\overline{K_3}$-free Subgraph*. On the other hand, given a graph and its branch decomposition of mim-width at most* 1*, all the above problems, as well as their connected variants and their dominating variants, are solvable in polynomial time.*

5 Concluding remarks

We discuss future work here. Our proof of Theorem 5 relies on the assumption that all cliques satisfy a fixed property Π, and hence Theorem 5 is not applicable to Induced Π Subgraph such that Π excludes some clique. Such problems include Independent Set, Induced Matching, Longest Induced Path, and Feedback Vertex Set. In fact, there exist XP algorithms of the problems listed above when parameterized by mim-width [1, 9, 23, 24]. This motivates us to seek Π such that Induced Π Subgraph is NP-hard for bounded mim-width graphs although Π excludes some clique. As the first step, it would be interesting to consider Induced K_3-free Subgraph.

In [2], Bergougnoux et al. showed that Clique is expressible in A&C DN + ∀, which is A&C DN logic that allows to use a single universal quantifier ∀, and hence their meta-theorem cannot be extended to A&C DN + ∀. Our results in this paper suggest that the barrier could be broken down for graphs with mim-width at most 1. The next goal is to obtain a more general logic than A&C DN such that all problems expressible in the logic are solvable in polynomial time for graphs with mim-width at most 1.

Finally, we end this paper by leaving the biggest open problem concerning mim-width: Given a graph G, is there a polynomial-time algorithm that computes a branch decomposition with mim-width 1, or concludes that G has mim-width more than 1?

References

1 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

2 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. A logic-based algorithmic meta-theorem for mim-width. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 3282–3304, 2023. doi:10.1137/1.9781611977554.ch125.

3 Benjamin Bergougnoux and M. Moustapha Kanté. More applications of the *d*-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM Journal on Discrete Mathematics*, 35(3):1881–1926, 2021. doi:10.1137/20m1350571.

4 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Algorithmica*, 84(5):1385–1417, 2022. doi:10.1007/s00453-022-00936-w.

5 Flavia Bonomo-Braberman, Nick Brettell, Andrea Munaro, and Daniël Paulusma. Solving problems on generalized convex graphs via mim-width. *J. Comput. Syst. Sci.*, 140:103493, 2024. doi:10.1016/J.JCSS.2023.103493.

6 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.143.

7 Nick Brettell, Jake Horsfield, Andrea Munaro, Giacomo Paesani, and Daniël Paulusma. Bounding the mim-width of hereditary graph classes. *Journal of Graph Theory*, 99(1):117–151, 2022. doi:10.1002/jgt.22730.

8 Nick Brettell, Jake Horsfield, Andrea Munaro, and Daniël Paulusma. List k-colouring P_t-free graphs: A mim-width perspective. *Information Processing Letters*, 173:106168, 2022. doi:10.1016/j.ipl.2021.106168.

9 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.

10 Yixin Cao, Yuping Ke, Yota Otachi, and Jie You. Vertex deletion problems on chordal graphs. *Theoretical Computer Science*, 745:75–86, 2018. doi:10.1016/j.tcs.2018.05.039.

11 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006. doi:10.4007/annals.2006.164.51.

12 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

13 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

14 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 82(9):2432–2473, 2020. doi:10.1007/s00453-020-00692-9.

15 Esther Galby, Paloma T. Lima, and Bernard Ries. Reducing the domination number of graphs via edge contractions and vertex deletions. *Discrete Mathematics*, 344(1):112169, 2021. doi:10.1016/j.disc.2020.112169.

16 Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *Theoretical Computer Science*, 814:28–48, 2020. doi:10.1016/j.tcs.2020.01.007.

17 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

18 Peter L. Hammer, Uri N. Peled, and Xiaorong Sun. Difference graphs. *Discrete Applied Mathematics*, 28(1):35–44, 1990. doi:10.1016/0166-218X(90)90092-Q.

19 Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. On the d-claw vertex deletion problem. *Algorithmica*, 86(2):505–525, 2024. doi:10.1007/S00453-023-01144-W.

20 Sang il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005. doi:10.1016/j.jctb.2005.03.003.

21 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019. doi:10.1016/j.tcs.2019.09.012.

22 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2018.42.

23 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, 2020. doi:10.1016/j.dam.2019.06.026.

24 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 82(1):118–145, 2020. doi:10.1007/s00453-019-00607-3.

25 Lars Jaffke, Paloma T. Lima, and Roohani Sharma. Structural parameterizations of b-coloring. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2023.40.

26 Dong Yeap Kang, O-joung Kwon, Torstein J.F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theoretical Computer Science*, 704:1–17, 2017. doi:10.1016/j.tcs.2017.09.006.

27 Hoang-Oanh Le and Van Bang Le. Complexity of the cluster vertex deletion problem on H-free graphs. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.MFCS.2022.68.

28 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

29 Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 248:28–32, 2018. doi:10.1016/j.dam.2017.04.043.

30 Andrea Munaro and Shizhou Yang. On algorithmic applications of sim-width and mim-width of (H_1, H_2)-free graphs. *Theoretical Computer Science*, 955:113825, 2023. doi:10.1016/j.tcs.2023.113825.

31 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.

32 Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theoretical Computer Science*, 615:120–125, 2016. doi:10.1016/j.tcs.2015.11.039.

33 Martin Vatshelle. *New Width Parameters of Graphs.* PhD thesis, University of Bergen, 2012.

34 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. doi:10.1137/0210022.

A Fast 3-Approximation for the Capacitated Tree Cover Problem with Edge Loads

Benjamin Rockel-Wolff
Research Institute for Discrete Mathematics, University of Bonn, Germany

Abstract

The capacitated tree cover problem with edge loads is a variant of the tree cover problem, where we are given facility opening costs, edge costs and loads, as well as vertex loads. We try to find a tree cover of minimum cost such that the total edge and vertex load of each tree does not exceed a given bound. We present an $\mathcal{O}(m \log n)$ time 3-approximation algorithm for this problem.

This is achieved by starting with a certain LP formulation. We give a combinatorial algorithm that solves the LP optimally in time $\mathcal{O}(m \log n)$. Then, we show that a linear time rounding and splitting technique leads to an integral solution that costs at most 3 times as much as the LP solution. Finally, we prove that the integrality gap of the LP is 3, which shows that we can not improve the rounding step in general.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Trees

Keywords and phrases Approximation Algorithms, Tree Cover, LP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.39

Related Version *Full Version*: `https://arxiv.org/abs/2404.10638` [11]

1 Introduction

Graph cover problems deal with the following base problem. Given a graph G, the task is to find a set of (connected) subgraphs of G, the cover, such that each vertex of G is contained in at least one of the subgraphs. Usually, the subgraphs are restricted to some class of graphs, like paths, cycles or trees. Different restrictions can be imposed on the subgraphs, like a maximum number of edges, or a total weight of the nodes for some given node weights. Recently, Schwartz [12] published an overview of the literature on different covering and partitioning problems.

We consider the capacitated tree cover problem with edge loads. It is a variation of the tree cover problem that has not been studied so far to the best of our knowledge.

In the capacitated tree cover problem with edge loads, we are given a complete graph $G = (V, E)$, metric edge costs $c : E \to \mathbb{R}_+$, vertex loads $b : V \to [0, 1)$, metric edge loads $u : E \to \mathbb{R}_{\geq 0}$ with $u(e) < u(f) \Rightarrow c(e) \leq c(f)$, and a facility opening cost $\gamma \geq 0$. The task is to find a number of components $k \in \mathbb{N}_{\geq 1}$ and a forest F in G consisting of k trees minimizing

$$\sum_{e \in E(F)} c(e) + \gamma k,$$

such that each tree T_i has total load

$$u(T_i) := \sum_{e \in E(T_i)} u(e) + \sum_{v \in V(T_i)} b(v) \leq 1.$$

For simplicity of presentation, we additionally require that $u > 0$. This is not necessary in general and the extended proofs for $u \geq 0$ are covered in the full paper [11].

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 39; pp. 39:1–39:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The capacitated tree cover problem with edge loads is closely related to the facility location problem with service capacities discussed by Maßberg and Vygen in [10]. Their problem uses Steiner trees to connect the nodes, not spanning trees. Furthermore, in their case edge cost and edge load are the same. They make use of this fact to prove a lower bound on the value of an optimum solution. Both problems have important practical applications in chip design. In [4] they are called the sink clustering problem and used for clock tree construction. In [2] they are used for repeater tree construction. In these applications terminals and edges have an electrical capacitance. A source can drive only a limited capacitance. Edge cost and capacitance usually are proportional to the length of an edge. As the edge length is given by the l_1-distance between its endpoints, this naturally matches our problem.

Our problem is also related to other facility location and clustering problems, like the (capacitated) k-center problem ([5, 8]) or the k-means problem ([6, 9]).

Other tree cover problems include the k-min-max tree cover problem and the bounded tree cover problem ([1, 3, 7]). In the k-min-max tree cover problem, we are given edge weights and want to find k trees such that the maximum of the total weights of the trees is minimized. In the bounded tree cover problem, we are given a bound on the maximum weight of a tree in the cover and try to minimize the number of trees that are required. For these problems Khani and Salavatipour [7] gave a 3- and 2.5-approximation respectively. They improve over the previously best known results by Arkin et al. [1], who presented a 4-approximation algorithm for the min-max tree cover problem and a 3-approximation algorithm for the bounded tree cover problem. Even et al. [3] independently gave a 4-approximation algorithm for the min-max tree cover problem. Furthermore, a rooted version of these problems has been studied. The best known approximation ratio for the capacitated tree case is 7 and was developed by Yu and Liu [15].

Many algorithms for cycle cover problems are also based on tree cover algorithms ([3, 13, 14]). An example is the capacitated cycle covering problem, where the cover consists of cycles (and singletons) and are given an upper bound on the total nodeweight of the cycles. The task is to minimize the total weight of the cycles plus the facility opening costs. Traub and Tröbst [13] presented a $2+\frac{2}{7}$-approximation for this problem. They use an algorithm for the capacitated tree cover problem as a basis for their $2+\frac{2}{7}$-approximation. In particular, they present a 2-approximation for the capacitated tree cover problem without edge loads.

2 Our contribution

In Section 3, we present an LP formulation of the capacitated tree cover problem with edge loads that is based on the formulation in [13].

Then, we will present a combinatorial algorithm that can optimally solve the LP in time $\mathcal{O}(m \log n)$ in Section 4, where n is the number of vertices and m is the number of edges of the graph.

Next, we show how to round the solution to an integral solution in Section 5, employing a splitting technique that runs in linear time from [10], and show that the resulting integral solution costs at most 3 times as much as the LP-solution. This proves our main theorem:

▶ **Theorem 1.** *There is a 3-approximation algorithm for the capacitated tree cover problem with edge loads that runs in time $\mathcal{O}(m \log n)$.*

While the overall approach is similar to the one used in [13], edges with load require a different algorithm for solving the LP. Furthermore, we need to be more careful in the analysis of our rounding step.

Finally, in Section 6, we will give an example proving that the integrality gap of our LP is at least 3.

3 The LP-formulation

We may assume that $\gamma \geq c(e)$ for all $e \in E$, as an edge with $c(e) > \gamma$ will never be used in an optimum solution (and could be removed from the solution of the algorithm without increasing the cost).

For simplicity, we will introduce some notation here: For any function $f : A \to B \subseteq \mathbb{R}$ from a finite set A into a set $B \subseteq \mathbb{R}$ and $X \subseteq A$ we write $f(X) := \sum_{x \in X} f(x)$.

Given a solution F to our problem with k components $\{T_1, \dots, T_k\}$, we know that each tree T_i contains exactly $|V(T_i)| - 1$ edges and hence $k = |V| - |E(F)|$. Each induced subgraph of F is a forest. So we know

$$|E(F[A])| \leq |A| - 1 \text{ for each } A \subseteq V.$$

Let us now consider the load on the subgraph of F, induced by $A \subseteq V$. Each connected component in $F[A]$ can have load at most 1. So there must be at least $b(A) + u(E(F[A]))$ components in $F[A]$. As each of the components is a tree, the inequality

$$|E(F[A])| \leq |A| - (b(A) + u(E(F[A])))$$

must be fulfilled. Using these properties, we can formulate the following LP relaxation of this problem:

$$\begin{aligned}
\min \quad & c^t x + \gamma(|V| - x(E)) && && (1)\\
\text{s.t.} \quad & x(E(G[A])) \leq |A| - 1 && \text{for each } A \subseteq V && (2)\\
& \sum_{e \in E(G[A])} (1 + u(e))x(e) \leq |A| - b(A) && \text{for each } A \subseteq V && (3)\\
& 0 \leq x(e) \leq 1 && \text{for each } e \in E && (4)
\end{aligned}$$

Here $x(e)$ denotes the fractional usage of the edge e. We will call an edge e *active* if $x(e) > 0$. The LP can be reformulated by using variables $y(e) := x(e)(1 + u(e))$:

$$\begin{aligned}
\min \quad & \sum_{e \in E} \frac{c(e)}{1 + u(e)} y(e) + \gamma \left(|V| - \sum_{e \in E} \frac{y(e)}{1 + u(e)} \right) && && (5)\\
\text{s.t.} \quad & \sum_{e \in E(A)} \frac{y(e)}{1 + u(e)} \leq |A| - 1 && \text{for each } A \subseteq V && (6)\\
& y(E(G[A])) \leq |A| - b(A) && \text{for each } A \subseteq V && (7)\\
& 0 \leq y(e) \leq 1 + u(e) && \text{for each } e \in E && (8)
\end{aligned}$$

For simplicity, we will always consider solutions x, y of both LPs at once. In the following, we will denote by $u_x(e) := x(e) \cdot u(e)$ the fractional load of edge e.

