

17th Scandinavian Symposium and Workshops on Algorithm Theory

SWAT 2020, June 22–24, 2020, Tórshavn, Faroe Islands

Edited by

Susanne Albers



Editors

Susanne Albers

Department of Computer Science, Technical University of Munich, 85748 Garching, Germany
albers@in.tum.de

ACM Classification 2012

Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-150-4

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/978-3-95977-150-4>.

Publication date

June, 2020

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>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):

<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.SWAT.2020.0

ISBN 978-3-95977-150-4

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.

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Susanne Albers</i>	0:ix

Invited Talks

Parameterized Complexity of PCA	
<i>Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov</i>	1:1–1:5
Landscape of Locality	
<i>Jukka Suomela</i>	2:1–2:1

Regular Papers

Preclustering Algorithms for Imprecise Points	
<i>Mohammad Ali Abam, Mark de Berg, Sina Farahzad, Mir Omid Haji Mirsadeghi, and Morteza Saghaforoush</i>	3:1–3:12
Parameter Analysis for Guarding Terrains	
<i>Akanksha Agrawal, Sudeshna Kolay, and Meirav Zehavi</i>	4:1–4:18
Vertex Downgrading to Minimize Connectivity	
<i>Hassene Aissi, Da Qi Chen, and R. Ravi</i>	5:1–5:15
Sea-Rise Flooding on Massive Dynamic Terrains	
<i>Lars Arge, Mathias Rav, Morten Revsbæk, Yujin Shin, and Jungwoo Yang</i>	6:1–6:19
Computing β -Stretch Paths in Drawings of Graphs	
<i>Esther M. Arkin, Faryad Darabi Sahneh, Alon Efrat, Fabian Frank, Radoslav Fulek, Stephen Kobourov, and Joseph S. B. Mitchell</i>	7:1–7:20
Submodular Clustering in Low Dimensions	
<i>Arturs Backurs and Sarel Har-Peled</i>	8:1–8:14
Kernelizing the Hitting Set Problem in Linear Sequential and Constant Parallel Time	
<i>Max Bannach, Malte Skambath, and Till Tantau</i>	9:1–9:16
Graph Realizations: Maximum Degree in Vertex Neighborhoods	
<i>Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz</i>	10:1–10:17
A Dynamic Space-Efficient Filter with Constant Time Operations	
<i>Ioana O. Bercea and Guy Even</i>	11:1–11:17
A Simple Algorithm for Minimum Cuts in Near-Linear Time	
<i>Nalin Bhardwaj, Antonio J. Molina Lovett, and Bryce Sankar</i>	12:1–12:18
Parameterized Study of Steiner Tree on Unit Disk Graphs	
<i>Sujoy Bhore, Paz Carmi, Sudeshna Kolay, and Meirav Zehavi</i>	13:1–13:18
Bounded-Angle Minimum Spanning Trees	
<i>Ahmad Biniaz, Prosenjit Bose, Anna Lubiw, and Anil Maheshwari</i>	14:1–14:22

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).
Editor: Susanne Albers



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Low-Stretch Spanning Trees of Graphs with Bounded Width <i>Glencora Borradaile, Erin Wolf Chambers, David Eppstein, William Maxwell, and Amir Nayyeri</i>	15:1–15:19
Parameterized Complexity of Two-Interval Pattern Problem <i>Prosenjit Bose, Saeed Mehrabi, and Debajyoti Mondal</i>	16:1–16:10
Recoloring Interval Graphs with Limited Recourse Budget <i>Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, and Anna Zych-Pawlewicz</i>	17:1–17:23
Optimal Randomized Group Testing Algorithm to Determine the Number of Defectives <i>Nader H. Bshouty, Catherine A. Haddad-Zaknoon, Raghd Boulos, Foad Moalem, Jalal Nada, Elias Noufi, and Yara Zaknoon</i>	18:1–18:12
On the Hardness of Computing an Average Curve <i>Kevin Buchin, Anne Driemel, and Martijn Struijs</i>	19:1–19:19
Sparse Regression via Range Counting <i>Jean Cardinal and Aurélien Ooms</i>	20:1–20:17
Drawing Graphs with Circular Arcs and Right-Angle Crossings <i>Steven Chaplick, Henry Förster, Myroslav Kryven, and Alexander Wolff</i>	21:1–21:14
Clustering Moving Entities in Euclidean Space <i>Stephane Durocher and Md Yeakub Hassan</i>	22:1–22:14
Trajectory Visibility <i>Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals</i>	23:1–23:22
Simplifying Activity-On-Edge Graphs <i>David Eppstein, Daniel Frishberg, and Elham Havvaei</i>	24:1–24:14
Generalized Metric Repair on Graphs <i>Chenglin Fan, Anna C. Gilbert, Benjamin Raichel, Rishi Sonthalia, and Gregory Van Buskirk</i>	25:1–25:22
Maximum Edge-Colorable Subgraph and Strong Triadic Closure Parameterized by Distance to Low-Degree Graphs <i>Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz</i>	26:1–26:17
Preprocessing Vertex-Deletion Problems: Characterizing Graph Properties by Low-Rank Adjacencies <i>Bart M. P. Jansen and Jari J. H. de Kroon</i>	27:1–27:15
Locality Sensitive Hashing for Set-Queries, Motivated by Group Recommendations <i>Haim Kaplan and Jay Tenenbaum</i>	28:1–28:18
Fast Multi-Subset Transform and Weighted Sums over Acyclic Digraphs <i>Mikko Koivisto and Antti Röyskö</i>	29:1–29:12
Exact Exponential Algorithms for Two Poset Problems <i>László Kozma</i>	30:1–30:15
Space-Efficient Data Structures for Lattices <i>J. Ian Munro, Bryce Sandlund, and Corwin Sinnamon</i>	31:1–31:22

Online Embedding of Metrics <i>Ilan Newman and Yuri Rabinovich</i>	32:1–32:13
Primal-Dual 2-Approximation Algorithm for the Monotonic Multiple Depot Heterogeneous Traveling Salesman Problem <i>S. Rathinam, R. Ravi, J. Bae, and K. Sundar</i>	33:1–33:13
On the Parameterized Complexity of Grid Contraction <i>Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale</i>	34:1–34:17
Simplification of Polyline Bundles <i>Joachim Spoerhase, Sabine Storandt, and Johannes Zink</i>	35:1–35:20
Quantum Algorithm for Finding the Optimal Variable Ordering for Binary Decision Diagrams <i>Seiichiro Tani</i>	36:1–36:19

■ Preface

The Scandinavian Symposium and Workshops on Algorithm Theory (SWAT, formerly the Scandinavian Workshop on Algorithm Theory) has been held every two years beginning in 1988. It alternates with its sister conference, the Algorithms and Data Structures Symposium (WADS), which is usually offered in Canada. This year marks the 17th SWAT, and for the first time the symposium is hosted on Faroe Islands.

A total of 118 papers were submitted to the conference, four of which were withdrawn. Among the remaining 114 papers, the program committee selected 34 for presentation at the conference. In addition, the conference program featured two invited lectures, given by Fedor Fomin (University of Bergen) and Jukka Suomela (Aalto University). Accompanying extended abstracts are also contained in the proceedings.

I would like to thank the program committee and the subreviewers for their great effort. For almost all of the papers, extensive and detailed evaluations were submitted. The program committee consisted of Peyman Afshani (Aarhus University), Susanne Albers (chair; Technical University of Munich), Per Austrin (KTH Royal Institute of Technology), Sayan Bhattacharya (University of Warwick), Joan Boyar (University of Southern Denmark), Parinya Chalermsook (Aalto University), Timothy Chan (University of Illinois at Urbana-Champaign), Faith Ellen (University of Toronto), Travis Gagie (Dalhousie University), Naveen Garg (Indian Institute of Technology Delhi), Fabrizio Grandoni (IDSIA, University of Lugano), Roberto Grossi (University of Pisa), Pinar Heggernes (University of Bergen), Thore Husfeldt (Lund University and IT University of Copenhagen), Telikepalli Kavitha (Tata Institute of Fundamental Research), Yusuke Kobayashi (Kyoto University), Kasper Green Larsen (Aarhus University), Daniel Lokshantov (University of California, Santa Barbara), Benjamin Moseley (Carnegie Mellon University), Wolfgang Mulzer (Freie Universität Berlin), Seth Pettie (University of Michigan), Michał Pilipczuk (University of Warsaw), Hadas Shachnai (Technion), Michiel Smid (Carleton University), Philipp Woelfel (University of Calgary), Christian Wulff-Nilsen (University of Copenhagen), and Qin Xin (University of the Faroe Islands).

This year's conference was organized by Qin Xin (University of the Faroe Islands) and his team.

The SWAT conference series is guided by a steering committee consisting of Lars Arge (Aarhus University), Magnús M. Halldórsson (chair; Reykjavík University), Andrzej Lingas (Lund University), Jan Arne Telle (University of Bergen), and Esko Ukkonen (University of Helsinki).



Parameterized Complexity of PCA

Fedor V. Fomin

Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no

Petr A. Golovach

Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Kirill Simonov

Department of Informatics, University of Bergen, Norway
Kirill.Simonov@uib.no

Abstract

We discuss some recent progress in the study of Principal Component Analysis (PCA) from the perspective of Parameterized Complexity.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases parameterized complexity, Robust PCA, outlier detection

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.1

Category Invited Talk

Funding This work is supported by the Research Council of Norway via the project “MULTIVAL”.

1 Introduction

Worst-case running time analysis has been at the center of nearly all developments in theoretical computer science since the inception of the field. Nevertheless, the worst-case approach to measure algorithm efficiency has a serious drawback: For many fundamental problems it does not provide a reasonable explanation why in real life situations these problems are efficiently solvable. The dramatic gap between theory and practice calls for a more nuanced approach, beyond the worst-case case algorithmic analysis. The forthcoming book edited by Tim Roughgarden [23] provides a comprehensive introduction to this emerging area of algorithms.

A particularly successful attempt of building a mathematical model improving over worst-case analysis for *NP-hard* problems is the field of *parameterized complexity*. Originating in the late 80s from the foundational work of Downey and Fellows [7], parameterized complexity has experienced tremendous growth, and is now considered to be one of the central subfields of theoretical computer science, with several textbooks [8, 9, 11, 21], including the most recent book on parameterized algorithms [5] and kernelization [13].

However, so far the mainstream of parameterized complexity was devoted to the study of with NP-hard optimization problems, mostly on graphs and networks. In this talk we want to discuss the applicability of parameterized complexity to the problems involving data point, vectors and matrices.



© Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 1; pp. 1:1–1:5

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Robust PCA

Classical *principal component analysis* (PCA) is one of the most popular and successful techniques used for dimension reduction in data analysis and machine learning [22, 19, 10]. In PCA one seeks the best low-rank approximation of data matrix M by solving

$$\begin{aligned} & \text{minimize } \|M - L\|_F^2 \\ & \text{subject to } \text{rank}(L) \leq r. \end{aligned}$$

Here $\|A\|_F^2 = \sum_{i,j} a_{ij}^2$ is the square of the Frobenius norm of matrix A . By the Eckart-Young theorem [10], PCA is efficiently solvable via Singular Value Decomposition (SVD). PCA is used as a preprocessing step in a great variety of modern applications including face recognition, data classification, and analysis of social networks. A well-documented drawback of PCA is its vulnerability to noise. Even when a small number of observations is corrupted, like a few elements or columns of matrix M are changed, PCA of M may not reveal any reasonable information about non-corrupted observations.

There is a large class of extensively studied various robust PCA problems, see e.g. [26, 28, 2]. In the robust PCA setting we observe a noisy version M of data matrix L whose principal components we have to discover. In the case when M is a “slightly” disturbed version of L , PCA performed on M provides a reasonable approximation for L . However, when M is very “noisy” version of L , like being corrupted by a few outliers, even one corrupted outlier can arbitrarily alter the quality of the approximation. Unfortunately, almost every natural mathematical model of robust PCA leads to an NP-hard computational problem, and hence computationally intractable from the perspective of the classical worst-case analysis.

One of the popular approaches to robust PCA, is to model outliers as additive sparse matrix. Thus we have a data $d \times n$ matrix M , which is the superposition of a low-rank component L and a sparse component S . That is, $M = L + S$. This approach became popular after the works of Candès et al. [3], Wright et al. [27], and Chandrasekaran et al. [4]. A significant body of work on the robust PCA problem has been centered around proving that, under some feasibility assumptions on M , L , and S , a solution to

$$\begin{aligned} & \text{minimize} && \text{rank}(L) + \lambda \|S\|_0 \\ & \text{subject to} && M = L + S, \end{aligned} \tag{1}$$

where $\|S\|_0$ denotes the number of non-zero entries in matrix S and λ is a regularizing parameter, recovers matrix L uniquely. While optimization problem (1) is NP-hard [15], it is possible to show that under certain assumptions on L and S , its convex relaxation can recover these matrices efficiently.

The problem strongly related to (1) was studied in computational complexity under the name MATRIX RIGIDITY [16, 17, 25]. Here, for a given matrix M , and integers r and k , the task is to decide whether at most k entries of M can be changes so that the rank of the resulting matrix is at most r . Equivalently, this is the problem to decide whether a given matrix $M = L + S$, where $\text{rank}(L) \leq r$ and $\|S\|_0 \leq k$. Thus we define the following problem.

ROBUST PCA

Input: Data matrix $M \in \mathbb{R}^{n \times d}$, integer parameters r and k .
Task: Decide whether there are $L, S \in \mathbb{R}^{n \times d}$, $\text{rank}(L) \leq r$ and $\|S\|_0 \leq k$, such that $M = L + S$.

We first look at ROBUST PCA from the perspective of parameterized complexity and discuss when the problem is tractable and when it is not.

► **Theorem 1** ([12]). *ROBUST PCA is solvable in time $2^{\mathcal{O}(r \cdot k \cdot \log(r \cdot k))} \cdot (nd)^{\mathcal{O}(1)}$.*

The proof of the theorem requires ideas from kernelization, linear algebra and algebraic geometry. Thus ROBUST PCA is fixed-parameter tractable when parameterized by $k + d$. It is also worth to note that the theorem is tight in the following sense: The problem is NP-hard for every $r \geq 1$ [14, 6] and is W[1]-hard parameterized by k [12].

3 PCA with Outliers

Another popular variant of robust PCA is PCA with outliers. Suppose that we have n points (observations) in d -dimensional space. We know that a part of the points are arbitrarily located (say, produced by corrupted observations) while the remaining points are close to an r -dimensional true subspace. We do not have any information about the true subspace and about the corrupted observations. Our task is to learn the true subspace and to identify the outliers. As a common practice, we collect the points into $n \times d$ matrix M , thus each of the rows of M is a point and the columns of M are the coordinates.

Xu et al. [28] introduced the following idealization of this problem.

$$\begin{aligned} & \text{minimize} && \text{rank}(L) + \lambda \|S\|_{0,r} \\ & \text{subject to} && M = L + S. \end{aligned} \tag{2}$$

Here $\|S\|_{0,r}$ denotes the number of non-zero rows in matrix S and λ is a regularizing parameter. Xu et al. [28] approached this problem by building its convex surrogate and applying efficient convex optimization-based algorithm for the surrogate. A huge body of work exists on a variant of this problem, called ROBUST SUBSPACE RECOVERY, see e.g. [20] for a survey. In this problem for the set of given n points in r -dimensional space, the task is to find an r -dimensional subspace containing the maximum number of points. Hardt and Moitra [18] prove non-approximability of the optimization version of ROBUST SUBSPACE RECOVERY under Small Set Expansion conjecture.

An approximation variant of (2) and of ROBUST SUBSPACE RECOVERY is the following problem. Given n points in \mathbb{R}^d , we seek for a set of k points whose removal leaves the remaining $n - k$ points as close as possible to some r -dimensional subspace. Here is the reformulation of the problem in terms of matrices.

PCA WITH OUTLIERS

Input: Data matrix $M \in \mathbb{R}^{n \times d}$, integer parameters r and k .

Task:

$$\begin{aligned} & \text{minimize} && \|M - L - S\|_F^2 \\ & \text{subject to} && L, S \in \mathbb{R}^{n \times d}, \\ & && \text{rank}(L) \leq r, \text{ and} \\ & && S \text{ has at most } k \text{ non-zero rows.} \end{aligned}$$

We will see how the tools from Real Algebraic Geometry [1] can be used to prove the following theorem.

► **Theorem 2** ([24]). *Solving PCA WITH OUTLIERS is reducible to solving $n^{\mathcal{O}(d^2)}$ instances of PCA.*

We also discuss some lower bounds for PCA WITH OUTLIERS.

References

- 1 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.
- 2 Thierry Bouwmans, Necdet Serhat Aybat, and El-hadi Zahzah. *Handbook of robust low-rank and sparse matrix decomposition: Applications in image and video processing*. Chapman and Hall/CRC, 2016.
- 3 Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3):11:1–11:37, 2011. doi:10.1145/1970392.1970395.
- 4 Venkat Chandrasekaran, Sujay Sanghavi, Pablo A. Parrilo, and Alan S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21(2):572–596, 2011. doi:10.1137/090761793.
- 5 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.
- 6 Chen Dan, Kristoffer Arnsfelt Hansen, He Jiang, Liwei Wang, and Yuchen Zhou. On low rank approximation of binary matrices. *CoRR*, abs/1511.01699, 2015. arXiv:1511.01699.
- 7 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *Proceedings of the 21st Manitoba Conference on Numerical Mathematics and Computing. Congr. Numer.*, 87:161–178, 1992.
- 8 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.
- 9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 10 Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.
- 11 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.
- 12 Fedor V. Fomin, Daniel Lokshtanov, Syed Mohammad Meesum, Saket Saurabh, and Meirav Zehavi. Matrix rigidity from the viewpoint of parameterized complexity. *SIAM J. Discrete Math.*, 32(2):966–985, 2018. doi:10.1137/17M112258X.
- 13 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 14 Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1 -norm low-rank matrix approximation. *CoRR*, abs/1509.09236, 2015. URL: <http://arxiv.org/abs/1509.09236>, arXiv:1509.09236.
- 15 Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1 -norm low-rank matrix approximation. *Math. Oper. Res.*, 43(4):1072–1084, 2018. doi:10.1287/moor.2017.0895.
- 16 Dmitry Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity (in russian). *Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka*, 1976.
- 17 Dmitry Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity. *Journal of Soviet Math.*, 14(5):1450–1456, 1980.
- 18 Moritz Hardt and Ankur Moitra. Algorithms and hardness for robust subspace recovery. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*, volume 30 of *JMLR Proceedings*, pages 354–375. JMLR.org, 2013.
- 19 Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.
- 20 Gilad Lerman and Tyler Maunu. An overview of robust subspace recovery. *Proceedings of the IEEE*, 106(8):1380–1410, 2018.

- 21 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.
- 22 Karl Pearson. LIII. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.
- 23 Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020.
- 24 Kirill Simonov, Fedor V. Fomin, Petr A. Golovach, and Fahad Panolan. Refined complexity of PCA with outliers. In *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, volume 97, pages 5818–5826. PMLR, 2019. URL: <http://proceedings.mlr.press/v97/simonov19a.html>.
- 25 L G Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, pages 162–176, 1977.
- 26 Namrata Vaswani and Praneeth Narayanamurthy. Static and dynamic robust PCA and matrix completion: A review. *Proceedings of the IEEE*, 106(8):1359–1379, 2018. doi:10.1109/JPROC.2018.2844126.
- 27 John Wright, Arvind Ganesh, Shankar R. Rao, YiGang Peng, and Yi Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In *Proceedings of 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2080–2088. Curran Associates, Inc., 2009. URL: <http://papers.nips.cc/paper/3704-robust-principal-component-analysis-exact-recovery-of-corrupted-low-rank-matrices-via-convex-optimization>.
- 28 Huan Xu, Constantine Caramanis, and Sujay Sanghavi. Robust PCA via outlier pursuit. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2496–2504. Curran Associates, Inc., 2010. URL: <http://papers.nips.cc/paper/4005-robust-pca-via-outlier-pursuit>.

Landscape of Locality

Jukka Suomela 

Aalto University, Finland

<https://jukkasuomela.fi/>

jukka.suomela@aalto.fi

Abstract

The *theory of distributed computing* aims at understanding which tasks can be solved efficiently in large distributed systems. This forms the basis for our understanding of the modern world, which heavily depends on world-wide communication networks and large-scale distributed computer systems.

In distributed computing the key computational resource is *communication*, and we seek to find out which computational problems can be solved with only a few communication steps. This is directly connected to the concept of *locality*: in T synchronous communication rounds, all nodes in a network can gather all information in their radius- T neighborhoods, but not any further. Hence the *distributed time complexity* of a graph problem can be defined in two equivalent ways: it is the number of communication rounds needed to solve the problem, and it is the distance up to which individual nodes need to see in order to choose their own part of the solution.

While the locality of graph problems has been studied already since the 1980s, only in the past four years we have started to take big leaps in understanding what the landscape of distributed time complexity looks like and with what kind of tools and techniques we can study it.

One concept that has been a driving force in the recent developments is the notion of *locally verifiable* problems. These are graph problems in which a solution is feasible if and only if it looks valid in all constant-radius neighborhoods; put otherwise, these are problems that could be solved efficiently with a *nondeterministic* distributed algorithm, and hence they form a natural distributed analogue of class NP. Now the key question is this: if a problem is locally verifiable, is it also locally solvable, and if not, what can we say about its distributed time complexity?

Naor and Stockmeyer [SIAM J. Comput. 1995] formalized the idea of locally verifiable problems by introducing the class of LCL problems (locally checkable labeling problems). While the concept is old, and over the years we have seen results related to the locality of many *specific* LCLs, little was known about the distributed complexity of LCLs *in general*. By 2015, we had only seen examples of LCLs with localities $O(1)$, $\Theta(\log^* n)$, and $\Theta(n)$, and it was wide open whether these three classes are all that there is.

All this started to change rapidly after we proved [Brandt et al., STOC 2016] that there are natural examples of LCLs that have a locality strictly between $\omega(\log^* n)$ and $o(n)$. The same paper also paved the way for the development of a new general-purpose proof technique for analyzing the locality of locally verifiable problems, namely *round elimination*.