▶ **Definition 2.** *For a solution x, y to the LP, we define the* support graph $G_x := (V, \{e \in E \mid x(e) > 0\})$*, i.e. the graph consisting of the vertices V and all active edges.*

We call an edge tight *if $y(e) = 1 + u(e)$, and we call a set $A \subseteq V$ of vertices* tight *if inequality (7) is tight.*

Our goal will be to solve the LP exactly and then round to a forest that may violate the capacity constraints. This increases the edge cost by at most a factor of 2. In a final step each tree T in the forest with a load $b(V(T)) + u(E(T)) > 1$ can be split into at most $2 \cdot (b(V(T)) + u(E(T)))$ trees. This may decrease the edge cost, but loses a factor of 3 in the number of components, compared to the LP solution.

4 Solving the LP

Altough the LP has an exponential number of inequalities, we can solve it using a simple greedy algorithm, shown in Algorithm 1. We will focus on solving the second LP (5) – (8).

As a first step, we sort the edges $\{e_1, \dots, e_m\} = E(G)$ such that

$$\frac{c(e_1) - \gamma}{1 + u(e_1)} \leq \dots \leq \frac{c(e_m) - \gamma}{1 + u(e_m)}.$$

In each iteration, we compute a partition $\mathcal{A}_i \subset 2^{V(G)}$ of the vertices of G, based on the previous partition $\mathcal{A}_{i-1}$. We initialize y to 0 and start with $\mathcal{A}_0 := \{\{v\} | v \in V(G)\}$. Then we iterate through the edges from e_1 to e_m. For each edge e_i, we do the following:

If e_i has endpoints in two different sets of the partition $A_i^1, A_i^2 \in \mathcal{A}_{i-1}$, we increase $y(e_i)$ to the maximum possible value. This maximum value is the sum of the slacks of inequalities (7) for the sets A_i^1 and A_i^2: $|A_i^1| - b(A_i^1) - y(E(G[A_i^1])) + |A_i^2| - b(A_i^2) - y(E(G[A_i^2]))$. However, we assign at most $1 + u(e_i)$, such that we do not violate inequality (8). Finally, if we increased $y(e_i)$ by a positive amount, we create the new partition $\mathcal{A}_i$ that arises from $\mathcal{A}_{i-1}$ by removing A_i^1 and A_i^2 and adding their union.

We set $\mathcal{A} := \bigcup_{i=1,\dots,m} \mathcal{A}_i$. Observe that $\mathcal{A}$ is a laminar family. This guarantees that the support graph is always a forest and inequality (6) is automatically fulfilled.

Algorithm 1 Algorithm for solving the LP (5) – (8).

```
Input  : $G, c, u$.
Output : $y$ optimum solution of the LP (5) – (8).
Sort edges such that $\frac{c(e_1)-\gamma}{1+u(e_1)} \leq \dots \leq \frac{c(e_m)-\gamma}{1+u(e_m)}$;
Set $\mathcal{A}_0 := \{\{v\}|v \in V(G)\}$ and $y := 0$;
for $i = 1 \dots m$ do
    if there are sets $A_i^1, A_i^2 \in \mathcal{A}_{i-1}$ with $e_i \cap A_i^1 \neq \emptyset$, $e_i \cap A_i^2 \neq \emptyset$ and $A_i^1 \neq A_i^2$ then
        $y(e_i) := \min\{1 + u(e_i), |A_i^1| - b(A_i^1) - y(E(A_i^1)) + |A_i^2| - b(A_i^2) - y(E(A_i^2))\}$;
        if $y(e_i) > 0$ then
            $\mathcal{A}_i := (\mathcal{A}_{i-1} \setminus \{A_i^1, A_i^2\}) \cup \{A_i^1 \cup A_i^2\}$;
        else
            $\mathcal{A}_i := \mathcal{A}_{i-1}$
```

▶ **Lemma 3.** *Let x, y be the solution computed by Algorithm 1. If a set $A \in \mathcal{A}$ from the algorithm is not tight, then all the edges in its induced subgraph $G_x[A]$ of the support graph are tight.*

Proof. Assume this were false. Take a minimal counterexample A. As the claim certainly holds for sets consisting only of one vertex ($G_x[A]$ has no edges if $|A| = 1$), we know that $|A| \geq 2$. We can write $A = A_i^1 \cup A_i^2$ with their associated edge e_i (for some i). We know that e_i has to be tight by line 5, as A is not tight. Otherwise, the algorithm could have increased $y(e_i)$ further. At least one of the subsets A_i^1 and A_i^2 of A is not tight, otherwise, A were tight. W.l.o.g we may assume that A_i^1 is not tight. Then all of its edges are tight, by minimality of A. However, then we know that $x(E(G[A_i^1])) = |A_i^1| - 1$. Thus,

$$\begin{aligned}&|A_i^1| - b(A_i^1) - y(E(G[A_i^1])) = |A_i^1| - b(A_i^1) - x(E(G[A_i^1])) - u_x(E(G[A_i^1]))\\&= |A_i^1| - b(A_i^1) - (|A_i^1| - 1) - u_x(E(G[A_i^1])) = 1 - (u_x(E(G[A_i^1]) + b(A_i^1)) < 1 + u(e_i).\end{aligned}$$

This implies that A_i^1 does not have enough slack to make e_i tight. Thus A_i^2 cannot be tight. As A contains an edge that is not tight in its support graph, this edge must be contained in A_i^2. We can conclude that A_i^2 is a smaller counterexample. This contradicts the minimality of A. ◀

▶ **Corollary 4.** *Let $e_i \in E$, A_i^1 and A_i^2 fulfill the conditions in line 4 of Algorithm 1. If e_i is tight then neither A_i^1, nor A_i^2 are tight.*

▶ **Theorem 5.** *Algorithm 1 works correctly and has running time $\mathcal{O}(m \log n)$.*

Proof. The running time is dominated by sorting. Due to space constraints, we will only give the ideas of the correctness proof. The details are contained in the full version [11] of this paper.

We first check that the algorithm outputs a solution to our LP. The minimum in line 5 guarantees that inequality (8) is fulfilled. We have already seen that the support graph of our solution is a forest, which means that inequality (6) is also satisfied. It remains to check that inequality (7) holds. Each $A \in \mathcal{A}$ fulfills the inequality, when it is introduced by line 5. Since $\mathcal{A}$ is a laminar family, we never change the value of $y(E(G[A]))$ after A has been introduced, so we already know that all $A \in \mathcal{A}$ satisfy inequality (7) when the algorithm is finished.

We define the slack of a set $A \subseteq V$ as the slack of inequality 7 for that set and denote it by $\sigma(A) := |A| - b(A) - y(E(G[A]))$.

Then we can prove that when the algorithm introduces a new set A, it has no more slack than each of the joined subsets.

▷ Claim 6. Let $A_i^1, A_i^2, A \in \mathcal{A}$ such that $A = A_i^1 \cup A_i^2$. We claim that $\sigma(A) \leq \sigma(A_i^j)$ for $j = 1, 2$.

The idea to prove this, is to show that $\sigma(A) = \sigma(A_i^1) + \sigma(A_i^2) - y(e_i)$. Then we use Corollary 4 and Lemma 3 to derive that $\max\{\sigma(A_i^1), \sigma(A_i^2)\} \leq y(e_i)$, which proves the claim. As a next step, we extend Claim 6 to all subsets of A.

▷ Claim 7. Let $A \in \mathcal{A}$. We claim that each subset $B \subseteq A$ has slack $\sigma(B) \geq \sigma(A)$.

We prove this by induction on the number of iterations. The main idea here is to first split B into subsets B_1, B_2 of the two sets A_i^1, A_i^2 that have been merged to form A. We show that $\sigma(B) \geq \sigma(B_1) + \sigma(B_2) - y(e_i)$. Then we apply our induction hypothesis to both sets and use the equality from Claim 6 to prove Claim 7.

Finally, we observe that for $B_1 \subseteq V$ and $B_2 \subseteq V$ from different connected components of G_x, we have $\sigma(B_1 \cup B_2) = \sigma(B_1) + \sigma(B_2)$. This means that we can split any subset of the vertices $B \subseteq V$ into subsets of the toplevel sets of $\mathcal{A}$, which are exactly the connected components of G_x. Then we can use Claim 7 on each of the subsets to see that they have nonnegative slack. The observation implies that also B has nonnegative slack. So inequality 7 is always satisfied.

Next we want to prove optimality. Assume that y were not optimum. Let y^* be an optimum solution that fulfills the following: It maximizes the index of the first edge in the order of the algorithm in which y and y^* differ. Among those it minimizes the difference in this edge. Let this index be denoted by k. As the algorithm always sets the values to the maximum that is possible without violating an inequality, we know that $y^*(e_k) < y(e_k)$.

By the ordering of the algorithm, we know that

$$\frac{c(e_k) - \gamma}{1 + u(e_k)} \leq \frac{c(e_i) - \gamma}{1 + u(e_i)}$$

for all $i > k$. Our goal will be, to find an edge e_i with $i > k$ such that we can increase $y^*(e_k)$ and avoid violating constraints or increasing the objective by decreasing $y^*(e_i)$ in x^*, y^*.

Let G_k be the connected component of e_k in the subgraph of G that contains only e_k and the active edges with index less than k. Define

$$\Gamma := \{e_i \in E(G_{x^*}) \mid i > k \text{ and } e_i \text{ incident to } v \in V(G_k)\}.$$

Note that $\Gamma \neq \emptyset$, because otherwise, we could increase $y^*(e_k)$ to $y(e_k)$ without violating any constraints. Since $c(e) \leq \gamma$, this would not increase the objective value.

We will prove that all tight sets containing the vertices of G_k must have a common edge in Γ.

▷ Claim 8. Let $\mathcal{T} := \{B \subseteq V \mid V(G_k) \subseteq B \text{ and } B \text{ tight}\}$ be the family of tight sets of x^*, y^* containing the vertices of G_k. We claim that

$$\Gamma \cap \bigcap_{B \in \mathcal{T}} E(G_{x^*}[B]) \neq \emptyset.$$

If there is an edge in Γ between vertices of G_k, then this certainly holds. Otherwise, we know that $V(G_k)$ is not tight, because the algorithm was able to set $y(e_k) > y^*(e_k)$.

Let $\mathcal{S} := \{S_1, \ldots, S_p\} \subseteq \mathcal{T}$ be a set of $p \geq 2$ tight sets containing the vertices of G_k and set $Z := \bigcup_{S_i \in \mathcal{S}} S_i$. First, we observe that Z is tight as well. Then, we will prove our claim by induction on p. The key idea is to use the tightness of Z and the S_i to decompose $y^*(E(G[Z]))$ into an alternating sum of y^* on intersections of the S_i and then reassemble $y^*(E(G[Z]))$ from the parts (see [11]). Then, we see that there must be slack on the cut defined by G_k in the intersection of all S_i. Thus there must be an edge in the cut, which proves our claim.

Finally, we can pick an edge $f \in \Gamma$ that is contained in all tight sets that contain the vertices of G_k. If $u(f) \leq u(e_k)$, we know that $\frac{1}{1+u(f)} \geq \frac{1}{1+u(e_k)}$. So we can decrease $y^*(f)$ and increase $y^*(e_k)$ by the same amount without violating any constraints. By the ordering of our algorithm, this can not increase the objective value. This would contradict our choice of y^*. Hence $u(f) > u(e_k)$. But then $c(f) \geq c(e_k)$ and we could decrease $x^*(f)$ and increase $x^*(e_k)$ without increasing the objective value. Furthermore, we also do not create a violation this way, because $\epsilon \cdot (1 + u(f)) > \epsilon \cdot (1 + u(e_k))$ for $\epsilon > 0$. This contradicts our choice of y^* and concludes the proof. ◀

The support graph of the LP solution computed by Algorithm 1 is always a forest. Thus, Theorem 5 implies the following:

▶ **Corollary 9.** *There is always a solution x, y to both LPs, such that the support graph G_x is a forest.*

5 The Rounding Strategy

Now we want to round the LP solution, computed by Algorithm 1, to get an integral solution. We do so by rounding up edges e with $x(e) \geq \alpha$, for some $0 \leq \alpha \leq 1$ to be determined later. All other edges are rounded down. The forest arising from this rounding step may contain components T with $b(V(T)) + u(E(T)) > 1$. These large components will be split into at most $2 \cdot (b(V(T)) + u(E(T)))$ legal components. We achieve this by using a splitting technique that is often used for these cases, for example in [10] and also in [13]. During splitting, we need to shortcut some paths. This is possible, because G is a complete graph and u and c are metric.

The splitting technique traverses the trees in a bottom up fashion (for an arbitrary root). At each node, it approximately solves a bin packing problem to split off components that are too heavy. However, for the analysis, we only require the result that it is possible to split the trees into $2 \cdot (b(V(T)) + u(E(T)))$ legal components.