Now after four years of work and a number of papers by several research teams working in the area, we have reached a point in which there is a *near-complete picture of the landscape of LCL problems* – and it looks nothing like what we would have expected.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Complexity classes

Keywords and phrases Theory of distributed computing, Network algorithms, Locality, Distributed time complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.2

Category Invited Talk



© Jukka Suomela;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preclustering Algorithms for Imprecise Points

Mohammad Ali Abam

Department of Computer Engineering, Sharif University of Technology, Iran
abam@sharif.edu

Mark de Berg 

Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands
m.t.d.berg@tue.nl

Sina Farahzad

Department of Computer Engineering, Sharif University of Technology, Iran
farahzad@ce.sharif.edu

Mir Omid Haji Mirsadeghi

Department of Mathematical Sciences, Sharif University of Technology, Iran
mirsadeghi@sharif.edu

Morteza Saghafian

Department of Mathematical Sciences, Sharif University of Technology, Iran
morteza.saghafian65@student.sharif.edu

Abstract

We study the problem of *preclustering* a set B of imprecise points in \mathbb{R}^d : we wish to cluster the regions specifying the potential locations of the points such that, no matter where the points are located within their regions, the resulting clustering approximates the optimal clustering for those locations. We consider k -center, k -median, and k -means clustering, and obtain the following results.

Let $B := \{b_1, \dots, b_n\}$ be a collection of disjoint balls in \mathbb{R}^d , where each ball b_i specifies the possible locations of an input point p_i . A partition \mathcal{C} of B into subsets is called an $(f(k), \alpha)$ -preclustering (with respect to the specific k -clustering variant under consideration) if (i) \mathcal{C} consists of $f(k)$ preclusters, and (ii) for any realization P of the points p_i inside their respective balls, the cost of the clustering on P induced by \mathcal{C} is at most α times the cost of an optimal k -clustering on P . We call $f(k)$ the *size* of the preclustering and we call α its *approximation ratio*. We prove that, even in \mathbb{R}^1 , one may need at least $3k - 3$ preclusters to obtain a bounded approximation ratio – this holds for the k -center, the k -median, and the k -means problem – and we present a $(3k, 1)$ preclustering for the k -center problem in \mathbb{R}^1 . We also present various preclusterings for balls in \mathbb{R}^d with $d \geq 2$, including a $(3k, \alpha)$ -preclustering with $\alpha \approx 13.9$ for the k -center and the k -median problem, and $\alpha \approx 254.7$ for the k -means problem.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Geometric clustering, k -center, k -means, k -median, imprecise points, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.3

Funding The work in this paper is supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

Clustering is one of the most important and widely studied problems in unsupervised learning. It comes in many different flavors, depending on the type of data to be clustered, the measure used to assess the quality of a clustering, and so on. In this paper we are interested in geometric clustering, where the data are points in \mathbb{R}^d , and we consider three well-known centroid-based clustering methods, namely k -center, k -median, and k -means, on so-called imprecise points.



© Mohammad Ali Abam, Mark de Berg, Sina Farahzad, Mir Omid Haji Mirsadeghi, and Morteza Saghafian;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 3; pp. 3:1–3:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3:2 Preclustering Algorithms for Imprecise Points

In (the geometric version of) centroid-based clustering one is given a set P of n points in \mathbb{R}^d , where d is a fixed constant, and an integer k . The goal is to partition P into k subsets P_1, \dots, P_k and assign a centroid q_i to each cluster P_i such that the cost of the resulting clustering is minimized. In the k -center problem the cost of the clustering is defined as $\max_{1 \leq i \leq k} \max_{p \in P_i} |pq_i|$, where $|pq|$ denotes the Euclidean distance between two points p and q . In the k -median problem the cost of a clustering is defined as $\sum_{1 \leq i \leq k} \sum_{p \in P_i} |pq_i|$, and in the k -means problem it is defined as $\sum_{1 \leq i \leq k} \sum_{p \in P_i} |pq_i|^2$. Given a collection of centroids it is always optimal to define the clusters by assigning each point in P to its nearest centroid. Thus an equivalent definition of the k -center problem, for instance, is to find a collection of $\{q_1, \dots, q_k\}$ as centroids that minimizes $\max_{p \in P} \min_{1 \leq i \leq k} |pq_i|$. In other words, we want to find k congruent balls of minimum radius that together cover all points in P .

The k -center problem in \mathbb{R}^d is NP-hard for $d \geq 2$ when k is part of the input. For the Euclidean k -center problem a PTAS exists, as shown by Agarwal and Procopiuc [1]. (For the k -center problem in general metric spaces, a PTAS does not exist; for this case an r -approximation algorithm with $r < 2$ is not possible unless $P=NP$, and several 2-approximation algorithms are known [5, 14].) The k -median and k -means problems are also NP-hard for $d \geq 2$, and they admit a PTAS as well [2, 4, 6, 8].

In the traditional setting the locations of the input points are known exactly. In practice this may not always be the case: typically locations are measured using GPS or other devices that are not completely accurate, or the points may move around inside a given region. This leads to the study of geometric algorithms on so-called *imprecise points*. Here, instead of specifying the exact coordinates of each input point, we specify a region for each point where it may be located. For points in the plane the regions are typically disks or squares. Over the past decade, many problems have been studied for imprecise points, including convex hulls (compute the smallest (or largest) possible convex hull of a set of imprecise points [7, 11]), Delaunay triangulations (preprocess a set of imprecise points such that for any given instantiation of the points in the given regions we can compute the Delaunay triangulation quickly [3]), separability problems [13], and more [9, 10, 12].

Problem statement and notation. In this paper we study the k -center, k -median, and k -means problem for imprecise points. The input is a set $B := \{b_1, \dots, b_n\}$ of (closed) balls in \mathbb{R}^d , each representing the possible locations of an input point. Our goal is to compute a *preclustering* of the imprecise points, that is, a partition of B into a collection \mathcal{C} of subsets called *preclusters* that gives a good clustering for any possible realization of the points inside the input balls. Next we define this more formally.

For a (precise) point set P , let $\text{OPT}_\infty(P, k)$ denote the cost of an optimal k -center clustering on P , that is,

$$\text{OPT}_\infty(P, k) := \min_{q_1, \dots, q_k \in \mathbb{R}^d} \max_{p \in P} \min_{1 \leq i \leq k} |pq_i|.$$

The cost of an optimal solution for the k -median and k -means problem on a set P are denoted by $\text{OPT}_1(P, k)$ and $\text{OPT}_2(P, k)$, respectively.¹ Now consider an imprecise point set specified by a set $B = \{b_1, \dots, b_n\}$ of balls. A point set $P := \{p_1, \dots, p_n\}$ such that $p_i \in b_i$ for all $1 \leq i \leq n$ is called a *B-instance*. A preclustering \mathcal{C} of the set B into preclusters B_i

¹ The subscript ∞ in OPT_∞ refers to the fact that if d_i denotes the distance of point $p_i \in P$ to its nearest center, then we are minimizing the norm of the vector $\langle d_1, \dots, d_n \rangle$ in the ℓ_∞ -metric. For k -median and k -means we are minimizing the norm in the ℓ_1 -metric and in the squared ℓ_2 -metric, respectively.

induces a clustering on any B -instance P in a natural manner, namely by creating a cluster $P_i := \{p \in P : p \in B_i\}$ for every precluster $B_i \in \mathcal{C}$. The cost of the preclustering \mathcal{C} on P , denoted by $\mathcal{C}\text{-COST}_\infty(P)$ for the k -center problem, is defined as the cost of the induced clustering on P if we choose the centroid of each cluster P_i optimally, namely by solving the 1-clustering problem on P_i . So for the k -center problem we have

$$\mathcal{C}\text{-COST}_\infty(P) := \max_{B_i \in \mathcal{C}} \min_{q \in \mathbb{R}^d} \max_{p \in P_i} |pq|.$$

The preclustering costs for the k -median and k -means problem are denoted by $\mathcal{C}\text{-COST}_1(P)$ and $\mathcal{C}\text{-COST}_2(P)$, respectively, and they are defined similarly. To quantify the quality of a preclustering \mathcal{C} on B (with respect to the k -clustering problem under consideration) we define \mathcal{C} to be a $(f(k), \alpha)$ -preclustering if

- \mathcal{C} consists of $f(k)$ preclusters,
- $\mathcal{C}\text{-COST}(P) \leq \alpha \cdot \text{OPT}(P, k)$ for any B -instance P .

We call $f(k)$ the *size* of the preclustering and we call α its *approximation ratio*. Ideally, we would like to have a $(k, 1)$ -preclustering, but this is not always possible. If the balls in B have a non-empty common intersection, then any preclustering with fewer than n preclusters may have an arbitrarily bad approximation ratio, even for the 2-center problem. Hence, we assume (as is often done in papers on imprecise points) that the balls in B are disjoint.

Our results. As mentioned, obtaining a $(k, 1)$ -preclustering is not always possible. This leads to the question: what is the smallest value for $f(k)$ such that we can always obtain an $(f(k), 1)$ -preclustering? More generally, which trade-offs are possible between the size $f(k)$ of the preclustering and its approximation ratio α ?

In Section 2 we study this problem in \mathbb{R}^1 . We show that there are input sets B that require at least $3k - 3$ preclusters to get a bounded approximation ratio; this holds for the k -center problem, the k -median problem, as well as the k -means problem. We complement this result by proving that any set B of intervals in \mathbb{R}^1 admits a $(3k, 1)$ -preclustering for the k -center problem. This preclustering can be computed in polynomial time.

In Section 3 we consider the d -dimensional version of the problem for $d \geq 2$. We give an example showing that here a $(3k, 1)$ -preclustering does not always exist, and we present a $(3k, \alpha)$ -preclustering with $\alpha \approx 13.9$ for the k -center and k -median problem, and $\alpha \approx 254.7$ for the k -means problem. A different parameterization of the strategy gives a $(6k, 3)$ -preclustering for k -center and k -median, and a $(6k, 10)$ -preclustering for k -means in \mathbb{R}^2 .

Finally, in Section 4 we obtain tight asymptotic bounds on the size of the preclustering needed to obtain any given approximation ratio $\varepsilon > 0$ for the k -center problem. In particular, we prove that $\Theta(\lceil 1/\varepsilon^d \rceil \cdot k)$ preclusters are always sufficient and sometimes necessary to obtain approximation ratio ε .

2 The 1-dimensional problem

We begin by proving that even in \mathbb{R}^1 – here the input balls are disjoint intervals on the line – preclusterings with only k preclusters cannot always guarantee a good approximation ratio. In fact, we sometimes need as much as $3k - 3$ preclusters in any preclustering with bounded approximation ratio.

► **Theorem 1.** *For any integer $k \geq 2$ and any given α , there is a set B of disjoint intervals in \mathbb{R}^1 that does not admit a (k', α) -preclustering with $k' < 3k - 3$. This holds for k -center, k -median, as well as k -means clustering.*

3:4 Preclustering Algorithms for Imprecise Points



■ **Figure 1** Illustration of the lower-bound construction for $k = 5$: a collection of $k - 1$ groups of three intervals (in grey), each group consisting of a left and right interval of length 1 separated by a gap of length ε , and a middle interval in inside this gap. The points in the B -instance used in the proof are shown slightly above intervals for clarity.

Proof. Let B be a collection of $3k - 3$ disjoint intervals in \mathbb{R}^1 consisting of $k - 1$ groups of three intervals each. The left and right interval in each group have length 1 and are at distance ε from each other, where ε is a sufficiently small number that will be specified later. The middle interval from the group lies in the gap between the left and right interval with its center at the center of the gap; see Fig. 1. Now consider a preclustering $\mathcal{C} = \{B_1, \dots, B_{k'}\}$. If $k' < 3k - 3$, then there is at least one precluster containing two intervals, b_i and b_j . Assume without loss of generality that $\text{length}(b_i) \geq \text{length}(b_j)$, and consider the B -instance in which each point p_t is placed in its interval $b_t \in B$ as follows.

- If $t = i$ or b_t is a middle interval, then p_t lies at the center of b_t .
- If $t \neq i$ and b_t is a left interval, then p_t lies at the right endpoint of b_t .
- If $t \neq i$ and b_t is a right interval, then p_t lies at the left endpoint of b_t .

Note that with this placement we have $|p_i p_j| \geq 1/2$. We will argue that by choosing ε appropriately we get the desired result.

First consider the k -center problem. Note that $\text{OPT}_\infty(P, k) \leq \varepsilon/2$. Indeed, by putting a centroid at the center of each of the $k - 1$ gaps and one centroid at p_i , all points in P are at distance at most $\varepsilon/2$ from a centroid. On the other hand, $\mathcal{C}\text{-COST}_\infty(P) \geq 1/4$ since the centroid for the cluster containing p_i and p_j is at distance at least $1/4$ from p_i or p_j . Hence,

$$\frac{\mathcal{C}\text{-COST}_\infty(P)}{\text{OPT}_\infty(P, k)} \geq \frac{1/4}{\varepsilon/2} = \frac{1}{2\varepsilon}.$$

For $\varepsilon < 1/(2\alpha)$ we thus enforce an approximation ratio greater than α .

The argument for k -median and k -means is similar. For k -median we have $\text{OPT}_1(P, k) \leq 2(k - 1)(\varepsilon/2)$ and $\mathcal{C}\text{-COST}_1(P) \geq 1/2$, so $\varepsilon < 1/(2(k - 1)\alpha)$ enforces an approximation ratio greater than α , while for k -means we have $\text{OPT}_2(P, k) \leq 2(k - 1)(\varepsilon/2)^2$ and $\mathcal{C}\text{-COST}_2(P) \geq 2(1/4)^2$, so $\varepsilon < \sqrt{1/(4(k - 1)\alpha)}$ suffices. ◀

► **Remark 2.** The construction in the proof of Theorem 1 uses an input set B of size $3k - 3$. We can easily generate an input set with the same behavior for any $n \geq 3k - 3$, by adding another $n - 3k + 3$ tiny intervals inside one of the gaps between a left and a right interval from the same group.

Theorem 1 states that for some problem instances any preclustering with fewer than $3k - 3$ preclusters has arbitrarily large approximation ratio. We now show how to obtain a 1-approximation with only $3k$ preclusters for the k -center problem. We assume from now on that $n > 3k$, otherwise we can trivially create a zero-cost solution with at most $3k$ preclusters.

Before we describe our preclustering strategy, we first generalize the k -center problem in \mathbb{R}^1 from points to intervals. In this generalization the input is a collection B of n intervals, and the goal is to find a collection $\mathcal{I} := \{I_1, \dots, I_k\}$ of intervals that together cover all intervals in B and such that the maximum radius of the intervals in \mathcal{I} is minimized. (The radius of an interval is half its length.) We denote the value of an optimal solution \mathcal{I} to the k -center problem on B by $\text{OPT}_\infty(B, k)$, so $\text{OPT}_\infty(B, k) := \max_{I_i \in \mathcal{I}} \text{radius}(I_i)$.

Our preclustering algorithm is now as follows.

PRECLUSTERING-1D(B, k)

1. Sort the intervals in B by radius, such that $\text{radius}(b_1) \geq \dots \geq \text{radius}(b_n)$.
2. For each $k' \in \{0, \dots, 2k\}$ do the following.
 - a. Let $\{B_1, \dots, B_{(3k-k')}\}$ be an optimal $(3k - k')$ -center clustering on $\{b_{k'+1}, \dots, b_n\}$, and let $\text{OPT}_\infty(\{b_{k'+1}, \dots, b_n\}, 3k - k')$ be its cost.
 - b. Let $\mathcal{C}(k')$ be the preclustering $\{\{b_1\}, \dots, \{b_{k'}\}, B_1, \dots, B_{(3k-k')}\}$.
3. Of all preclusterings $\mathcal{C}(0), \dots, \mathcal{C}(2k)$ found in Step 2, let $\mathcal{C}(k')$ be the one that minimizes $\text{OPT}_\infty(\{b_{k'+1}, \dots, b_n\}, 3k - k')$. Let $\mathcal{C} := \mathcal{C}(k')$ and return \mathcal{C} .

► **Theorem 3.** *Any set B of disjoint intervals in \mathbb{R}^1 admits a $(3k, 1)$ -preclustering for the k -center problem and this algorithm can be executed in polynomial time.*

Proof. Obviously PRECLUSTERING-1D(B, k) gives a preclustering \mathcal{C} with $3k$ preclusters. It remains to prove that \mathcal{C} has approximation ratio 1. Let P be a B -instance, and let $Q \in \{q_1, \dots, q_k\}$ be an optimal set of centroids for the k -center problem on P . Thus by placing an interval of radius $\text{OPT}_\infty(P, k)$ centered at each centroid $q_i \in Q$, we cover all points in P . By assigning each point in P to its nearest centroid in Q , with ties broken arbitrarily, we obtain a partition of P into k clusters. This partition induces a preclustering \mathcal{C}^* of size k on B . We use \mathcal{C}^* to define two types of intervals: *outer intervals*, which are the leftmost or rightmost interval in any of the preclusters $B_i \in \mathcal{C}^*$, and *inner intervals*, which are the remaining intervals. Note that the number of outer intervals is at most $2k$. Define k^* as the largest k' such that $b_1, \dots, b_{k'}$ are all outer intervals, where b_1, \dots, b_n is the sorted set of intervals obtained in Step 1 of the algorithm. Since b_{k^*+1} is an inner interval, we have

$$\text{OPT}_\infty(P, k) \geq \text{radius}(b_{k^*+1}). \quad (1)$$

The preclustering $\mathcal{C} := \mathcal{C}(k')$ returned by our algorithm minimizes $\text{OPT}_\infty(\{b_{k'+1}, \dots, b_n\}, 3k - k')$. Note that $\mathcal{C}\text{-COST}_\infty(P) \leq \text{OPT}_\infty(\{b_{k'+1}, \dots, b_n\}, 3k - k')$, since the intervals $b_1, \dots, b_{k'}$ are all in singleton preclusters and an interval covering all intervals in a precluster B_i obviously covers all points from P in those interval. Hence,

$$\mathcal{C}\text{-COST}_\infty(P) \leq \text{OPT}_\infty(\{b_{k^*+1}, \dots, b_n\}, 3k - k^*).$$

It remains to argue that $\text{OPT}_\infty(P, k) \geq \text{OPT}_\infty(\{b_{k^*+1}, \dots, b_n\}, 3k - k^*)$. To this end, we create a collection \mathcal{I} of intervals as follows.

- For each outer interval b_j with $j > k^*$ we create an interval equal to b_j .
- For each precluster $B_i \in \mathcal{C}^*$ that has at least one inner interval, we create a minimum-length interval covering all inner intervals of B_i .

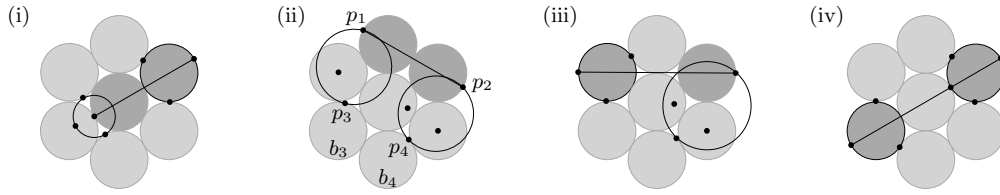
Note that \mathcal{I} contains at most $3k - k^*$ intervals, and that these intervals together cover all intervals in $\{b_{k^*+1}, \dots, b_n\}$. Hence,

$$\max_{I \in \mathcal{I}} \text{radius}(I) \geq \text{OPT}_\infty(\{b_{k^*+1}, \dots, b_n\}, 3k - k^*).$$

Moreover, $\text{OPT}_\infty(P, k) \geq \text{radius}(I)$ for any $I \in \mathcal{I}$. Indeed, if I is equal to an outer interval b_j with $j > k^*$ then $\text{OPT}_\infty(P, k) \geq \text{radius}(b_j)$ by Inequality (1), and otherwise I is the minimum-length interval covering all inner intervals of some precluster B_i . (In the latter case we also have $\text{OPT}_\infty(P, k) \geq \text{radius}(I)$ because in any B -instance the cluster of B_i includes a point in both outer intervals) We conclude that

$$\text{OPT}_\infty(P, k) \geq \max_{I \in \mathcal{I}} \text{radius}(I) \geq \text{OPT}_\infty(\{b_{k^*+1}, \dots, b_n\}, 3k - k^*).$$

3:6 Preclustering Algorithms for Imprecise Points



■ **Figure 2** The seven balls shown in the figure do not admit a $(3k, 1)$ -preclustering for $k = 2$.

It remains to argue that $\text{PRECLUSTERING-1D}(B, k)$ can be implemented to run in polynomial time. The most time-consuming step is Step 2a, which can be implemented to run in $O(n^2k)$ time using dynamic programming in a straightforward manner. ◀

Theorem 3 only holds for the k -center problem. In the next section we present a more general algorithm, which not only works in higher dimensions but also for k -median and k -means. The approximation ratio will not be as good as the one provided by Theorem 2, however.

3 The d -dimensional problem

In the previous section we saw that for some problem instances any preclustering with fewer than $3k - 3$ preclusters has an arbitrarily large approximation ratio. The result is stated for \mathbb{R}^1 but it also holds in \mathbb{R}^d for $d > 1$: we can use exactly the same construction, replacing the intervals by d -dimensional balls whose centers lie on the x_1 -axis. We also presented an algorithm giving a $(3k, 1)$ -preclustering for intervals in \mathbb{R}^1 , for the k -center problem.