▶ **Lemma 10** (Maßberg and Vygen 2008 [10]). *If G is a complete graph and u and c are metric, there is a linear time algorithm that splits a tree with total load $b(V(T))+u(E(T)) > 1$ into at most $2 \cdot (b(V(T)) + u(E(T)))$ legal trees and does not increase the total edge cost.*

In Section 5.1, we will study the LP solution, that we get from Algorithm 1. We will exploit the structure of this solution in our analysis. Then we will bound the number of components that we get after rounding and splitting in Section 5. We do this by providing an upper bound on the value of each edge after rounding and splitting. Finally, in Section 5.2.4, we show that $\alpha := \frac{2}{3}$ will lead to an approximation factor of 3 independent of the edge loads.

5.1 The general structure of the LP solution

Let x, y be a solution found by the algorithm. Recall that for edges $e \in E(G)$, $u_x(e) := x(e) \cdot u(e)$ was the fractional load of the edge e in our solution. Note that then it holds for each set $A \subseteq V(G)$ and edge $e \in E(G)$ that

$$y(E(G[A])) = x(E(G[A])) + u_x(E(G[A])) \text{ and } y(e) = x(e) + x(e)u(e).$$

Without loss of generality, we can assume that $0 < x(e) < 1$ for all edges and G_x does not contain singletons. We simply remove all edges with $x(e) = 0$. Then we contract all inclusionwise maximal sets $A \in \mathcal{A}$ such that all edges in their respective induced support graph are tight and set the load of the new vertex to $b(A) + u_x(E(G[A]))$. Corollary 4 implies that we contracted all tight edges this way. These operations only make the approximation guarantee worse, because these components will have the same value in the rounded solution as in the LP-solution. In the remaining graph the following assertions hold:

1. $|\{v\}| - b(\{v\}) - y(E(G[\{v\}])) = 1 - b(v) \leq 1$ for all $v \in V$.
2. All $A \in \mathcal{A}$ containing more than 1 vertex are tight, by Lemma 3.

Now, we will take a closer look at the sets A_i^j for $i = 1, \ldots, m$ and $j = 1, 2$. In the following analysis, we will assume without loss of generality that $|A_i^1| \leq |A_i^2|$. By the above assertions, we have for an edge e_i and the two associated sets A_i^1, A_i^2, either

(i) both A_i^1 and A_i^2 contain only one vertex and one of them is not tight, or
(ii) A_i^1 contains only one vertex and is not tight and A_i^2 contains more vertices and is tight

To make the following easier to read, we add the following definitions

▶ **Definition 11.** *Edges that fulfill condition (i) are called* seed edges *and edges that fulfill condition (ii) are called* extension edges. *For each edge e_i we denote by v_{e_i} the unique vertex in set A_i^1.*

Note that every edge e_i is either a seed edge or an extension edge, but this only holds after contracting the sets of tight edges as described above.

Thus, whenever e_i is a seed edge, the algorithm sets

$$y(e_i) := |A_i^1| - b(A_i^1) - y(E(G[A_i^1])) + |A_i^2| - b(A_i^2) - y(E(G[A_i^2])) = 1 - b(A_i^1) + 1 - b(A_i^2),$$

where we use the fact that $E(G[A_i^j]) = \emptyset$ for $j = 1, 2$. Since both A_i^1 and A_i^2 were singletons, we can conclude

$$x(e_i) + u(e_i)x(e_i) = y(e_i) = 2 - b(A_i^1 \cup A_i^2).$$

Similarly, for extension edges, we get

$$x(e_i) + u(e_i)x(e_i) = 1 - b(A_i^1).$$

In the analysis of the rounding step, we need some further observations:

▶ **Observation 12.** *Let T be a connected component in G_x. Then*

- *T is a tree.*
- *If $|V(T)| > 1$, then T contains exactly one seed edge and all other edges are extension edges.*
- *If T contains a seed edge e_i, then $i = \min_{e_j \in E(T)} j$ or in other words, e_i was the first edge of T considered in the algorithm.*

A proof of these observations is not difficult, but deferred to the full version of this paper for space reasons.

5.2 Analyzing the rounding step

First note that by our rounding procedure, the sum of the edge-weights can increase by at most $\frac{1}{\alpha}$. So for the edge-weights it is sufficient to make sure that $\alpha \geq \frac{1}{2}$ and the main difficulty is to bound the number of components.

Before we choose α, let us estimate how many components we get after rounding and splitting. To do this, we take a look at the connected components after rounding. Let T be such a component. We denote by $\mathrm{comp}(T)$ the number of connected components we need to split T into.

Let C^* be the set of components before splitting and C be the set of components after splitting. Our goal here is to estimate $|C|$ by a contribution $\mathrm{est}(e)$ of each edge $e \in E(G)$, such that the number of components after splitting is

$$|C| = \sum_{T \in C^*} \mathrm{comp}(T) \leq \sum_{T \in C^*} |V(T)| - \sum_{e \in E(G)} \mathrm{est}(e) = |V(G)| - \sum_{e \in E(G)} \mathrm{est}(e)$$

There are three cases:

1. *singletons*: T consists of only one vertex.
2. *good trees*: T consists of more than one vertex and $u(E(T)) + b(V(T)) \leq 1$
3. *large trees*: T consists of more than one vertex and $u(E(T)) + b(V(T)) > 1$

Case 1. T is a singleton. Its number of components is

$$\mathrm{comp}(T) := 1 = |V(T)| - 0.$$

Case 2. T is a good tree. So we can keep this component for a solution to the problem. The number of components is

$$\mathrm{comp}(T) := 1 = |V(T)| - (|V(T)| - 1) \leq |V(T)| - \sum_{e \in E(T)} [1 - 2b(v_e) - 2u(e)] \,.$$

For all $e \in E(T)$, we set $\mathrm{est}(e) := 1 - 2b(v_e) - 2u(e)$.

Case 3. T is a large tree. So we have to split this component to get a feasible solution. Denote by e' the edge in T with the lowest index according to the sorting of the algorithm. Let $\bar{v} \neq v_{e'}$ be incident to e'. Note that this does not have to be a seed edge, as the components after rounding do not necessarily contain a seed edge. We rewrite the number of components:

$$\begin{aligned}\mathrm{comp}(T) \leq 2 \cdot (u(E(T)) + b(V(T))) = \quad & |V(T)| - [2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})] \\ & - \sum_{e' \neq e \in E(T)} [1 - 2b(v_e) - 2u(e)] \,.\end{aligned}$$

If T contains a seed edge, then this edge is e'. This means that the number of components can be estimated by edges in T. We set $\mathrm{est}(e') := 2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})$ and $\mathrm{est}(e) := 1 - 2b(v_e) - 2u(e)$ for all $e \in E(T) \setminus \{e'\}$.

Otherwise T only consists of extension edges. In this case, we write

$$\begin{aligned}&[2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})] + \sum_{e' \neq e \in E(T)} [1 - 2b(v_e) - 2u(e)] \\ &\qquad\qquad = [1 - 2b(\bar{v})] + \sum_{e \in E(T)} [1 - 2b(v_e) - 2u(e)] \,.\end{aligned}$$

Then, we set $\mathrm{est}(e) := 1 - 2b(v_e) - 2u(e)$ for all $e \in E(T)$. However, in this case we need to account for the additional $1 - 2b(\bar{v})$. To do so, we call the edge incident to $\bar{v}$ that is not contained in T a *filler edge*. For all filler edges $\{v, w\} = e \in E(G)$, we w.l.o.g. assume that e is a filler edge for the component that contains v and set

$$\mathrm{est}(e) := \begin{cases} 2 - (b(v) + b(w)), & \text{if } e \text{ is the filler edge of two components} \\ 1 - b(v), & \text{otherwise.} \end{cases}$$

For all edges not considered before, we set $\mathrm{est}(e) := 0$.

Now we have that

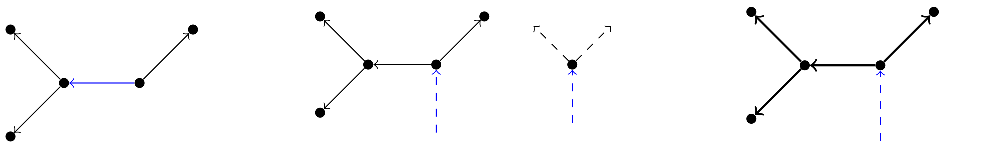
$$|C| \leq |V(G)| - \sum_{e \in E(G)} \mathrm{est}(e)$$

Our next goal is to find a lower bound on $\sum_{e \in E(G)} \mathrm{est}(e)$. Here we will leave out most of the computations, due to space restrictions, but they are contained in the full version [11] of this paper.

5.2.1 Lower bounds for the extension edges

We start with the simpler case of extension edges. An overview over the cases in which they can appear is shown in Figure 1. Let e be an extension edge. If it appears inside a good tree or a large tree. Then

$$\mathrm{est}(e) = 2x(e) - 1 - 2u(e)(1 - x(e)).$$



(a) Inside a (good or large) tree.

(b) Leading towards a good tree or a singleton.

(c) As a filler edge leading to a large tree.

Figure 1 The cases in which extension edges can occur. Dashed edges have been rounded down, while solid ones have been rounded up. Thick edges belong to a large tree. For each edge e the arrowhead points towards v_e.

If it is incident to a singleton or a good tree, we can estimate

$$\operatorname{est}(e) = 0 \geq 2x(e) - 1 - (2x(e) - 1).$$

If it is a filler edge, we can estimate

$$\operatorname{est}(e) = 1 - 2b(v_e) = 1 - 2(1 - x(e) - x(e)u(e)) = 2x(e) - 1 + x(e)u(e) \geq 2x(e) - 1.$$

5.2.2 Lower bounds for the seed edges

Next we consider seed edges. An overview over the cases in which they can appear is shown in Figure 2. Let e be a seed edge. $e = \{v_e, \bar{v}\}$ can not be contained in a singleton. It can also not be contained in a good tree, as we have

$$1 + u(e) > y(e) = 1 - b(v_e) + 1 - b(\bar{v}) \Leftrightarrow b(v_e) + b(\bar{v}) + u(e) > 1.$$

So if it is contained in a connected component, then this component is a large tree and it was the first edge considered in this component. We estimate

$$\operatorname{est}(e) = 2x(e) - 2 - 2u(e)(1 - x(e)).$$

Otherwise both endpoints are incident to different components. This means that it was rounded down. If these components are singletons or good trees, we can estimate

$$\operatorname{est}(e) = 0 \geq 2x(e) - 2.$$

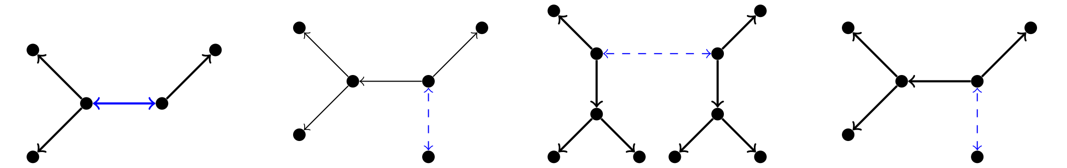
If both are large trees, then e is a filler edge for both and we have

$$\operatorname{est}(e) = 2 - 2(b(v_e) + b(\bar{v})) = 2 - 2(2 - x(e) - x(e)u(e)) = 2x(e) - 2 + 2u(e)x(e) \geq 2x(e) - 2.$$

The last case is that e is incident to one large tree and a good tree or a singleton. This means it is a filler edge for only one endpoint. W.l.o.g. let this endpoint be v_e. We set $y_1 := (1 + u(e))$, $x_1 := 1 - b(v_e)$ and $y_2 := (1 + u(e))$, $x_2 := 1 - b(\bar{v})$. For a later estimate note that then $x_2 \leq \alpha$ as $x(e) \leq \alpha$. We can estimate

$$\operatorname{est}(e) \geq 2x(e) - 2 - (2x_2 - 1).$$

Now almost all estimates are of the same form.



(a) Inside a large tree.

(b) Incident only to good trees or singletons.

(c) As a filler edge for two large trees.

(d) As a filler edge for one large tree and incident to a good tree or a singleton.

Figure 2 The cases in which seed edges can occur. Dashed edges have been rounded down, while solid ones have been rounded up. Thick edges belong to a large tree. For each extension edge e the arrowhead points towards v_e. Seed edges have arrowheads on both ends.

5.2.3 Summary of the estimates

Before we choose α and derive the approximation guarantee, let us summarize the derived estimates.

$$\mathrm{est}(e) \geq$$

$$\begin{array}{lr}
 & \textbf{seed edges} \\
2x(e) - 2 - 0 & \text{incident to good trees or singletons, filler for both ends} \\
2x(e) - 2 - (2x_2 - 1) & \text{filler for one end} \\
2x(e) - 2 - 2u(e)(1 - x(e)) & \text{in a large tree} \\
 & \\
 & \textbf{extension edges} \\
2x(e) - 1 - 0 & \text{filler edge} \\
2x(e) - 1 - (2x(e) - 1) & \text{incident to good tree or singleton} \\
2x(e) - 1 - 2u(e)(1 - x(e)) & \text{inside a component}
\end{array}$$

The base part, which is left in black, now sums up to at most $2x(E(G)) - |V(G)|$, because there is exactly one seed edge for every component that is not a singleton. So it remains to estimate the parts marked in blue. Our goal will be to estimate this part in terms of $|V(G)| - x(E(G))$. That is, find a β, such that we have "sum of blue parts" $\leq \beta(|V(G)| - x(E(G))$. We will achieve this by first estimating for each $\{v_e, w\} \in E(G_{x^*})$ that

$$\text{“blue part”} \leq \begin{cases} \beta(b(v_e) + b(w) + x(e)u(e)) & \text{for seed edges} \\ \beta(b(v_e) + x(e)u(e)) & \text{for extension edges.} \end{cases}$$

Then, we can use that to sum up the estimates of the differences

$$\text{“sum of blue parts”} \leq \beta(b(V(G)) + u(x(E(G)))) \leq \beta(|V(G)| - x(E(G))),$$

where the last inequality follows directly from the LP-inequalities.

In total, we are left with

$$\begin{aligned} |C| &\leq |V(G)| - \sum_{e \in E(G)} \mathrm{est}(e) \\ &\leq |V(G)| - (2x(E(G)) - |V(G)| - \beta(|V(G)| - x(E(G)))) = (2 + \beta)(|V(G)| - x(E(G))) \end{aligned}$$

For some special instances, we can get approximation ratios that are better than 3, but in general we can not be better than a factor 3 with this technique. We will show this in the last section.

5.2.4 A general approximation guarantee

For a general approximation guarantee, we will choose $\alpha := \frac{2}{3}$ to achieve a 3-approximation (so $\beta = 1$).

We start with the edges $\{v_e, w\} = e$ with a "blue part" of 0: Seed edges that are incident to good trees or singletons, filler edges for both ends or extension edges that are filler edges. Here, we directly see that

$$0 \leq b(v_e) + b(w) + x(e)u(e) \text{ for seed or } 0 \leq b(v_e) + x(e)u(e) \text{ for extension edges.}$$

Next, we cover extension edges that are incident to good trees or singletons. They have been rounded down as well, so we have $x(e) \leq \frac{2}{3}$. This implies

$$2x(e) - 1 \leq \frac{1}{3} \leq 1 - x(e) = b(v_e) + x(e)u(e).$$

Analogously, for seed edges that are filler edges for one end, we get

$$2x_2 - 1 \leq b(v_e) + b(w) + x(e)u(e).$$

Finally, we cover the edges that have been rounded up. These are seed edges in a large tree or extension edges inside a component. Here we have $x(e) \geq \frac{2}{3} \geq 2(1 - x(e))$. So we can get

$$2u(e)(1 - x(e)) \leq x(e)u(e)$$

and again from this $2u(e)(1-x(e)) \leq b(v_e)+b(w)+x(e)u(e)$ for seed edges or $2u(e)(1-x(e)) \leq b(v_e) + x(e)u(e)$ for extension edges.

6 The integrality gap of the LP

We will now prove that the integrality gap of the LP is 3. This means that using the approach discussed here, we can not achieve a better approximation guarantee.

▶ **Theorem 13.** *The integrality gap of the LP-relaxation given in Section 3 is at least* 3.

Proof. For an instance I denote by $\mathrm{OPT}(I)$ the value of an optimum (integral) solution and by $\mathrm{OPT}_{\mathrm{LP}}(I)$ the value of an optimum LP-solution. We will provide a sequence I_k of instances, such that $\lim\limits_{k\to\infty} \frac{\mathrm{OPT}(I_k)}{\mathrm{OPT}_{\mathrm{LP}}(I_k)} = 3$.

Let $0 < \epsilon < \frac{1}{2}$. For some $k \geq 3$, let G be a k-star. That is a graph with $k+1$ vertices $\{C\} \cup \{v_1, \ldots, v_k\}$ and edges $\{\{C, v_i\} \mid i = 1, \ldots, k\}$. We set $c \equiv 0$ and $\gamma := 1$. For all edges $e \in E(G)$, we set $u(e) := \frac{1}{2}$. Finally, we set $b(C) := 1 - \epsilon$ and $b(v_i) := \epsilon$ for $i = 1, \ldots, k$. In order to get to a complete graph, we extend G, by adding edges between all pairs v_i, v_j for $i < j$ and set $c(\{v_i, v_j\}) = 0$ and $u(\{v_i, v_j\}) := 1 - \epsilon$. Clearly, the resulting u and c are metric. We will denote this instance by $I_{k,\epsilon}$. A depiction of $I_{k,\epsilon}$ is shown in Figure 3.

In an optimum integral solution to this instance, no edge can be used. This means that $\mathrm{OPT}(I_{k,\epsilon}) = k + 1$. Now we solve the *LP* using the algorithm from section 2, showing that

$$\mathrm{OPT}_{\mathrm{LP}}(I_{k,\epsilon}) = |V| - \sum_{i=1,\ldots,k} x(e_i) = k + 1 - \frac{2}{3} - (k-1)\left(\frac{2}{3} - \frac{\epsilon}{3}\right) = \frac{k}{3} + \frac{(k-1)\epsilon}{3} + 1.$$

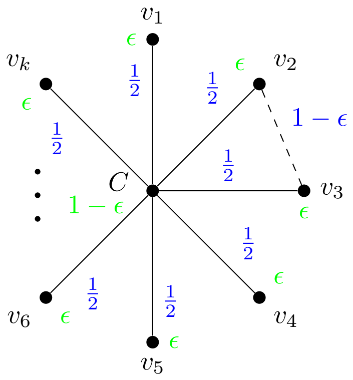


Figure 3 A picture showing the instance described in the proof of Theorem 13. The solid edges belong to the k-star. Edge loads are marked in blue and node loads are marked in green. The dashed edge is an example for the edges added to complete the graph.

Setting $I_k := I_{k,\frac{1}{k^2}}$, we get

$$\lim_{k\to\infty} \frac{\text{OPT}(I_k)}{\text{OPT}_{\text{LP}}(I_k)} = \lim_{k\to\infty} \frac{k+1}{\frac{k}{3} + \frac{(k-1)}{3k^2} + 1} = 3. \qquad \blacktriangleleft$$

Together with the upper bound of 3 given by the analysis of the algorithm, we can conclude:

▶ **Corollary 14.** *The integrality gap of the LP is* 3.

References

1 Esther M Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006. doi:10.1016/J.JALGOR.2005.01.007.

2 Christoph Bartoschek. *Fast Repeater Tree Construction*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2014.

3 Guy Even, Naveen Garg, Jochen Könemann, Ramamoorthi Ravi, and Amitabh Sinha. Min–max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004. doi:10.1016/J.ORL.2003.11.010.

4 Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. Combinatorial optimization in vlsi design. *Combinatorial Optimization – Methods and Applications*, 31:33–96, 2011. doi:10.3233/978-1-60750-718-5-33.

5 Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985. doi:10.1287/MOOR.10.2.180.

6 Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the 18th annual symposium on Computational geometry*, pages 10–18. ACM, 2002. doi:10.1145/513400.513402.

7 M. Reza Khani and Mohammad R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, 2014. doi:10.1007/S00453-012-9740-5.

8 Samir Khuller and Yoram J Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000. doi:10.1137/S0895480197329776.

9 Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

10 Jens Maßberg and Jens Vygen. Approximation algorithms for a facility location problem with service capacities. *ACM Transactions of Algorithms*, 4(4):50:1–50:15, 2008. doi:10.1145/1383369.1383381.

11 Benjamin Rockel-Wolff. A fast 3-approximation for the capacitated tree cover problem with edge loads, 2024. arXiv:2404.10638.

12 Stephan Schwartz. An overview of graph covering and partitioning. *Discrete Mathematics*, 345(8):112884–112900, 2022. doi:10.1016/J.DISC.2022.112884.

13 Vera Traub and Thorben Tröbst. A fast $(2 + \frac{2}{7})$-approximation algorithm for capacitated cycle covering. *Mathematical Programming*, 192(1):497–518, 2022. doi:10.1007/S10107-021-01678-3.

14 Zhou Xu, Dongsheng Xu, and Wenbin Zhu. Approximation results for a min–max location-routing problem. *Discrete Applied Mathematics*, 160(3):306–320, 2012. doi:10.1016/J.DAM.2011.09.014.

15 Wei Yu and Zhaohui Liu. Better approximability results for min–max tree/cycle/path cover problems. *Journal of Combinatorial Optimization*, 37(2):563–578, 2019. doi:10.1007/S10878-018-0268-8.

Approximation Algorithms for the Airport and Railway Problem

Mohammad R. Salavatipour
Department of Computer Science, University of Alberta, Edmonton, Canada

Lijiangnan Tian
Department of Computer Science, University of Alberta, Edmonton, Canada

Abstract

In this paper, we present approximation algorithms for the airport and railway problem (AR) on several classes of graphs. The AR problem, introduced by [2], is a combination of the Capacitated Facility Location problem (CFL) and the network design problem. An AR instance consists of a set of points (cities) V in a metric $d(.,.)$, each of which is associated with a non-negative cost f_v and a number k, which represent respectively the cost of establishing an airport (facility) in the corresponding point, and the universal airport capacity. A feasible solution is a network of airports and railways providing services to all cities without violating any capacity, where railways are edges connecting pairs of points, with their costs equivalent to the distance between the respective points. The objective is to find such a network with the least cost. In other words, find a forest, each component having at most k points and one open facility, minimizing the total cost of edges and airport opening costs. Adamaszek et al. [2] presented a PTAS for AR in the two-dimensional Euclidean metric $\mathbb{R}^2$ with a uniform opening cost. In subsequent work [1] presented a bicriteria $\frac{4}{3}\left(2+\frac{1}{\alpha}\right)$-approximation algorithm for AR with non-uniform opening costs but violating the airport capacity by a factor of $1+\alpha$, i.e. $(1+\alpha)k$ capacity where $0<\alpha\leq 1$, a $\left(2+\frac{k}{k-1}+\varepsilon\right)$-approximation algorithm and a bicriteria Quasi-Polynomial Time Approximation Scheme (QPTAS) for the same problem in the Euclidean plane $\mathbb{R}^2$. In this work, we give a 2-approximation for AR with a uniform opening cost for general metrics and an $O(\log n)$-approximation for non-uniform opening costs. We also give a QPTAS for AR with a uniform opening cost in graphs of bounded treewidth and a QPTAS for a slightly relaxed version in the non-uniform setting. The latter implies $O(1)$-approximation on graphs of bounded doubling dimensions, graphs of bounded highway dimensions and planar graphs in quasi-polynomial time.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms

Keywords and phrases Facility Location, Approximation Algorithms, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.40

Funding *Mohammad R. Salavatipour*: Supported by NSERC DG.
Lijiangnan Tian: Supported by first authors NSERC.

Acknowledgements We want to thank Zachary Friggstad for his comments that improved and simplified Theorem 1. We also talked to Mohsen Rezapour for some initial discussions.

1 Introduction

We study a problem that integrates capacitated facility location and network design problems. The problem referred to as *Airport and Railway* problem denoted as AR (introduced by [2] and studied further in [1]) is the following. Suppose we are given a complete weighted graph $G=(V,E)$ embedded in some metric space (for instance the Euclidean plane), with two cost functions $f: V\to\mathbb{R}_{\geq 0}$ for opening facilities (also known as *airports*) at vertices (also known as *cities*) and $c: E\to\mathbb{R}_{\geq 0}$ for installing railways on the edges in order to connect cities to airports. We are also given a positive integer $k\in\mathbb{Z}_+$ as the *capacity* of each airport.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 40; pp. 40:1–40:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

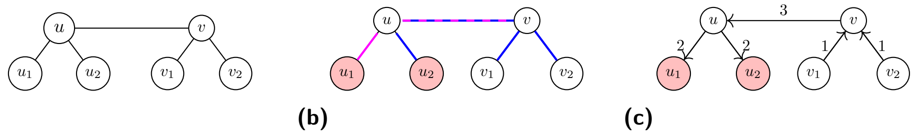


Figure 1 a) An example tree where we assume the airport capacity is 3 and u_1 and u_2 have an opening cost of zero while other vertices have cost infinity; b) The solution to $\widetilde{\mathrm{AR}}$. Pink vertices represent cities with an airport. Each edge is coloured to indicate its cluster. The dashed edge is used by both clusters; c) The solution to AR′. Each directed edge is labelled with its flow value.

The goal is to partition the vertices into a set of clusters each of size at most k, find a set of vertices $A \subseteq V$ at which we open facilities (airports) so that each cluster has exactly one airport, and a set of edges $R \subseteq E$, such that the edges on each cluster induce a connected graph, while minimising the total cost of the edges plus the opening of selected facilities.

Clearly, the graph induced by each cluster must be a tree. So we have a collection of trees, each of size at most k and each having an open facility. The idea is each open facility serves as an airport that will serve all the cities in the cluster it belongs to (including the city at that vertex). The goal is to minimise the total cost

$$C = \sum_{v \in A} f_v + \sum_{e \in R} c_e.$$

To be more precise, a cluster is an airport and the set of all the cities served by it, together with the set of railways connecting the cities to the airport that forms a tree. Adamaszek et al. [1] also defined a relaxed version of AR (they called AR′) where in a feasible solution a component of the forest might have multiple airports and multiple copies of any edge and each component allows routing one unit of flow from all its cities to the airports so that each airport receives at most k flows and each copy of an edge has capacity k. Note that in this version of the problem, the cities belonging to different airports can share the edges of the network. So an edge might be used by cities from different clusters but no more than k in total; in this case, the cost of the edge occurs only once in the objective.