Fig. 2 shows that a $(3k, 1)$ -preclustering is not always possible for the k -center problem in \mathbb{R}^2 . The figure shows a set B of seven unit balls, with one central ball touching the other six balls. For $k = 2$ a preclustering of size $3k$ would use five singleton preclusters and one precluster with two balls. There are four combinatorially distinct ways of choosing the precluster of two balls, indicated by the dark grey balls in parts (i)–(iv) of the figure. For each case, a B -instance is shown (the black dots), and the optimal solution to the 2-center problem for the instance is shown (the two black circles). The best preclustering is the one in part (ii). Here the two points p_1, p_2 in the dark grey balls are placed at distance 4 from each other, so $\mathcal{C}\text{-COST}_\infty(P) = 2$. The point p_3 inside the ball b_3 is placed as close to p_1 as possible, while p_4 is placed as close to p_2 as possible. The other points are placed such that they are either contained in the ball with diameter p_1p_3 or in the ball with diameter p_2p_4 . Hence, $\text{OPT}_\infty(P) = (\sqrt{13} - 1)/2$. The balls in this construction are not disjoint, but we can scale them by a factor $(1 - \varepsilon)$ to obtain an instance where any $(3k, \alpha)$ -preclustering has $\alpha \geq 2/((\sqrt{13} - 1)/2) = 4/(\sqrt{13} - 1) \approx 1.54$.

We now present a preclustering strategy that works for k -center, k -means and k -median in any dimension. It is similar to, and actually somewhat simpler than, the preclustering algorithm we presented for the 1-dimensional k -center problem.

$\text{PRECLUSTERING-DD}(B, k)$

1. Sort the balls in B by radius, such that $\text{radius}(b_1) \geq \dots \geq \text{radius}(b_n)$.
2. Define $B_{\text{small}} := \{b_{2k+1}, \dots, b_n\}$; we call the balls in B_{small} *small*. Let $\{P_1, \dots, P_k\}$ be an optimal k -center (or k -median, or k -means) clustering on the point set $\text{centers}(B_{\text{small}}) := \{c_j : 2k + 1 \leq j \leq n\}$, where c_j is the center of the ball b_j . Let $\{B_1, \dots, B_k\}$ be the preclustering on B_{small} induced by it.
3. Return the preclustering $\mathcal{C} := \{\{b_1\}, \dots, \{b_{2k}\}, B_1, \dots, B_k\}$.

Before we analyze the algorithm's approximation ratio, we note that, depending on the dimension d and the value of k , we may not be able to implement Step 2 efficiently. However, instead of computing an optimal k -clustering on the centers of the small balls, we can also compute a $(1 + \varepsilon')$ -approximation of the optimal clustering. For an appropriate $\varepsilon' = O(\varepsilon)$ this increases the approximation ratio by only a factor $1 + \varepsilon$, as explained later.

Obviously $\text{PRECLUSTERING-DD}(B, k)$ gives a preclustering of size $3k$. To analyze the approximation ratio, we use the following lemma.

► **Lemma 4.** *For any B -instance P the preclustering $\mathcal{C} := \{\{b_1\}, \dots, \{b_{2k}\}, B_1, \dots, B_k\}$ computed by the algorithm satisfies:*

- (i) $\mathcal{C}\text{-COST}_\infty(P) \leq \text{OPT}_\infty(P, k) + 2 \cdot \text{radius}(b_{2k+1})$
- (ii) $\mathcal{C}\text{-COST}_1(P) \leq \text{OPT}_1(P, k) + 2 \sum_{j=2k+1}^n \text{radius}(b_j)$
- (iii) $\mathcal{C}\text{-COST}_2(P) \leq 4 \cdot \text{OPT}_2(P, k) + 6 \sum_{j=2k+1}^n \text{radius}(b_j)^2$.

Proof. We first prove part (i) of the lemma. Let P be any B -instance, let $p_j \in P$ denote the point inside b_j , and let c_j be the center of b_j . Recall that $P_i \subset P$ is the subset of points in the instance corresponding to the precluster B_i . Define $P_{\text{small}} := \{p_{2k+1}, \dots, p_n\}$ to be the set of points from P in the small balls, and define $C_{\text{small}} := \{c_{2k+1}, \dots, c_n\}$. Note that $P_{\text{small}} = P_1 \cup \dots \cup P_k$ and that

$$|p_j c_j| \leq \text{radius}(b_j) \leq \text{radius}(b_{2k+1}) \quad (2)$$

for all $p_j \in P_{\text{small}}$. We define the following sets of centroids:

- Let $Q := \{q_1, \dots, q_k\}$ be the set of centroids in an optimal k -center solution for the entire point set P . We have

$$\max_{p_j \in P_{\text{small}}} \min_{q_i \in Q} |p_j q_i| \leq \max_{p_j \in P} \min_{q_i \in Q} |p_j q_i| = \text{OPT}_\infty(P, k). \quad (3)$$

- Let $Q' := \{q'_1, \dots, q'_k\}$ be the set of centroids in the optimal k -center clustering on C_{small} used in Step 2 of the algorithm. Thus

$$\max_{c_i \in C_{\text{small}}} \min_{q'_j \in Q'} |c_i q'_j| = \text{OPT}_\infty(C_{\text{small}}, k) \leq \max_{c_i \in C_{\text{small}}} \min_{q_j \in Q} |c_i q'_j|. \quad (4)$$

- Let $Q'' := \{q''_1, \dots, q''_k\}$, where q''_i is the optimal centroid for P_i . Note that for all P_i we have

$$\max_{p_j \in P_i} |p_j q''_j| \leq \max_{p_j \in P_i} |p_j q'_j|. \quad (5)$$

Since the total cost of the singleton preclusters is trivially zero, we have

$$\begin{aligned} & \mathcal{C}\text{-COST}_\infty(P) \\ &= \max_{1 \leq i \leq k} \max_{p_j \in P_i} |p_j q''_i| \\ &\leq \max_{1 \leq i \leq k} \max_{p_j \in P_i} |p_j q'_i| && \text{(Inequality (5))} \\ &\leq \max_{1 \leq i \leq k} \max_{p_j \in P_i} (|p_j c_j| + |c_j q'_i|) && \text{(triangle inequality)} \\ &\leq \text{radius}(b_{2k+1}) + \max_{1 \leq i \leq k} \max_{p_j \in P_i} |c_j q'_i| && \text{(Inequality (2))} \\ &\leq \text{radius}(b_{2k+1}) + \max_{c_j \in C_{\text{small}}} \min_{q'_i \in Q'} |c_j q'_i| && \text{(definition of } C_{\text{small}}) \\ &\leq \text{radius}(b_{2k+1}) + \max_{c_j \in C_{\text{small}}} \min_{q_i \in Q} |c_j q_i| && \text{(Inequality (4))} \\ &\leq \text{radius}(b_{2k+1}) + \max_{p_j \in P_{\text{small}}} \min_{q_i \in Q} (|c_j p_j| + |p_j q_i|) && \text{(triangle inequality)} \\ &\leq 2 \cdot \text{radius}(b_{2k+1}) + \max_{p_j \in P_{\text{small}}} \min_{q_i \in Q} |p_j q_i| && \text{(Inequality (2))} \\ &\leq 2 \cdot \text{radius}(b_{2k+1}) + \text{OPT}_\infty(P, k) && \text{(Inequality (3))} \end{aligned}$$

3:8 Preclustering Algorithms for Imprecise Points

To prove part (ii) of the lemma, which deals with the k -median problem, note that Inequality (2) still holds while Inequalities (3)–(5) hold if we replace the max-operator by a summation. Part (ii) can thus be derived using a similar derivation as for part (i).

To prove part (iii), which deals with the k -means problem, we need to work with squared distances. Note that Inequality (2) still holds, while Inequalities (3)–(5) hold if we replace the max-operator with a summation and all distance values with their squared values. For squared distances the triangle inequality does not hold. Instead we use the Cauchy-Schwarz inequality, which implies that if a, b, c are positive reals with $a \leq b + c$, then $a^2 \leq 2b^2 + 2c^2$. A similar computation as above can now be used to prove part (iii), we have

$$\begin{aligned}
& \mathcal{C}\text{-COST}_2(P) \\
&= \sum_{i=1}^k \sum_{p_j \in P_i} |p_j q_i''|^2 \\
&\leq \sum_{i=1}^k \sum_{p_j \in P_i} |p_j q_i'|^2 && \text{(Inequality (5))} \\
&\leq \sum_{i=1}^k \sum_{p_j \in P_i} (2|p_j c_j|^2 + 2|c_j q_i'|^2) && \text{(Cauchy-Schwarz)} \\
&\leq 2 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 2 \sum_{i=1}^k \sum_{p_j \in P_i} |c_j q_i'|^2 && \text{(Inequality (2))} \\
&\leq 2 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 2 \sum_{c_j \in C_{\text{small}}} \min_{q_i' \in Q'} |c_j q_i'|^2 && \text{(definition of } C_{\text{small}}) \\
&\leq 2 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 2 \sum_{c_j \in C_{\text{small}}} \min_{q_i \in Q} |c_j q_i|^2 && \text{(Inequality (4))} \\
&\leq 2 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 2 \sum_{p_j \in P_{\text{small}}} \min_{q_i \in Q} (2|c_j p_j|^2 + 2|p_j q_i|^2) && \text{(Cauchy-Schwarz)} \\
&\leq 6 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 4 \sum_{p_j \in P_{\text{small}}} \min_{q_i \in Q} |p_j q_i|^2 && \text{(Inequality (2))} \\
&\leq 6 \sum_{j=2k+1}^n \text{radius}(b_j)^2 + 4 \cdot \text{OPT}_2(P, k) && \text{(Inequality (3))}
\end{aligned}$$

◀

The lemma above shows that our preclustering gives an additive error that depends on the radii of the small balls. The following two lemmas will be used to turn this into a multiplicative error. Let r_d^* be the smallest possible radius of any ball that intersects three disjoint unit balls in \mathbb{R}^d .

► **Lemma 5.** *We have*

- (i) $\text{OPT}_\infty(P, k) \geq r_d^* \cdot \text{radius}(b_{2k+1})$
- (ii) $\text{OPT}_1(P, k) \geq r_d^* \cdot \sum_{j=2k+1}^n \text{radius}(b_j)$
- (iii) $\text{OPT}_2(P, k) \geq (r_d^*)^2 \cdot \sum_{j=2k+1}^n \text{radius}(b_j)^2$

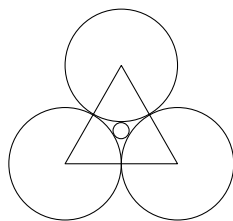
Proof. For part (i) notice that by the Pigeonhole Principle an optimal clustering must have a cluster containing at least three points from $\{p_1, \dots, p_{2k+1}\}$. The cost of this cluster is lower bounded by the radius of the smallest ball intersecting three balls of radius at least b_{2k+1} , which is in turn lower bounded by $r_d^* \cdot \text{radius}(b_{2k+1})$.