When considering special metrics (e.g. shortest path metrics induced by trees or other special graph classes) we may not have a feasible solution to AR in the strict setting that clusters need to be disjoint. For this reason, we consider a slightly relaxed version of AR, denoted by $\widetilde{\mathrm{AR}}$ where the clusters do not need to be edge-disjoint but each cluster will pay for the edges it uses separately. In other words, each edge is allowed to be used by multiple clusters but each of them needs to pay the cost of the edges they use separately. Considering this relaxed version becomes useful when we are working on specific metrics e.g. shortest path metrics of certain graph classes such as trees (e.g. see Figure 1). Note that in $\widetilde{\mathrm{AR}}$, each connected component in a feasible solution may contain multiple clusters and the total cost that we want to minimise is $\sum_{v \in A} f_v + \sum_{e \in R} c_e \cdot \phi(e)$ where $\phi(e)$ is the number of clusters using the edge e. We highlight that AR′ is a strictly more relaxed setting vs. $\widetilde{\mathrm{AR}}$. In AR′ the cities sending flows to different airports can share the edges of the network and if the flow over an edge is $\leq k$ (even if used to send flow to different airports) the cost of the edge is paid for only once. This is not the case in $\widetilde{\mathrm{AR}}$. For instance, a feasible solution to $\widetilde{\mathrm{AR}}$ in this Figure 1 has two clusters, one u_1, u, v and the other u_2, v_1, v_2 and has a total cost of 6 whereas a feasible solution to AR′ has one component with cost 5.

The AR problem has some characteristics of the Capacitated Facility Location (CFL) problem and network design problem. The instance of AR is the same as CFL with uniform capacities. However, in CFL one has to open a number of facilities and assign each client/city to an open facility (by a direct edge) so that each facility is assigned at most k clients and we minimise the total opening cost and connection cost. The main difference is that in CFL each cluster forms a star (with the facility being the centre) while in AR each cluster is a tree, whose cost might be much cheaper than the star. In AR, the clients might share the same path to be connected to the facility and hence reduce the total cost of forming the railroad network. AR has also similarities to the Capacitated Vehicle Routing Problem (CVRP) and Capacitated Minimum Spanning Tree (CMST). In CMST, the goal is to construct a minimum-cost collection of trees covering all the input vertices, each tree spanning at most k vertices, connected to a single root node. As discussed in [1], AR can be modelled as CMST in general weighted (non-metric) graphs.

The following variants of AR have been studied [1, 2]. For some constant $\beta > 1$, AR_β refers to the bicriteria version of AR, where airport capacity is allowed to be violated by a factor of β (also known as resource augmentation). AR^∞ is a relaxed version where the airport capacity is dropped, or equivalently, set to infinity: $k = +\infty$. When airport opening costs are uniform we refer to it by $\mathbb{1}$AR. Another special case is AR_P where each component is a path with both endpoints having an airport. AR_P is a relaxation of the capacitated vehicle routing problem (CVRP) since not all the paths need to have a common endpoint (the centralised dépôt in CVRP). The original problem is sometimes denoted as AR_F (or simply AR) where we have a general forest.

1.1 Related Work

As mentioned above, [1, 2] have studied AR and some variants of it defined above. No true (non-trivial) approximation is known for AR in general setting. For the case of uniform airport opening cost, for both 1AR and $1\mathrm{AR}_P$, [2] show that the problems are NP-hard in Euclidean metrics and present PTAS's for them.

In [1] the authors consider bicriteria approximations. They present a $\frac{4}{3}\cdot(2+\frac{1}{p})$-approximate for AR_{1+p}, $p \in (0, 1]$ for general metrics. For Euclidean $\mathbb{R}^2$ they present a QPTAS for $\mathrm{AR}_{1+\mu}$, for arbitrary $\mu > 0$ (i.e. violating the capacities by $1 + \mu$) and a $(2 + \frac{k}{k-1} + \varepsilon)$-approximation in polynomial time. To obtain the latter result they obtain a PTAS for AR' on Euclidean metrics and show that a solution to AR' implies a solution for AR at a loss of factor $2 + \frac{k}{k-1}$.

In CFL, we are given a weighted (metric) graph $G = (V, E)$, a facility opening cost function $f : V \to \mathbb{R}_{\geq 0}$, and edge costs $c : E \to \mathbb{R}_{\geq 0}$, and a capacity u_v. The goal is to open a set of facilities $F \subseteq V$, and assign each point $v \in V$ to an open facility so that each open facility v has at most u_v points assigned to it while minimizing the total opening costs plus the assignment costs of points to open facilities. The only difference between CFL and AR is that in CFL the assignment edges in each cluster form a star whereas in AR it forms a minimum tree spanning the nodes of that cluster. There are constant approximation algorithms for CFL in general as well as uniform settings [12, 16].

For CVRP and its variants there are constant-factor approximations in general settings and QPTAS for special metrics such as Euclidean and doubling metrics and minor-free graphs [3, 6, 9, 10]. Another related problem is the *capacitated cycle cover problem* (CCCP) studied in [20]. In this problem, we are given a weighted graph G and parameters k and γ. The goal is to find a spanning collection of cycles of size at most k while minimizing the cost of the edges of the cycles plus γ times the number of cycles. This problem is related to Min-Max Tree Cover and Bounded Tree Cover studied earlier [13, 21]. In [20] the authors present a $(2 + \frac{2}{7})$-approximation for CCCP. This also implies a $(4 + \frac{4}{7})$-approximation for uniform AR.

For CMST, Jothi and Raghavachari [11] give a 3.15-approximation algorithm for Euclidean CMST and a $(2+\gamma)$-approximation for metric CMST, where $\gamma \leq 2$ is the ratio of minimum-cost Steiner tree and minimum spanning tree. As pointed out by [1], AR can be reduced to CMST in non-metric setting.

We refer to [1] for discussion of other related works such as capacitated-cable facility location problem (CCFLP) [17] and sink clustering problem [14].

1.2 Contributions

Although AR (and $\widetilde{\mathrm{AR}}$) are similar to both CFL and CVRP, the mix of capacitated facility location and network design components appears to make it significantly more difficult than both. The approximability of AR for general metrics remains uncertain. Even for more restricted settings such as special metrics (e.g. trees) or uniform opening costs, the approximability of the problem is open.

In this paper, we make progress on some special cases. First, we consider AR with uniform opening cost (i.e. $\mathbb{1}\mathrm{AR}$) on various metrics. For general metrics, we present a simple 2-approximation algorithm for this.

▶ **Theorem 1.** *There is a* 2*-approximation for uniform* AR *on general metrics.*

We also consider graphs of bounded treewidth and present a QPTAS for $\widetilde{\mathrm{AR}}$ on such metrics.

▶ **Theorem 2.** *There is a* QPTAS *for uniform* $\widetilde{\mathrm{AR}}$ *on graphs of bounded treewidth which runs in time* $n^{O(\omega^\omega \cdot \log^3 n/(\varepsilon^2 \log^\omega \omega))}$*. where* ω *is the treewidth of the input graph.*

Next, we consider AR' in the general setting (i.e. with non-uniform facility opening costs). We propose an exact algorithm for trees and graphs of bounded treewidth.

▶ **Theorem 3.** AR' *can be solved in polynomial time on graphs with bounded treewidth.*

Using embedding results for general metrics into tree metrics with $O(\log n)$ distortion as well as embedding of graphs of bounded doubling dimension, graphs of bounded highway dimension, and minor-free graphs into graphs with polylogarithmic treewidth as well as $O(1)$-reduction from AR to AR' ([1]) we obtain the following corollary.

▶ **Corollary 4.** *There is a polynomial time* $O(\log n)$*-approximation for* AR *on general graphs, a* QPTAS *for* AR' *and therefore a quasi-polynomial* $O(1)$*-approximation for* AR *for graphs with bounded doubling dimension, graphs of bounded highway dimension, and minor-free graphs.*

We also show that at a factor 2 loss, we can reduce the general AR problem to the case that facilities have cost 0 or $+\infty$, we denote this case by $0/{+\infty}$ AR. In other words, the special case of the problem that all facilities (to be opened) are given to us and we simply have to build clusters of size at most k each of which has one of the open facilities. Even for this special case, a good approximation remains elusive.

▶ **Theorem 5.** *Given an instance* G *for* AR*, we can build an instance* G' *for* $0/{+\infty}$ AR *such that any* α*-approximate solution to* $0/{+\infty}$ AR *implies a* 2α*-approximate solution for* AR *on* G*.*

In the next section, we prove Theorem 1. Then in Section 3 we prove Theorem 2 and in Section 4 we prove Theorem 3 and Corollary 4. We defer the proof of Theorem 5 to the full version.

2 Algorithm for Uniform AR in General Metric

In this section, we prove Theorem 1. Since each facility (airport) is trivially serving its own city, we refer to the remaining capacity $k-1$ (to serve other clients) as k for simplicity. We assume opening a facility at each vertex costs a uniform value f. Given an instance G we first define a modified instance $\tilde{G}$ for each input graph G. The graph $\tilde{G}$ is obtained by adding a dummy node r to G and connecting r to all the vertices $v \in V$ with an edge of cost $c_{vr} = f$. We first define the MST_r^σ problem and prove the following lower bound.

▶ **Definition 6.** *In the MST_r^σ problem, we are given a graph $G = (V, E)$ with a vertex $r \in V$. The task is to find the minimal cost of the spanning tree of the input graph, while ensuring that the degree of vertex r in the solution is at most σ.*

▶ **Lemma 7.** *If σ is the number of components in an optimum solution to AR on G then the cost of an optimal solution to the MST_r^σ problem on $\tilde{G}$ is a lower bound on the optimal solution to AR on G.*

Proof. Consider an optimal solution ξ to AR on G. Say there are σ components in ξ. After adding into ξ a dummy node r and connecting r to the vertices that are open facilities with an edge of cost f, we obtain a spanning tree T for $\tilde{G}$ of the same cost, where the vertex r has a degree of σ. Namely, this is a feasible solution to MST_r^σ. Therefore, an optimal solution to MST_r^σ on $\tilde{G}$ cannot cost more than the optimal solution to AR on G. ◀

Our algorithm first guesses the number of components in the optimal solution. We do this by enumerating all possibilities. Say there are σ components in the optimal solution for some integer $\sigma \leq n$. Note that we know $\sigma \geq \lceil \frac{n}{k} \rceil$ for certain, as otherwise there must exist some cities that are not getting served. Our algorithm is as follows.

Construct the instance $\tilde{G}$. Solve the MST_r^σ problem on instance $\tilde{G}$. After removing the dummy vertex r, we obtain a set $\mathcal{T} = \{T_1, T_2, \dots T_\sigma\}$ of σ connected components (i.e. trees). Note that we can solve the MST_r^σ problem using the technique of matroid intersection [7].

Let $M_1 = (\tilde{E}, \mathcal{I}_1)$ represent the graphic matroid of $\tilde{G}$ (also known as the cycle matroid or polygon matroid), where the ground set $\tilde{E}$ is the set of edges in $\tilde{G}$, and the set of independent sets $\mathcal{I}_1$ consists of acyclic subgraphs of $\tilde{G}$. That is to say, each independent set corresponds to the edges of a forest in the underlying graph $\tilde{G}$. Let $M_2 = (\tilde{E}, \mathcal{I}_2)$ denote the partition matroid, where the set of independent sets $\mathcal{I}_2$ is defined as follows, where $N(r)$ represents all the edges incident to the vertex r and $\tilde{V}$ is the vertex set of $\tilde{G}$,

$$\mathcal{I}_2 = \left\{ S \subseteq \tilde{E} \;\middle|\; |S \cap N(r)| \leq \sigma,\ |S \cap (\tilde{E} \setminus N(r))| \leq |\tilde{V}| - 1 - \sigma \right\}.$$

In other words, each independent set of this partitional matroid corresponds to the edge set of a subgraph of $\tilde{G}$ with at most $|\tilde{V}| - 1$ edges, where there are at most σ edges incident to the vertex r and at most $|\tilde{V}| - 1 - \sigma$ edges not incident to r.

Note that a feasible solution to MST_r^σ is an independent set of both matroids. The underlying graph must form a spanning tree, so it is an independent set of M_1. The set of edges must satisfy the degree requirement for vertex r, so it is an independent set of M_2. For each connected component $T_i \in \mathcal{T}$, we obtain a cycle C_i in the following way: double the edges of T_i and trace them while short-cutting whenever we encounter a vertex that has been visited. We cut each cycle C_i into a set of disjoint subpaths of fixed length k, except for at most one subpath per cycle that is strictly shorter than k. Essentially, we have transformed the trees in $\mathcal{T}$ into a set of paths. Let $\mathcal{P}_k$ denote the set of paths with length exactly k. For each path in $\mathcal{P}_k$, we simply open one of its cities as an airport. Note that

$|\mathcal{P}_k| \leq \lfloor \frac{n}{k} \rfloor$ since there are at most n vertices (other than the vertex r) in the graph. In addition, as we know $\sigma \geq \lceil \frac{n}{k} \rceil$, we have $|\mathcal{P}_k| \leq \lfloor \frac{n}{k} \rfloor \leq \lceil \frac{n}{k} \rceil \leq \sigma$. Consequently, the cost of opening these $|\mathcal{P}_k|$ airports is $|\mathcal{P}_k| \cdot f \leq \sigma \cdot f$. For those subpaths of length less than k, we simply open one of its vertices as the facility. Note that since there are $|\mathcal{T}| = \sigma$ trees T_i (hence there are σ corresponding cycles C_i), we have at most σ such short subpaths. The current cost is bounded by twice the edge cost of all the trees in $\mathcal{T}$, as well as the facility cost of all these subpaths, which is at most $f \cdot \sigma + |\mathcal{P}_k| \cdot f \leq 2\sigma \cdot f$. Meanwhile, the cost of the MST_r^σ solution is the edge cost of all the trees in $\mathcal{T}$, plus the cost of incident edges of r in the solution, which is $f \cdot \sigma$. Thus, it is obvious that the cost is no more than twice the cost of the MST_r^σ solution.

From the analysis above, it should be easy to see that Theorem 1 follows.

3 QPTAS for Uniform Case in Graphs of Bounded Treewidth

In this section, our goal is to prove Theorem 2. First, recall the definition of graphs with bounded treewidth.

▶ **Definition 8.** *A* tree decomposition *of a graph $G = (V, E)$ is a tree $T = (V', E')$ and a mapping $\Xi : V' \to 2^V$ where each vertex $\beta \in V'$ (also known as a bag) corresponds to a set of vertices of G, such that*

- *For each vertex v in G, it must be included in at least one bag of T.*
- *For each edge uv in G, the pair of vertices $u, v \in V$ must be included in at least one bag of T.*
- *For each vertex v in G, consider the set of all the bags in T that include v. These bags induce a connected component in T.*

The width of a tree decomposition is defined as the cardinality of its largest bag minus one. The treewidth of a graph G is the smallest w such that G has a tree decomposition with width w. Given a graph $G = (V, E)$ of treewidth ω, there is a tree decomposition $T = (V', E')$ of G where T is binary, with depth $h \in O(\log n)$ (where $n = |V|$) and treewidth not exceeding $\omega' = 3\omega + 2$, according to [4]. For simplicity, denote ω' as ω instead. We assume the tree height $h = \delta \log n$ for some constant $\delta > 0$.

Our algorithm for uniform $\widetilde{\mathrm{AR}}$ on bounded treewidth graph relies on the technique developed in [10] for designing QPTAS for CVRP on such metrics. First, we ignore the concept of facilities/airports, we simply pay an extra f for each cluster in our solution (later we designate one vertex in each cluster as the facility to be opened). For that, we define a new version of the problem which we call UAR (meaning AR with *undetermined* airports).

▶ **Definition 9.** (UAR) *The goal is to find a set $\mathcal{F}$ of (not necessarily disjoint) clusters (i.e. trees) using edges in the graph. The size of each cluster must not exceed the capacity constraint k. Each cluster $\gamma \in \mathcal{F}$ has a cost of f and we want to minimise the total cost, which is defined as*

$$|\mathcal{F}| \cdot f + \sum_{\gamma \in \mathcal{F}} \mathrm{cost}(\gamma)$$

where $\mathrm{cost}(\gamma)$ *denotes the railway cost of the cluster γ.*

Since this is a relaxed version of the original problem (as we do not specify the location of the facilities), its cost is a lower bound of that of the original problem. We can think of each vertex in V to have one unit of demand which needs to be sent to an airport to be served. We may add dummy demands to a vertex during the algorithm, so a vertex may end up having more than one unit of demand. The size of a cluster is defined to be the sum of demands on all its vertices, instead of just the number of vertices. Note that a component may not include every vertex that it passes through, as a component may be simply using the edges of a vertex to get to somewhere else, which can also be seen as not picking up the demand of the vertex. Be mindful that, from the perspective of demands, the size of a component is the number of demands it includes, instead of the number of vertices. Therefore the clusters in the solution are not necessarily edge-disjoint or vertex-disjoint, but the total number of demands in each cluster obeys the capacity constraint.

For clarity, we refer to the vertices in T as *bags*, to differentiate them from the vertices in G. For the notation β, we refer to it as the name of the bag $\beta \in V(T)$ as well as the corresponding set of vertices $\beta \subseteq V(G)$. For each bag β, denote the union of vertices in all of the bags in the subtree T_β as C_β. Note that C_β also denotes the set of all bags in T_β.

Each vertex of G may appear in multiple bags of T as tree decomposition generates duplicates. In order to make sure the demand of a vertex does not get duplicated in T, for every vertex $v \in V(G)$, we assume that the copy/instance of v in the bag $\tilde{\beta}$ that is the closest to the root bag (we know there is a unique one and we denote this copy of v as $\tilde{v}$) has a demand of one, and the rest of the copies of v (which resides in other bags) have demand zero.

Given an optimal solution denoted as OPT, we will demonstrate a process for transforming it into a near-optimal solution for UAR and thereby show the existence of such a near-optimal solution. This transformation occurs incrementally on T, moving from the bottom to the top, one level at a time. The solution before modifying level ℓ is denoted as OPT_ℓ, and after the modification as $\mathrm{OPT}_{\ell-1}$.

Overview of the approach and relation to [10]. Our goal is to show the existence of a near-optimum solution with certain structures. Suppose OPT is an optimum solution for UAR and *OPT* is its value. We aim to find a near-optimal solution, of cost $(1+O(\varepsilon))\mathit{OPT}$, where each vertex has at least one unit of demand, and the size of partial clusters in any subtree T_β can only be one of *polylogarithmically* many values. Two concepts are required to describe the following data structures, namely, the notions of partial and complete clusters. We consider a non-root bag $\beta \in V(T)$ and the subtree rooted at β, T_β. A complete cluster in T_β is a cluster that is entirely in the graph C_β, and a partial cluster is one that uses vertices both inside C_β and outside. Similar to [10], we first assume that the number of clusters in OPT is sufficiently large, that is, at least $\lambda \log n$ for some large number λ. Otherwise, if the number of clusters in OPT is upper-bounded by $\Sigma = \lambda \log n$ then a simple DP can solve the problem exactly (see [19]). Given an optimal solution OPT, we will demonstrate a process for transforming it into a near-optimal solution with certain structural properties that help us find one using dynamic programming. This transformation occurs incrementally on T, moving from the bottom to the top, one level at a time. The solution before modifying level ℓ is denoted as OPT_ℓ, and after the modification as $\mathrm{OPT}_{\ell-1}$. Looking at how OPT_ℓ looks like, we would like to "approximately" keep the sizes of partial clusters that extend below β in T_β. A standard approach is to "bucket" the sizes of partial clusters into buckets where each bucket contains all those sizes that are within $(1+\varepsilon)$ of each other (e.g. bucket i being values in $(1+\varepsilon)^i \dots [(1+\varepsilon)^{i+1} - 1]$. This will reduce the complexity of the DP table to quasi-polynomial: we keep the number of partial clusters of each bucket and try to fill in the DP table bottom-up. The problem is that then when we are combining solutions in the DP table, since we are keeping the sizes approximately (and sacrificing precision), we may violate the capacities unknowingly. The idea developed in [10] was to modify OPT by reducing the sizes of the clusters (at a small increase in the number of clusters) so that even if we scale the sizes of the new clusters by a small number, they are still capacity-respecting. They used a technique that was used later in [15], called *adaptive rounding* that we also use here to round the sizes of partial clusters in T_β for any bag $\beta \in T$. At each bag β, for clusters that are in the same "bucket" we swap parts of them with a net effect of reducing their sizes while having only a poly-logarithmic many possible bucket sizes at the end. We formalize this in the following.

▶ **Definition 10.** *Define the **threshold values** $\{\sigma_1, \dots, \sigma_\tau\}$ where*

$$\sigma_i = \begin{cases} i & 1 \le i \le \lceil 1/\varepsilon \rceil \\ \lceil \sigma_{i-1} \cdot (1+\varepsilon) \rceil & i > \lceil 1/\varepsilon \rceil \end{cases}$$

in such a way that the last threshold $\sigma_\tau = k$. So $\tau \in O(\log k/\varepsilon)$.

We adapted the definitions from [10]. Consider a bag β that is situated at level ℓ. We consider partial clusters that cross β and based on their size in C_β we bucket them. Bucket i contains those partial clusters whose size is in the range $[\sigma_i, \sigma_{i+1})$. Now let's focus on all (partial) clusters that are in bucket i of bag β. Each of these clusters has some vertices in C_β and some vertices outside. For a set $S \subset \beta$ consider all the partial clusters in bucket i that their intersection with β is S. So each of them will form a number of connected components in C_β where each component contains some part of S; this defines a partition of S. We consider all those partial clusters that have the same partition of S together (defined below).

▶ **Definition 11.** *For a bag β at level ℓ in T, for each set $S \subseteq \beta$ and partition $\wp_S$ of S, consider the set $b_S^{\wp_S}$ which contains the clusters that use exactly the set of vertices $S \subseteq \beta$ to span into C_β, where $\wp_S$ denotes a partition of the set S based on connectivity of the of those clusters in C_β. Define the i-th bucket of $b_S^{\wp_S}$, denoted as b_i, to store clusters in OPT_ℓ that have a size between $[\sigma_i, \sigma_{i+1})$ inside C_β, where σ_i is the i-th threshold value. Denote this bucket by a tuple $(\beta, b_i, S, \wp_S)$. Denote the number of clusters in bucket $(\beta, b_i, S, \wp_S)$ as $n_{\beta,i}^{S,\wp_S}$.*

Essentially, the set S represents the interface that the clusters in the bucket $(\beta, b_i, S, \wp_S)$ use to attach to the rest of their parts in C_β, and $\wp_S$ is a set that describes the connectivity between the vertices of S in C_β. That is, each part in the partition $\wp_S$ specifies a subset of vertices of S that need to be connected below. So if $u, v \in S$ and there is some set $P \in \wp_S$ such that $P \supseteq \{u, v\}$, then u and v need to be connected in C_β by some cluster. For simplicity, we just write $\wp_S$ as $\wp$.

▶ **Definition 12.** *A bucket b is said to be* `small` *if it contains no more than $\alpha \log^2 n/\varepsilon$ clusters and is otherwise said to be* `big`*, for some constant $\alpha \ge \max\{1, 20\delta\}$.*

▶ **Definition 13.** *For a big bucket $(\beta, b_i, S, \wp)$, define g groups where $g = \frac{2\delta \log n}{\varepsilon}$, denoted as $G_{i,1}^{\beta,S,\wp}, G_{i,2}^{\beta,S,\wp}, \dots, G_{i,g}^{\beta,S,\wp}$ in the following way (for simplicity assume the size of this bucket is a multiple of g, if not add some empty clusters to achieve this). Sort the clusters in the (padded) bucket in non-decreasing order, and put the first $\frac{n_{\beta,i}^{S,\wp}}{g}$ clusters into $G_{i,1}^{\beta,S,\wp}$, the second $\frac{n_{\beta,i}^{S,\wp}}{g}$ into $G_{i,2}^{\beta,S,\wp}$, etc. For each group $G_{i,j}^{\beta,S,\wp}$, denote the size of its smallest cluster as $h_{i,j}^{\beta,S,\wp,\min}$ and the size of its biggest cluster as $h_{i,j}^{\beta,S,\wp,\max}$.*

Suppose we are considering a big bucket of β and a partial cluster Γ is in the group $j > 1$ of the big bucket. We find its top (that is, the part of the cluster that is outside of T_β) and reassign it to another partial cluster (that is no bigger than Γ) with the same order in the previous group (i.e., group $j-1$) as the order of Γ in group j. The vertices that were originally covered by the partial clusters in the last group are referred to as *orphans*. This is essentially the rounding between groups of a big bucket that was done in [10] for the CVRP on bounded treewidth graphs. The idea is that by this operation, the size of each cluster goes down enough such that if we "approximate" the sizes by the size of the biggest cluster in each group, we are still satisfying the capacity constraints. However, some vertices that were covered by the partial clusters of the last group are now left "uncovered" (or orphan). We will use some extra clusters to pick up (serve) the now orphan vertices.

We come up with a structure theorem that shows the existence of a near-optimal solution with certain structures, and then provide a dynamic programming algorithm for the UAR problem.

3.1 Structure Theorem for Graphs with Bounded Treewidth

The steps of modifying OPT to a near-optimal solution (denoted as OPT$'$) are largely the same as the ones in [10]. Let's assume we randomly choose clusters from OPT, denoted as C, with a probability of ε. After selecting these clusters, we duplicate each chosen one and assign both duplicates of each chosen cluster to one of the levels ℓ that it visits[1], with equal probability. These duplicated clusters are referred to as the *extra clusters*. We will bound their total cost. The proof is very similar to the one in [10] and we only need to show the part concerning the facility costs.

Recall f is the (uniform) facility opening cost, ε is the probability each cluster γ in OPT is selected as the extra cluster, k is the capacity of each cluster, and ω is the treewidth of G.