For part (ii) let P_1, P_2, \dots, P_k be the clusters in an optimal k -median clustering on P , and let q_i be the centroid of P_i in this clustering. Let B_i be the set of balls corresponding to the points in P_i . We claim that

$$\sum_{p_j \in P_i} |p_j q_i| \geq r_d^* \cdot \left(\left(\sum_{b_j \in B_i} \text{radius}(b_j) \right) - \text{sum of the radii of the two largest balls in } B_i \right). \quad (6)$$

To show this, let $b(q_i, r)$ be the ball of radius r centered at q_i , let $P_i(r) := \{p_j \in P_i : b_j \cap b(q_i, r) \neq \emptyset\}$ be the set of points in P_i whose associated ball intersects $b(q_i, r)$, and let $B_i(r)$ be the corresponding set of balls. Since for sufficiently large r we have $P_i = P_i(r)$, it suffices to show that for all $r > 0$ we have

$$\sum_{p_j \in P_i(r)} |p_j q_i| \geq r_d^* \cdot \left(\left(\sum_{p_j \in B_i(r)} \text{radius}(b_j) \right) - \text{sum of the radii of the two largest balls in } B_i(r) \right).$$



■ **Figure 3** The figure shows the smallest possible ball intersecting three disjoint unit balls in 2D. The larger balls are the unit balls and the radius of the small ball is $r_2^* = \frac{2}{\sqrt{3}} - 1$.

To prove this, consider this inequality as r increases from $r = 0$ to $r = \infty$. As long as $|P_i(0)| \leq 2$ the right-hand side is zero and so the inequality is obviously true. As we increase r further, $b(q_i, r)$ starts intersecting more and more balls from B_i . Consider what happens to the inequality when $b(q_i, r)$ starts intersecting another ball $b_\ell \in B_i$. Then p_ℓ is added to $P_i(r)$, so the left-hand side of the inequality increases by $|p_\ell q_i|$, which is at least r . The right-hand side increases by at most r_d^* times the radius of the third-largest ball in B_i . By definition of r_d^* , if three balls intersect a ball of radius r then the smallest has radius at most r/r_d^* . Hence, the right-hand side increases by at most r and the inequality remains true.

Recall that b_1, \dots, b_{2k} are the $2k$ largest balls in B . Hence, summing Inequality (6) over all clusters P_1, \dots, P_k gives

$$\text{OPT}_1(P, k) = \sum_{i=1}^k \sum_{p_j \in P_i} |p_j q_i| \geq r_d^* \cdot \left(\sum_{i=1}^k \sum_{b_j \in B_i} \text{radius}(b_j) - \sum_{j=1}^{2k} \text{radius}(b_j) \right) = r_d^* \cdot \sum_{j=2k+1}^n \text{radius}(b_j).$$

For part (iii) the same proof as (ii) works if we replace all distances with squared distances. ◀

► **Lemma 6.** For all $d \geq 2$ we have $r_d^* = 2/\sqrt{3} - 1$.

Proof. It is easy to see that $r_d^* \leq r_2^*$, since any configuration of three disjoint unit disks in the plane, with a fourth disk intersecting all three, can be extended to \mathbb{R}^d by embedding the centers of the balls on a 2-dimensional plane in \mathbb{R}^d . Next we show that $r_d^* \geq r_2^*$ for all $d \geq 2$, which implies that $r_d^* = r_2^*$.

Let $d \geq 2$ and let b, b', b'' be three disjoint unit balls in \mathbb{R}^d . Let c, c', c'' denote the centers of $b, b',$ and b'' , respectively, and let h be a 2-dimensional plane containing c, c', c'' . Let D be a smallest ball that intersects b, b', b'' and whose center is restricted to lie on h . Then $\text{radius}(D) \geq r_2^*$. We claim that D is in fact a smallest ball intersecting b, b', b'' even if we do not restrict the center of this ball to be on h . Indeed, if a ball D' with center $q \notin h$ intersects b, b', b'' , then the ball of the same radius as D' and whose center is the orthogonal projection of q onto h also intersects b, b', b'' .

It remains to show that $r_2^* = 2/\sqrt{3} - 1$. The configuration minimizing the radius of the smallest ball intersecting b, b', b'' is where b, b', b'' are pairwise touching, resulting in the claimed bound – see Fig. 3. ◀

We are now ready to prove the following theorem.

► **Theorem 7.** *Let B be a set of disjoint balls in \mathbb{R}^d with $d \geq 2$. Then*

- (i) *there exists a $(3k, 7 + 4\sqrt{3})$ -preclustering for the k -center and the k -median problem,*
- (ii) *there exists a $(3k, 130 + 72\sqrt{3})$ -preclustering for the k -means problem.*

Moreover, a $(3k, 7 + 4\sqrt{3} + \varepsilon)$ -preclustering for the k -center and the k -median problem, and a $(3k, 130 + 72\sqrt{3} + \varepsilon)$ -preclustering for the k -means problem can be computed in polynomial time.

Proof. Parts (i) and (ii) follow immediately by putting together Lemmas 4–6. It remains to argue that we can compute a preclustering whose approximation ratio is as claimed in polynomial time. Recall that each of the three clustering problems admits a PTAS [1, 2, 4, 6, 8], that is, for any given $\varepsilon' > 0$ we can compute a $(1 + \varepsilon')$ -approximation to an optimal clustering in polynomial time. To obtain the result, we set $\varepsilon' := \varepsilon / (1 + \frac{1}{r_d^*})$ for the k -center and k -median problem and $\varepsilon' := \varepsilon / (2 + \frac{2}{(r_d^*)^2})$ for the k -means problem. Then in Step 2 of PRECLUSTERING-DD(B, k) we compute a $(1 + \varepsilon')$ -approximation of the optimal clustering. The resulting algorithm runs in polynomial time. The only change in the analysis will appear in Inequality (4) of Lemma 4, where we get an extra multiplicative factor $1 + \varepsilon'$. With the above choice of ε' the approximation ratio for the whole algorithm will increase by ε . ◀

Generalizing the solution. We generalize the above theorem in order to control the number of preclusters for various approximations. Let r_d^p be the minimum possible value for the radius of a ball being tangent to p disjoint unit balls in \mathbb{R}^d for $d \geq 2$. Notice that $r_d^3 = r_d^*$. We can generalize the above result for appropriate p as follows.

The algorithm here is similar to PRECLUSTERING-DD, but in Step 2 we replace b_{2k+1} by $b_{(p-1)k+1}$ and in Step 3 we return the preclustering $\mathcal{C} := \{b_1\}, \dots, \{b_{(p-1)k}\}, B_1, \dots, B_k$. Note that Lemmas 4, 5 still hold if we replace $2k + 1$ with $(p - 1)k + 1$ and r_d^* with r_d^p .

► **Theorem 8.** *Let B be a set of disjoint balls in \mathbb{R}^d with $d \geq 2$. Then*

- (i) *there exists a $(pk, 1 + \frac{2}{r_d^p})$ -preclustering for the k -center and the k -median problem.*
- (ii) *there exists a $(pk, 4 + \frac{6}{(r_d^p)^2})$ -preclustering for the k -means problem.*

Moreover, a $(pk, 1 + \frac{2}{r_d^p} + \varepsilon)$ -preclustering for the k -center and the k -median problem, and a $(pk, 4 + \frac{6}{(r_d^p)^2} + \varepsilon)$ -preclustering for the k -means problem can be computed in polynomial time.

For instance, for $d = 2$ and $p = 6$ we have $r_2^6 = 1$ – indeed, any ball intersecting six disjoint unit balls in \mathbb{R}^2 has at least unit radius itself – leading to the following corollary. (For other bounds on r_d^p , see at [15].)

► **Corollary 9.** *Any set of disjoint balls in \mathbb{R}^2 admits a $(6k, 3)$ -preclustering for the k -center and the k -median problem, and a $(6k, 10)$ -preclustering for k -means problem.*

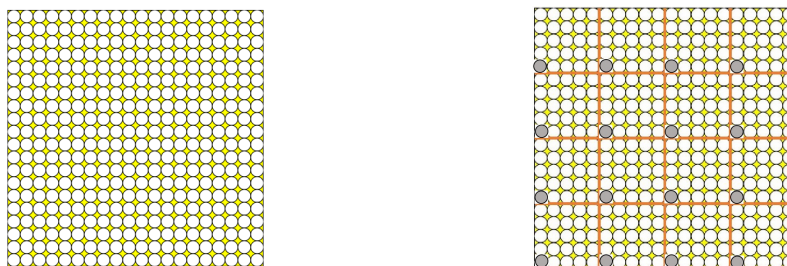
4 Asymptotically tight trade-offs for the k -center problem

Next, we explain how to obtain a $(\Theta(\lceil 1/\varepsilon^d \rceil \cdot k), \varepsilon)$ -preclustering for the k -center problem, by adding more steps to the algorithm PRECLUSTERING-DD(B, k).

► **Lemma 10.** *For any point set P in \mathbb{R}^d , any integer $k \geq 1$, and any $\varepsilon > 0$ we have*

$$OPT_\infty(P, c_d(\varepsilon) \cdot k) \leq \varepsilon \cdot OPT_\infty(P, k)$$

for $c_d(\varepsilon) = \lceil \sqrt{d}/\varepsilon \rceil^d$.



(a) n unit balls forming a square in 2D.

(b) clustering the circles into square-shaped clusters.

■ **Figure 4** Illustration for the proof Theorem 12.

Proof. First consider the case $k = 1$. Let Q be the optimal centroid for P and let S be the smallest hypercube centered at Q and containing P . Note that the edge length of S is at most $2\text{OPT}_\infty(P, k)$. Partition S into $\lceil \sqrt{d}/\varepsilon \rceil^d$ smaller hypercubes of edge length at most $2\varepsilon \cdot \text{OPT}_\infty(P, k)/\sqrt{d}$, and for each such hypercube make a cluster containing all points in it. Note that each such cluster can be covered by a ball of radius $\varepsilon \cdot \text{OPT}_\infty(P, k)$. Hence,

$$\text{OPT}_\infty(P, \lceil \sqrt{d}/\varepsilon \rceil^d \cdot k) \leq \varepsilon \cdot \text{OPT}_\infty(P, k).$$

For $k > 1$ we can simply apply the result for $k = 1$ to each of the k clusters in an optimal k -center clustering on P . ◀

With this lemma in hand we can now run algorithm $\text{PRECLUSTERING-DD}(B, k')$ with the appropriate value of k , namely $k' = c_d(\varepsilon/(7 + 4\sqrt{3})) \cdot k$, and then by Theorem 7 we get a $(3k', \varepsilon)$ -preclustering with $k' = \Theta(\lceil 1/\varepsilon \rceil^d \cdot k)$.

► **Theorem 11.** *Let B be a set of disjoint balls in \mathbb{R}^d with $d \geq 2$. Then there exists a $(\Theta(\lceil 1/\varepsilon^d \rceil \cdot k), \varepsilon)$ -preclustering for B for any positive constant ε .*

Finally, we show that this number of preclusters is asymptotically the best number we can achieve.