▶ **Lemma 14.** *The expected cost of the extra clusters sampled is* $2\varepsilon \cdot \textsc{opt}$.

We make use of the following modified definitions and lemmata from [10]. They apply to our problem as the proofs of the lemmata are almost identical.

Denote the bags in level ℓ of T as B_ℓ. Define the set X_ℓ to comprise the extra clusters assigned to bags at level ℓ. For every bag $\beta \in B_\ell$ and its bucket $(\beta, b_i, S, \wp)$, let $X^{S,\wp}_{\beta,i}$ represent the extra clusters (using vertices in S to span into C_β, with $\wp$ depicting connectivity downwards) in X_ℓ whose partial clusters inside C_β has a size that falls within the range defined by bucket b_i. For an extra cluster $\gamma \in X^{S,\wp}_{\beta,i}$, it covers some partial cluster $\zeta \in G^{\beta,S,\wp}_{i,g}$ (which is without its top). That is, the extra cluster γ only picks up demands at the levels $\geq \ell$ and acts as the top of ζ, in particular, this combined cluster picks up only those demands of ζ's vertices (which are all orphans).

▶ **Lemma 15.** *At any level ℓ, each bag $\beta \in B_\ell$ and its big buckets $(\beta, b_i, S, \wp)$ satisfy, w.h.p.*

$$\left|X^{S,\wp}_{\beta,i}\right| \geq \frac{\varepsilon^2}{\delta \log n} \cdot n^{S,\wp}_{\beta,i}.$$

▶ **Lemma 16.** *For all bags β at level ℓ in T, their big buckets $(\beta, b_i, S, \wp)$ and partial clusters in $G^{\beta,S,\wp}_{i,g} \subseteq b_i$, we can make adjustments to the extra clusters present in $X^{S,\wp}_{\beta,i}$ without incurring any additional cost, and introduce some dummy demands within β when necessary, so that:*

1. *The partial clusters in $G^{\beta,S,\wp}_{i,g}$ are now incorporated into some clusters in $X^{S,\wp}_{\beta,i}$. (That is, all the demands that were covered by some partial cluster in $G^{\beta,S,\wp}_{i,g}$ are picked up by some cluster in $X^{S,\wp}_{\beta,i}$.)*

[1] If a cluster γ passes crosses bag of level ℓ, we say γ visits or crosses level ℓ.

2. *The modified partial clusters that cover the orphans (i.e., vertices in $G_{i,g}^{\beta,S,\wp}$) have precisely the size of $h_{i,g}^{\beta,S,\wp,\max}$ and all clusters remain underneath the size limit of k units of demand.*
3. *For each modified partial cluster in the set $X_{\beta,i}^{S,\wp}$, its partial clusters at a bag $\beta' \in B_{\ell'}$ is also of one of $O(\log k \log^2 n/\varepsilon^2)$ many sizes, where ℓ' is any lower levels $> \ell$.*

Note that when we add dummy demands for some cluster γ in some bucket $(\beta, b_i, S, \wp)$, we simply add these dummy demands onto the vertices in S. Using these lemmata and a very similar proof to the one in [10], we can obtain a Structure Theorem for our UAR problem in the case of graphs with bounded treewidth.

▶ **Theorem 17.** (Structure Theorem) *Consider an instance $\mathcal{I}$ for the* UAR *problem. Denote its optimal solution as* OPT*, with cost* $\textsc{opt}$*. We can transform* OPT *to another solution* OPT′ *so that, with high probability,* OPT′ *is a near-optimal solution of cost at most* $(1+2\varepsilon)\textsc{opt}$*. Additionally, at every β in* OPT′*, all the clusters in C_β have one of $O(\log k \log^2 n/\varepsilon^2)$ possible sizes. Consider a bucket $(\beta, b_i, S, \wp)$ in* OPT′*. We must have*

- *If b_i is small, the number of partial clusters in C_β whose size falls within b_i is at most $\alpha \log^2 n/\varepsilon$.*
- *If b_i is big, it has exactly $g = 2\delta \log n/\varepsilon$ group sizes which are denoted as*

$$\sigma_i \le h_{i,1}^{\beta,S,\wp,\max} \le h_{i,2}^{\beta,S,\wp,\max} \le \cdots \le h_{i,g}^{\beta,S,\wp,\max} < \sigma_{i+1}$$

  Each cluster in b_i has a size of one of the h-values above.

Having this structure theorem one can design a (relatively complex) DP to compute a near-optimum solution as guaranteed by this structure theorem. This DP builds upon ideas of the DP in [10] but has more complexity as the clusters here do not necessarily have a common point (like the dépôt in the CVRP problem). This will show that we can compute a solution such as OPT′ in Theorem 17 in time $n^{O(\omega^\omega \cdot \log^3 n/(\varepsilon^2 \log^\omega \omega))}$.

We can transform the approximate solution obtained for the UAR problem into a solution to the $\widetilde{\mathrm{AR}}$ problem, without any increase in the cost. All we need to do is to pick a node in each cluster to open a facility at (since we are already paying f for each cluster, this cost is accounted for in the solution to UAR). This can be easily done since in a solution to UAR each vertex is “covered” by a unique cluster.

4 Constant Approximation for Nonuniform-AR

In this section, we prove Theorem 3. For ease of exposition, we present the proof for the case of trees (the extension to graphs with bounded treewidth appears in the full version). Recall that in the relaxation AR′, we are given a graph $G = (V, E)$ where each vertex $v \in V$ has a non-negative opening cost a_v and each edge $e \in E$ has a non-negative weight c_e. Every edge and vertex has capacity $k \in \mathbb{N}_+$. Find a subset of vertices $\Phi \subseteq V$ as facilities (also known as airports), and a multiset Ξ of edges from E to get a transportation network that ensures one unit of flow from each vertex in V can be sent to facilities in Φ, without violating the capacity constraint on any edge or facility. The goal is to find such a network while minimising the total cost $\sum\limits_{v \in \Phi} a_v + \sum\limits_{e \in \Xi} c_e$. First, we prove some properties in an optimum solution to AR′.

▶ **Lemma 18.** *In an optimum solution, we can assume there are not any flows of opposite directions on the same edge, as we can uncross them by redirecting each flow and attain a lower cost.*

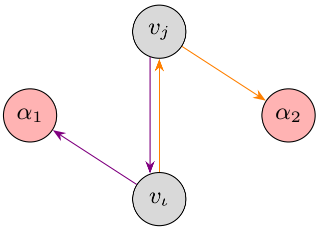


Figure 2 A simplest example of crossing flows in AR′. The red vertices are open facilities.

Note that it is allowed for multiple clients to use the same edge to send their demands in the same direction.

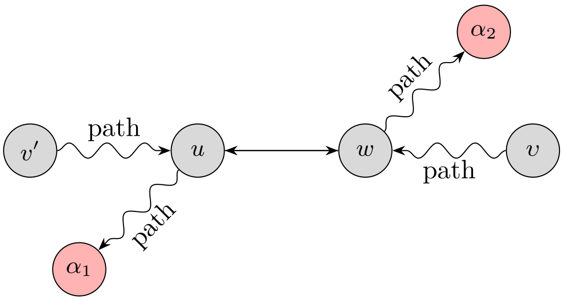


Figure 3 The crossing flow is at the edge uw.

Proof. Without loss of generality, assume the vertices v' and v caused crossing flow at edge uw. That is, the demand of v' travels from v' to u, crosses the edge uw from u to w, and from w to a facility α_2; and the demand of v travels from v to w, crosses the edge uw from w to u, and from u to a facility α_1. We can reroute so that the demand of v' travels from v' to u, and then from u to the facility α_1; and similarly, the demand of v travels from v to w, and then from w to the facility α_2. It is easy to see such a rerouting makes the total cost decrease, for the demands of both vertices v' and v now take a shorter path to be served. ◀

Consider a tree T as the input graph. A subproblem here is defined on the subtree T_v for each vertex v. Since we aim to obtain a flow network in T, each vertex v, as the root of the subtree T_v, will be considered a portal in the corresponding subproblem. There is thus a DP cell for each vertex v in T. Note that at each vertex v, the portal configuration ψ_v simplifies to the direction and value of the flow at v

$$\psi_v = \pm f_v$$

where we use $-$ (minus sign) to signify the flow is leaving T_v, and $+$ (plus sign) to signify the flow is entering T_v. f_v is the absolute value of the signed integer ψ_v and denotes the value of the unidirectional (integral) flow passing through the vertex v and satisfies $0 \leq f_v \leq n$, where n is the number of vertices in T. Note that in AR′, if an edge needs to carry a flow f_v, then we need to install $\left\lceil \frac{f_v}{k} \right\rceil$ parallel edges in the solution. At each vertex v, we also consider both of the scenarios where v is an airport or it is not. We use a Boolean variable $\pi_v = \textsc{True}$ (or $\pi_v = 1$) to indicate that the portal v is opened as an airport.

We define the DP table $\mathbf{D}$ as follows, for each v in T, let the entry $\mathbf{D}[v, \pi_v, \psi_v]$ store the cost of the optimal solution to AR′ on T_v with the amount of flow going in/out of T_v conforming to ψ_v, with portal v opened as an airport if and only if π_v.

At each node, we also consider its parent edge and see it as part of the subtree T_v. For the root node ϑ, we assume its parent edge has cost 0. The result will be $\min_{\pi_\vartheta}\{\mathbf{D}[\vartheta, \pi_\vartheta, \psi_\vartheta = 0]\}$ as there will be no flow entering or leaving T at the root.

Base cases: At a leaf node v, denote the parent edge of v as e. Recall $f_v = |\psi_v|$.

$$\mathbf{D}[v, \pi_v, \psi_v] = a_v \cdot \pi_v + \begin{cases} c_e & \text{if } \psi_v = -1 \\ c_e \cdot \left\lceil \dfrac{f_v}{k} \right\rceil & \text{if } 0 \le \psi_v < +k \text{ and } \pi_v = 1 \\ +\infty & \text{otherwise} \end{cases}$$

Here $\psi_v = -1$ means there is one unit of flow going out of the leaf v (actually does not need to open a facility at v). If $0 \le \psi_v < +k$, it means v does not emit any flow or it is absorbing flows, then we have to make sure $\pi_v = \textsc{True}$. Note that in this case, $\left\lceil \frac{f_v}{k} \right\rceil = 1$ when $0 < \psi_v < +k$, and $\left\lceil \frac{f_v}{k} \right\rceil = 0$ when $\psi_v = 0$. If $\psi_v \ge +k$ then we know it is not achievable, since a facility has capacity k and cannot absorb more flows. If $\psi_v < -1$ then it is simply impossible, as a vertex only has one unit of demand and cannot emit more than that. For these cases, we set the entry to $+\infty$.

For a node v with z children $w_1, w_2, \ldots, w_z$, similar to the case of uniform facility cost on trees in the previous chapter, we define an inner DP table $\mathbf{B}$. Assume we have computed $\mathbf{D}[w_j, \pi_{w_j}, \psi_{w_j}]$ for all possible π_{w_j} and ψ_{w_j}, for all $1 \le j \le z$. Let $\mathbf{B}[v, \pi_v^j, \psi_v^j, j]$ store the cost of the optimal solution to AR$'$ on T_v as if the portal v only has children $w_1, w_2, \ldots, w_j$. Lastly, we define $\mathbf{D}[v, \pi_v, \psi_v] = \mathbf{B}[v, \pi_v, \psi_v, z]$.

Case 1: $j = 1$. Only consider the first child of v.

$$\mathbf{B}[v, \pi_v^1, \psi_v^1, 1] = \min_{\psi_{w_1}} \left\{ \mathbf{D}[w_1, \pi_{w_1}, \psi_{w_1}] + a_v \cdot \pi_v^1 + c_e \cdot \left\lceil \frac{f_v^1}{k} \right\rceil \,\middle|\, \eta(\pi_v^1, \psi_v^1, \psi_{w_1}) = \textsc{True} \right\}$$

where $\eta(\pi_v^1, \psi_v^1, \psi_{w_1})$ is a Boolean indicator function that takes into account the flow on v's parent edge and the edge vw_1, as well as the decision about whether or not to open the portal v as an airport. It is true if and only if all these parameters are compatible. Recall that f_v is the absolute value of ψ_v.

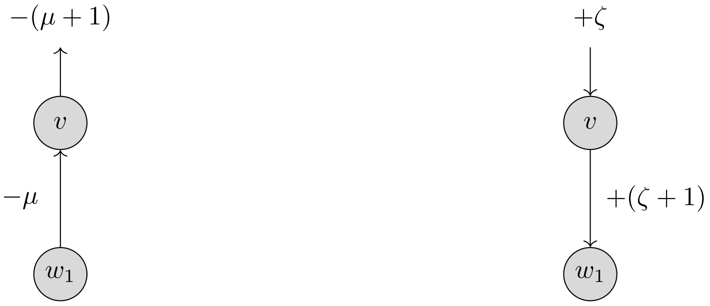
$$\eta(\pi_v^1, \psi_v^1, \psi_{w_1}) = \begin{cases} \textsc{True} & \text{if } 0 \le \psi_v^1 - \psi_{w_1} < k \ \wedge \ \pi_v^1 = \textsc{True}, \\ & \text{or if } \psi_{w_1} - \psi_v^1 = 1 \\ \textsc{False} & \text{otherwise} \end{cases}$$

The case $\psi_{w_1} - \psi_v^1 = 1$ means that v does not act like an airport as it is not absorbing any flow, and is sending its own demand elsewhere (hence unnecessary to open an airport there).

The case $0 \le \psi_v^1 - \psi_{w_1} < k$ means the portal v is absorbing flows and v must be opened as an airport. The other cases are impossible, either because v is absorbing too much flow which violates its capacity limit, or because v is sending out more than one unit of flow.

Case 2: For $2 \le j \le z$. Assume all entries of the form $\mathbf{B}[v, \pi_v^{j-1}, \psi_v^{j-1}, j-1]$ have been computed. We define

$$\mathbf{B}[v, \pi_v^j, \psi_v^j, j] = \min_{\substack{\pi_{w_j}, \pi_v^{j-1}, \psi_{w_j}, \psi_v^{j-1}: \\ \pi_v^j \ge \pi_v^{j-1}, \\ \eta\left(\pi_v^j, \psi_v^j, \psi_v^{j-1}, \psi_{w_j}\right) = \textsc{True}}} \qquad (\Omega)$$



(a) Portal v is sending its demand outside T_v.

(b) Portal v is sending its demand into T_{w_1}.

Figure 4 Here μ and ζ are non-negative integers. The label on edge vw_1 represents ψ_{w_1} and the label above v stands for ψ_v^1.

The expression Ω should be

$$\left\{ \mathbf{D}[w_j, \pi_{w_j}, \psi_{w_j}] + \mathbf{B}[v, \pi_v^{j-1}, \psi_v^{j-1}, j-1] + a_v \cdot \left(\pi_v^j - \pi_v^{j-1}\right) + c_e \cdot \left\lceil \frac{f_v^j - f_v^{j-1}}{k} \right\rceil \right\}$$

where we define the indicator function η as follows:

$$\eta(\pi_v^j, \psi_v^j, \psi_v^{j-1}, \psi_{w_j}) = \begin{cases} \textsc{True} & \text{if } 0 < \psi_v^j - (\psi_v^{j-1} + \psi_{w_j}) \leq k \ \wedge \ \pi_v^j = \textsc{True}, \\ & \text{or if } \psi_v^{j-1} + \psi_{w_j} = \psi_v^j \\ \textsc{False} & \text{otherwise} \end{cases}$$

Let e denote v's parent edge. The case $\psi_v^{j-1} + \psi_{w_j} = \psi_v^j$ means that after taking w_j (the j-th child of v) into consideration, the flow on e whilst only considering the first $j-1$ children (which is ψ_v^{j-1}), and the flow on the edge vw_j adds up to the flow on e while considering all the j children (which is ψ_v^j). This means the portal v is not absorbing any of the flow from T_{w_j}, and thus there is no need to open it as an airport if it has not been opened. The case $0 < \psi_v^j - (\psi_v^{j-1} + \psi_{w_j}) \leq k$ means after taking w_j into consideration, the portal v is absorbing flows and needs to be opened, if it has not been opened. Note that $\left\lceil \frac{f_v^j - f_v^{j-1}}{k} \right\rceil$ can be negative if $f_v^j < f_v^{j-1}$, which means the "load" on the parent edge of v has decreased and we pay less on the edge cost. This exact algorithm on trees suggests we have an $O(\log n)$-approximation algorithm for the general metric (using metric approximation, also known as embeddings by tree metrics).

4.1 Algorithm Efficiency

We will use a bottom-up approach, assuming that the relevant entries for subproblems have already been pre-computed. At any step, checking the value for the indicator function η takes $O(1)$ time. To compute $\mathbf{B}[v, \pi_v^j, \psi_v^j, j]$, we need to consider all possible ψ_{w_j} and ψ_v^{j-1}, which is in total $O(n^2)$ possibilities. Since there are n nodes in the tree, the time for computing the table $\mathbf{D}$ is in $O(n^4)$.

4.2 Generalisation for AR with Steiner Vertices

In this section, we describe how the algorithm above can be generalised for AR$'$ with Steiner vertices with a few modifications. More generally, this algorithm can apply to the case where the set of facilities or the set of clients is not the same as the entire vertex set of the input graph. If a vertex v is not part of the set of facilities, it should not be opened as a facility (after all, no facility cost has been defined for it). So the Π-vector should not allow any copy of v to be opened. If a vertex v is not part of the set of clients, it carries no demand, and so does any of its copies in the tree decomposition.

Note that this will be useful when we try to embed a graph into a graph with bounded treewidth where the host graph of the input graph (via graph embedding) may have Steiner vertices. If Δ is the aspect ratio of G (ratio of largest to smallest edge cost) then by standard scaling (see for e.g. [10]) one can assume that Δ is bounded by polynomial in n at a loss of $(1+\epsilon)$ on optimum solution.

We use the following lemma by [18] about embedding graphs of doubling dimension D into a graph with treewidth $\omega \leq 2^{O(D)}\left\lceil\left(\frac{4D\log\Delta}{\varepsilon}\right)^{D}\right\rceil$.

▶ **Lemma 19** (Theorem 9 in [18]). *Let (X,d) be a metric with doubling dimension D and aspect ratio Δ. Given any $\varepsilon > 0$, the metric (X,d) can be $(1+\varepsilon)$ probabilistically approximated by a family of treewidth ω-metrics for*

$$\omega \leq 2^{O(D)}\left\lceil\left(\frac{4D\log\Delta}{\varepsilon}\right)^{D}\right\rceil .$$

We adapt Theorem 8 and its proof from [10] to get the following result.

▶ **Theorem 20.** *For any $\varepsilon > 0$ and $D > 0$, given an input graph G of the AR' problem where G has doubling dimension D, there is an algorithm that finds a $(1+\varepsilon)$-approximate solution in time $n^{O\left(D^D \log^D n/\varepsilon^D\right)}$.*

We introduce the following lemma proposed by [8] about embedding graphs of highway dimension W into a graph with treewidth $\omega \in (\log\Delta)^{O\left(\log^2\left(\frac{W}{\varepsilon\lambda}\right)/\lambda\right)}$.

▶ **Lemma 21** (Theorem 1.3 in [8]). *Let G be a graph with highway dimension W of violation $\lambda > 0$, and aspect ratio Δ. For any $\epsilon > 0$, there is a polynomial-time computable probabilistic embedding H of G with expected distortion $1+\varepsilon$ and treewidth ω where*

$$\omega \in (\log\Delta)^{O\left(\log^2\left(\frac{W}{\varepsilon\lambda}\right)/\lambda\right)}.$$

We adapt Theorem 9 and its proof from [10] to get the following result.

▶ **Theorem 22.** *For any $\varepsilon > 0$, $\lambda > 0$ and $W > 0$, given an input graph G of the AR' problem where G has highway dimension W and violation λ, there is an algorithm that finds a $(1+\varepsilon)$-approximate solution in time $n^{O\left(\log^{\log^2\left(\frac{W}{\varepsilon\lambda}\right)\cdot\frac{1}{\lambda}} n\right)}$.*

We introduce the following lemma proposed by [5] about embedding minor-free graphs (including planar graphs, which is a kind of K-minor-free graphs) into a graph with treewidth $O_K\left((\ell+\ln n)^6/\varepsilon\cdot\ln^2 n\cdot(\ln n+\ln\ell+\ln(1/\varepsilon))^5\right)$ where ℓ is the logarithm of the aspect ratio of the input graph.

▶ **Lemma 23** (Theorem 1.1 in [5]). *For every fixed graph K, there exists a randomised polynomial-time algorithm that, given an edge-weighted K-minor-free graph $G=(V,E)$ and an accuracy parameter $\varepsilon > 0$, constructs a probabilistic metric embedding of G with expected distortion $(1+\varepsilon)$ into a graph of treedepth (the treedepth of a graph is an upper bound on its treewidth)*

$$O_K\left((\ell+\ln n)^6/\varepsilon\cdot\ln^2 n\cdot(\ln n+\ln\ell+\ln(1/\varepsilon))^5\right)$$

where $n=|V|$ and $\ell=\log\Delta$ is the logarithm of the aspect ratio Δ of the metric induced by G.

▶ **Theorem 24.** *For any $\varepsilon > 0$, given an input graph G of the* AR′ *problem where G is a minor-free graph, there exists an algorithm that finds a $(1+\varepsilon)$-approximate solution in time $n^{O_K\left(\log^8 n\cdot(\log n+\log(1/\varepsilon))^5/\varepsilon\right)}$.*

Theorems 20, 22, and 24 imply Corollary 4.

5 Concluding Remarks

The special case of $0/+\infty$ AR (at a factor 2 loss) is equivalent to the following variant of CCCP: given a collection R of dépôts in a metric, find a collection of cycles of size $\leq k$ each containing a unique dépôt that together covers all the non-dépôt nodes. Although there are constant-factor approximations for CVRP, we do not know of a good approximation for this version.

References

1 Anna Adamaszek, Antonios Antoniadis, Amit Kumar, and Tobias Mömke. Approximating Airports and Railways. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2018.5.

2 Anna Adamaszek, Antonios Antoniadis, and Tobias Mömke. Airports and Railways: Facility Location Meets Network Design. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2016.6.

3 Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. *Mathematical Programming*, 197(2):451–497, 2023. doi:10.1007/S10107-022-01841-4.

4 Hans L. Bodlaender and Torben Hagerup. Parallel Algorithms with Optimal Speedup for Bounded Treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

5 Vincent Cohen-Addad, Hung Le, Marcin Pilipczuk, and Michał Pilipczuk. Planar and Minor-Free Metrics Embed into Metrics of Polylogarithmic Treewidth with Expected Multiplicative Distortion Arbitrarily Close to 1, 2023. arXiv:2304.07268.

6 Aparna Das and Claire Mathieu. A Quasi-polynomial Time Approximation Scheme for Euclidean Capacitated Vehicle Routing. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 390–403. SIAM, 2010. doi:10.1137/1.9781611973075.33.

7 Jack Edmonds. Matroid Intersection. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization I*, volume 4 of *Annals of Discrete Mathematics*, pages 39–49. Elsevier, 1979. doi:10.1016/S0167-5060(08)70817-3.

8 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1+\epsilon)$-Embedding of Low Highway Dimension Graphs into Bounded Treewidth Graphs. *SIAM Journal on Computing*, 47(4):1667–1704, January 2018. doi:10.1137/16m1067196.

9 Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R. Salavatipour. Improved Approximations for Capacitated Vehicle Routing with Unsplittable Client Demands. In Karen I. Aardal and Laura Sanità, editors, *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2022. doi:10.1007/978-3-031-06901-7_19.

10 Aditya Jayaprakash and Mohammad R. Salavatipour. Approximation Schemes for Capacitated Vehicle Routing on Graphs of Bounded Treewidth, Bounded Doubling, or Highway Dimension. *ACM Transactions on Algorithms*, 19(2), March 2023. doi:10.1145/3582500.

11 Raja Jothi and Balaji Raghavachari. Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and Its Variants in Network Design. *ACM Transactions on Algorithms*, 1(2):265–282, October 2005. doi:10.1145/1103963.1103967.

12 Mong-Jen Kao. Improved LP-based approximation algorithms for facility location with hard capacities. *arXiv preprint*, 2021. arXiv:2102.06613.

13 M. Reza Khani and Mohammad R. Salavatipour. Improved Approximation Algorithms for the Min-max Tree Cover and Bounded Tree Cover Problems. *Algorithmica*, 69(2):443–460, 2014. doi:10.1007/S00453-012-9740-5.

14 Jens Maßberg and Jens Vygen. Approximation Algorithms for a Facility Location Problem with Service Capacities. *ACM Transactions on Algorithms*, 4(4), August 2008. doi:10.1145/1383369.1383381.

15 Claire Mathieu and Hang Zhou. A PTAS for Capacitated Vehicle Routing on Trees. *ACM Transactions on Algorithms*, 19(2):17:1–17:28, 2023. doi:10.1145/3575799.

16 Runjie Miao and Jinjiang Yuan. A note on LP-based approximation algorithms for capacitated facility location problem. *Theoretical Computer Science*, 932:31–40, 2022. doi:10.1016/j.tcs.2022.08.002.

17 R. Ravi and Amitabh Sinha. Approximation Algorithms for Problems Combining Facility Location and Network Design. *Operations Research*, 54(1):73–81, 2006. URL: https://EconPapers.repec.org/RePEc:inm:oropre:v:54:y:2006:i:1:p:73-81.

18 Kunal Talwar. Bypassing the Embedding: Algorithms for Low Dimensional Metrics. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 281–290, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007399.

19 Lijiangnan Tian. Approximation Schemes for the Airport and Railway Problem. Master's thesis, Department of Computing Science, Faculty of Science, University of Alberta, 2023.

20 Vera Traub and Thorben Tröbst. A Fast $(2 + \frac{2}{7})$-Approximation Algorithm for Capacitated Cycle Covering. *Mathematical Programming*, 192(1):497–518, 2022. doi:10.1007/S10107-021-01678-3.

21 Wei Yu, Zhaohui Liu, and Xiaoguang Bao. New Approximation Algorithms for the Minimum Cycle Cover Problem. *Theoretical Computer Science*, 793:44–58, 2019. doi:10.1016/J.TCS.2019.04.009.