► **Theorem 12.** *There exists a set B of n disjoint balls in \mathbb{R}^d such that in any $(f(k), \varepsilon)$ -preclustering of B for the k -center problem, we have $f(k) = \Omega(\lceil 1/\varepsilon^d \rceil \cdot k)$.*

Proof. Observe that it suffices to prove the lower bound for $k = 1$; for larger k we can simply copy the construction k times and put the copies sufficiently far from each other. Now, for $k = 1$ consider a set B of $n^{1/d} \times \dots \times n^{1/d}$ unit balls arranged in a grid-like pattern, as in Fig. 4a. Note that $\text{OPT}_\infty(P, 1) \leq \sqrt{d}(n^{1/d} + 1)$ for any B -instance P . Now partition the “grid” into $(\sqrt{d}/\varepsilon)^d$ “subgrids” as in Fig. 4b. For each subgrid, select the ball with the lexicographically smallest center (shaded in Fig. 4b), and let $B^* \subset B$ be the set of selected balls. If a preclustering uses fewer than $(\sqrt{d}/\varepsilon)^d$ preclusters, two of the balls from B^* will end up in the same precluster. But then there is a B -instance P where $\mathcal{C}\text{-COST}_\infty(P) > \varepsilon \cdot \sqrt{d} \cdot n^{1/d} + 1$. Hence, any $(f(1), \varepsilon)$ -precluster must have $\Omega(\lceil 1/\varepsilon^d \rceil)$ preclusters. ◀

5 Concluding remarks

In this paper, we introduced the concept of preclustering for imprecise points and studied it for k -center, k -median and k -means problems. It would be interesting if one can fill the gap between lower and upper bounds for the number of preclusters needed in order to approximate the optimum solution. Also one can try to generalize the ideas used in section 4 for the k -median and k -means versions. It would also be interesting to study non-disjoint balls, and try to obtain preclusterings whose size and approximation ratio depend on the amount of overlap between the balls.

References

- 1 Pankaj K. Agarwal and Cecilia Magdalena Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.
- 2 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k -medians and related problems. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 106–113, 1998. doi:10.1145/276698.276718.
- 3 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011. doi:10.1007/s00453-010-9430-0.
- 4 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k -means clustering based on weak coresets. In *Proceedings of ACM Symposium on Computational Geometry*, pages 11–18, 2007. doi:10.1145/1247069.1247072.
- 5 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k -center problem. *Math. Oper. Res.*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.
- 6 Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D 2-sampling based PTAS for k -means and other clustering problems. In *Computing and Combinatorics COCOON 2012*, volume 7434 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2012. doi:10.1007/978-3-642-32241-9_2.
- 7 Wenqi Ju, Jun Luo, Binhai Zhu, and Ovidiu Daescu. Largest area convex hull of imprecise data based on axis-aligned squares. *J. Comb. Optim.*, 26(4):832–859, 2013. doi:10.1007/s10878-012-9488-5.
- 8 Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean kappa-median problem. In *Algorithms - ESA Proceedings*, volume 1643 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 1999. doi:10.1007/3-540-48481-7_33.
- 9 Chih-Hung Liu and Sandro Montanari. Minimizing the diameter of a spanning tree for imprecise points. *Algorithmica*, 80(2):801–826, 2018. doi:10.1007/s00453-017-0292-6.
- 10 Maarten Löffler. *Data Imprecision in Computational Geometry*. PhD thesis, Utrecht University, Netherlands, 2009.
- 11 Maarten Löffler and Marc J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010. doi:10.1007/s00453-008-9174-2.
- 12 Takayuki Nagai and Nobuki Tokura. Tight error bounds of geometric problems on convex objects with imprecise coordinates. In *JCDCG*, volume 2098 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000. doi:10.1007/3-540-47738-1_24.
- 13 Farnaz Sheikhi, Ali Mohades, Mark de Berg, and Ali D. Mehrabi. Separability of imprecise points. *Comput. Geom.*, 61:24–37, 2017. doi:10.1016/j.comgeo.2016.10.001.
- 14 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993. doi:10.1007/BF01585178.
- 15 István Talata. Exponential lower bound for the translative kissing numbers of d -dimensional convex bodies. *Discrete and Computational Geometry*, 19(3):447–455, 1998. doi:10.1007/PL00009362.

Parameter Analysis for Guarding Terrains

Akanksha Agrawal

Ben-Gurion University of the Negev, Beersheba, Israel
agrawal@post.bgu.ac.il

Sudeshna Kolay

Ben-Gurion University of the Negev, Beersheba, Israel
sudeshna@post.bgu.ac.il

Meirav Zehavi

Ben-Gurion University of the Negev, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

The TERRAIN GUARDING problem is a well-known variant of the famous ART GALLERY problem. Only second to ART GALLERY, it is the most well-studied visibility problem in Discrete and Computational Geometry, which has also attracted attention from the viewpoint of Parameterized complexity. In this paper, we focus on the parameterized complexity of TERRAIN GUARDING (both discrete and continuous) with respect to two natural parameters. First we show that, when parameterized by the number r of reflex vertices in the input terrain, the problem has a polynomial kernel. We also show that, when parameterized by the number c of minima in the terrain, DISCRETE ORTHOGONAL TERRAIN GUARDING has an XP algorithm.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Terrain Guarding, Reflex Vertices, Terrain Minima, FPT Algorithm, XP Algorithm, Kernelization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.4

Acknowledgements The second author would like to thank Prof. Mark de Berg for very insightful preliminary discussions for the second problem. The first and third authors are thankful to Prof. Saket Saurabh for helpful discussions.

1 Introduction

The (CONTINUOUS and DISCRETE) TERRAIN GUARDING problem is a widely studied problem in Discrete and Computational Geometry. In particular, it is the most well-studied visibility problem except for the classic ART GALLERY problem. Formally, a *1.5-dimensional terrain* $T = (V, E)$, or *terrain* for short, is a graph on vertex-set $V = \{v_1, v_2, \dots, v_n\}$ where each vertex v_i is associated with a point (x_i, y_i) on the two-dimensional Euclidean plane such that $x_1 \leq x_2 \leq \dots \leq x_n$ where, for any $i \in \{1, 2, \dots, n-2\}$, having $x_i = x_{i+1} = x_{i+2}$ implies that either $y_i < y_{i+1} < y_{i+2}$ or $y_i > y_{i+1} > y_{i+2}$ (see Figure 1); the edge-set of this graph is $E = \{\{v_i, v_{i+1}\} : i \in \{1, 2, \dots, n-1\}\}$. In the two-dimensional Euclidean plane, let R_1 be a ray starting from vertex $v_1 \in V$ towards negative infinity, and R_2 be a ray starting from $v_n \in V$ towards positive infinity. The region bounded by $T \cup R_1 \cup R_2$ and lying on and above T is called the *region bounded by the terrain T* . Note that the points lying on the terrain include the vertices $v_i \in V$, $1 \leq i \leq n$, as well as the points that lie on the edges in E . The CONTINUOUS TERRAIN GUARDING problem takes as input a terrain and a positive integer k , and the objective is to decide whether one can place guards on at most k points on the terrain such that each point on the terrain is seen by at least one guard. When we say that a point p sees a point q , we mean that the line segment \overline{pq} lies in the region bounded by the terrain. Notice that the guards may be placed on points on the terrain that do not



© Akanksha Agrawal, Sudeshna Kolay, and Meirav Zehavi;
licensed under Creative Commons License CC-BY

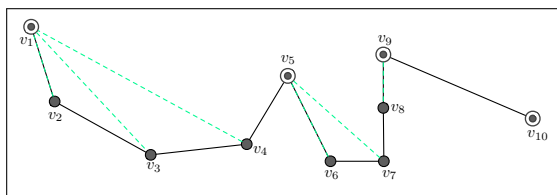
17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 4; pp. 4:1–4:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A terrain, where convex vertices are denoted by circles, reflex vertices are denoted by double circles, and edges are denoted by straight line segments. The set of reflex vertices sees all the vertices of the terrain.

belong to V . The DISCRETE TERRAIN GUARDING problem is defined similarly, with the requirement that the guards must be placed on vertices in V only, as well as that only each vertex in V must be guaranteed to be seen by at least one guard.

One of the reasons why the TERRAIN GUARDING problem and its numerous variants are important is because there is a wide variety of applications in the design of communication technologies such as cellular networks and line-of-sight transmission networks for radio broadcasting, as well as in coverage of highways, streets and walls with street lights as well as security cameras and natural terrain border security [2, 12]. In Discrete and Computational Geometry, the problem has its origin in 1995, when an NP-hardness proof was claimed by Chen *et al.* [5]. This proof was never completed and it took almost 15 years until King and Krohn [18] finally showed that this problem is indeed NP-hard. In between, the problem has received a lot of attention from the viewpoint of approximation algorithms. In 2005, Ben-Moshe *et al.* [2] obtained the first constant-factor approximation algorithm for DISCRETE TERRAIN GUARDING. Subsequently, the approximation factor was gradually improved in [6, 17, 11], until a PTAS was proposed by Gibson *et al.* [14] for DISCRETE TERRAIN GUARDING. Recently, Friedrichs *et al.* [12] showed that even the CONTINUOUS TERRAIN GUARDING problem admits a PTAS.

A special case of TERRAIN GUARDING that has received notable attention is ORTHOGONAL TERRAIN GUARDING, which was recently shown to be NP-hard [4]. Here, the terrain is *orthogonal*: for each vertex v_i , $2 \leq i \leq n - 1$, either both $x_{i-1} = x_i$ and $y_i = y_{i+1}$ or both $y_{i-1} = y_i$ and $x_i = x_{i+1}$. In other words, each edge is either a horizontal line segment or a vertical line segment, and each vertex is incident to at most one horizontal edge and at most one vertical edge (see Figure 2 for two examples of orthogonal terrains). This problem is of particular interest to the algorithm design community as it provides more structure and therefore more positive results than TERRAIN GUARDING [15, 19, 20, 10]. Although the PTASes designed in [14] and [12] clearly work for the ORTHOGONAL TERRAIN GUARDING problem as well, studies on this particular variant of TERRAIN GUARDING bring out interesting structural properties specific to this variant. For instance, in the work of Katz and Roisman [15] a relatively simple 2-approximation algorithm is described for DISCRETE ORTHOGONAL TERRAIN GUARDING. Recently, Lyu and Üngör [19] improved upon this result by developing a *linear-time* 2-approximation algorithm for ORTHOGONAL TERRAIN GUARDING. In [20] and [10], restricted versions were studied under which ORTHOGONAL TERRAIN GUARDING can be solved in polynomial time.

With a satisfactory landscape of approximation results for TERRAIN GUARDING, the focus shifted to parameterized variants of the problem. In fact, in their landmark paper [18] King and Krohn state that “the biggest remaining question regarding the complexity of TERRAIN GUARDING is whether or not it is FPT”. Khodakarami *et al.* [16] introduced the parameter “the depth of the onion peeling of a terrain” and showed that TERRAIN

GUARDING is FPT with respect to this parameter. In [1], for the solution size k as parameter a subexponential-time algorithm for TERRAIN GUARDING with running time $n^{\mathcal{O}(\sqrt{k})}$ was given in both discrete and continuous domains. In the same paper, an FPT algorithm with running time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ was presented for DISCRETE ORTHOGONAL TERRAIN GUARDING. We remark that a lower bound of $2^{\Omega(\sqrt{n})}$ for the time complexity of any algorithm for TERRAIN GUARDING under the Exponential Time Hypothesis (ETH) was claimed in the conference version [3] of [4], but the proof was said to be false and replaced by a lower bound of $2^{\Omega(n^{1/3})}$ under the ETH in [4].

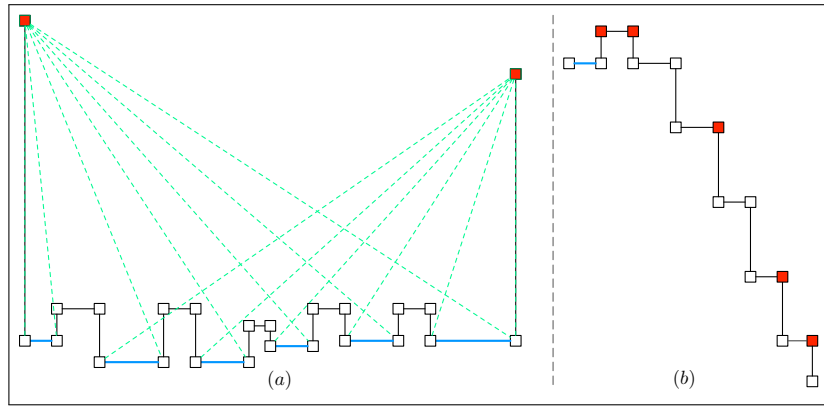
The Parameters. We consider two structural parameters. So far, the understanding of the parameterized complexity of TERRAIN GUARDING has been very limited, and, more generally, *exact* (exponential-time) algorithms for any visibility problem are extremely scarce. All our results utilize new and known structural properties of terrains. The individual results make use of different methods in parameterized complexity, and thus show several ways of how the aforementioned structural properties can be exploited algorithmically. In particular, we show how the paradigm of parameterized complexity can successfully yield *positive*, non-trivial results in the context of visibility. We believe that our work will open the door for additional research of which structural properties of terrains, polygons and related input domains make them easy to solve, and which do not. For example, here we see that terrains somewhat close to being convex, or which has constantly many minima, can be efficiently guarded.

We first consider the number r of *reflex vertices* of the terrain as a parameter; reflex vertices are those whose incident edges create an angle strictly larger than 180 degree in the region bounded by the terrain (see Figure 1);¹ all other vertices are *convex vertices*. It is known (and follows from, say, Theorem 1.5 of [21]) that if one places a guard on each of the reflex vertices of the terrain, then all points of the terrain are guarded. Hence, the parameterized instances of interest are those where $r > k$, k being the intended solution size. Thus, r can be considered as a natural relaxation of parameterization by solution size (whose status is a longstanding open problem). Further, we believe that it is interesting in its own right, since having a small (but not necessarily a fixed constant) parameter r means that the terrain is close to being convex. For such terrains, our result (formally stated ahead) not only shows that the problem is solvable efficiently (by a parameterized algorithm) but also that, in fact, the entire terrain can be shrunk to be of small size (by a kernelization algorithm).

The second structural parameter we consider is the number c of *minima* of the terrain, for orthogonal terrains. Recall that in the orthogonal terrains that we consider, every vertex is incident to at most one horizontal and at most one vertical edge. Then in an orthogonal terrain, a *minimum* is a horizontal edge whose y -coordinates are all the same as well as smaller than that of the (at most) two neighbours on either end (see the blue edges in Figure 2). Notice that in an orthogonal terrain, except for possibly the first and the last vertices of the terrain, the minima occur in pairs of convex endpoints of horizontal edges (see Figure 2).

It is to be noted that c , unlike r , cannot be related to k – it can be arbitrarily smaller or arbitrarily larger than k (as well as than r); see Figure 2. We believe that in many naturally occurring terrains the number of minima is much smaller than the number of observable vertices (where the gradient of the terrain changes). Indeed, it is conceivable to have (e.g., on natural hills or artificial structures) a huge number of vertices with slight changes of slope, and only few that actually alter the current “trend” (of having increasing y -coordinates or decreasing y -coordinates) of the terrain, in which case c is small.

¹ For the sake of convenience, we use the convention that the end vertices of the terrain are also its reflex vertices, unless otherwise stated.



■ **Figure 2** As illustration of c being arbitrary (a) larger or (b) smaller than k . In the terrains vertices are denoted by squares and edges by straight line segments. The red vertices are solution vertices, and the blue edges are the minima.

Our Contribution. First, we consider the DISCRETE TERRAIN GUARDING problem parameterized by the number r of reflex vertices. Then, an instance of the problem is denoted by (T, k, r) , where the input terrain T has r reflex vertices, and the objective is to determine if there is a k -sized vertex guard set for guarding all vertices of the terrain. Parameterized by r , we obtain a polynomial kernel for TERRAIN GUARDING:

► **Theorem 1.** *For an instance (T, k, r) of DISCRETE TERRAIN GUARDING, in polynomial time we can find an equivalent instance $(T' = (V', E'), k, r)$ of the problem, where $|V'| \in \mathcal{O}(r^2)$. Moreover, the problem admits a polynomial kernel, when parameterized by r .*

Our algorithm exploits structural properties of consecutively appearing convex vertices to identify vertices sufficient to capture a solution. We also find vertices guarding which would imply that all vertices of the terrain are guarded. Then, roughly speaking, we remove useless vertices (and make their neighbors adjacent) to obtain an instance with $\mathcal{O}(r^2)$ vertices. We remark that Theorem 1 also works for CONTINUOUS TERRAIN GUARDING, by using an appropriate “discretization” as described in [12] (for details see Section 3 and Appendix A.1).

Next, we consider DISCRETE ORTHOGONAL TERRAIN GUARDING parameterized by the number of minima, c , of the input orthogonal terrain. We design a somewhat tricky dynamic programming algorithm for it that belongs to XP. The membership in FPT remains open.

► **Theorem 2.** *DISCRETE ORTHOGONAL TERRAIN GUARDING parameterized by c can be solved in $4^c \cdot n^{2c+\mathcal{O}(1)}$ time.*

2 Preliminaries

For a positive integer k , we use $[k]$ as a shorthand for $\{1, 2, \dots, k\}$. We use standard notation and terminology from the book of Diestel [8] for graph-related terms which are not explicitly defined here. We only consider simple undirected graphs. Given a graph H , $V(H)$ and $E(H)$ denote its vertex-set and edge-set, respectively.

Terrains. Consider a terrain $T = (V, E)$, where $V = \{v_i = (x_i, y_i) \mid i \in [n]\}$ and $x_1 \leq x_2 \leq \dots \leq x_n$. We denote the ordering of vertices in T by $v_1 \prec v_2 \prec \dots \prec v_n$. Moreover, for vertices $v_i, v_j \in V$, we write $v_i \prec v_j$ if $i < j$, and $v_i \preceq v_j$ if $i \leq j$. We say that a subset

of points P on the terrain sees a subset of points Q on the terrain if each point in Q is seen by at least one point in P . A *subterrain* of T is an induced subgraph of T over a set $\{v_i, v_{i+1}, \dots, v_j\}$ of consecutive vertices in V with $i \leq j \in [n]$.

► **Proposition 3** (Order Claim [2]). *For a terrain $T = (V, E)$, consider four vertices $v_i \prec v_j \prec v_t \prec v_r$, such that v_i sees v_t , and v_j sees v_r . Then, v_i sees v_r .*

Consider an orthogonal terrain $T = (V, E)$. A *minimum* (resp. *maximum*) of T is a pair of consecutive vertices (v_i, v_{i+1}) of T , where $1 \leq i \leq n$, such that the following conditions are satisfied: i) $y_i = y_{i+1}$, ii) if v_{i-1} exists, then $y_{i-1} > y_i$ (resp. $y_{i-1} < y_i$), and iii) if v_{i+2} exists, then $y_{i+1} < y_{i+2}$ (resp. $y_{i+1} > y_{i+2}$).² We denote the set of minima and maxima of T by $\text{Min}(T)$ and $\text{Max}(T)$, respectively.

Parameterized Complexity. In Parameterized Complexity each problem instance is accompanied by a parameter k . A central notion in this field is *fixed-parameter tractability (FPT)*. This means, for a given instance (I, k) , solvability in time $f(k)|I|^{\mathcal{O}(1)}$ where f is some computable function of k . A *kernelization* algorithm for a parameterized problem Π is a polynomial time procedure which takes as input an instance (x, k) , where k is the parameter, and returns an instance (x', k') such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and $|x'| + k' \leq g(k)$, where g is a computable function. In the above, we say that Π admits a $g(k)$ -kernel. If $g(k)$ is a polynomial function, then the kernel is a *polynomial kernel* for Π . For more information on Parameterized Complexity we refer the reader to [9, 7].

3 Polynomial Kernel for Discrete Terrain Guarding

We design a polynomial kernel for DISCRETE TERRAIN GUARDING when parameterized by the number of reflex vertices. Towards this result, we prove (in Lemma 4) that given an instance (T, k, r) of DISCRETE TERRAIN GUARDING, in polynomial time we can compute an equivalent instance (T', k', r) of DISCRETE TERRAIN GUARDING with $\mathcal{O}(r^2)$ vertices. We now interpret the instance of DISCRETE TERRAIN GUARDING as an instance of DOMINATING SET with $\mathcal{O}(r^2)$ vertices. After this we apply the well-known polynomial time reduction (chain) from DOMINATING SET to DISCRETE TERRAIN GUARDING to obtain our kernel.

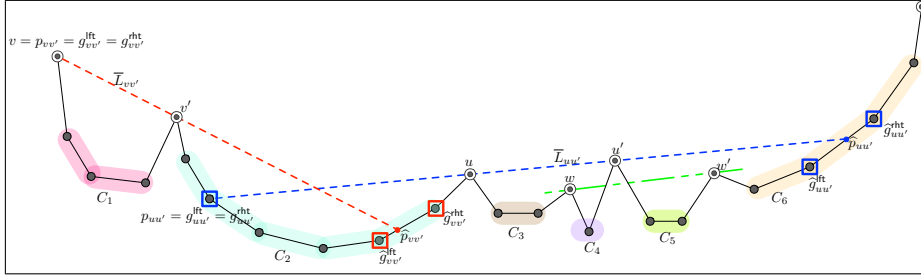
The main goal will be to prove the next lemma, which is the first statement of Theorem 1.

► **Lemma 4.** *For an instance (T, k, r) of DISCRETE TERRAIN GUARDING, in polynomial time we can compute an equivalent instance $(T' = (V', E'), k, r)$ of the problem, where $|V'| \in \mathcal{O}(r^2)$.*

Using the above lemma and polynomial time reducibility among NP-complete problems, we can obtain a proof of Theorem 1. Now we focus on the proof of Lemma 4. Let $(T = (V, k), k, r)$ be an instance of DISCRETE TERRAIN GUARDING.

We will design three marking schemes that will mark at most $\mathcal{O}(r^2)$ vertices. Roughly speaking, we will argue that there is a solution contained in the marked set of vertices, and guarding marked vertices is enough to guard all the vertices. Our first marking scheme will be used to ensure that there is a solution that contains only marked vertices. Our second and third marking schemes will be used to ensure that it is enough to guard the marked vertices. Finally, to obtain the proof of Lemma 4, we construct a modified terrain by adding edges between “consecutive” marked vertices in the original terrain.

² Recall that in an orthogonal terrain each vertex is adjacent to at most one horizontal edge and at most one vertical edge.



■ **Figure 3** An illustration of convex regions in a terrain and Marking Scheme II. The terrain has six convex regions C_1, C_2, \dots, C_6 , and the reflex vertices are double circled. The blue/red/green (dotted) lines/points/squares are the objects defined in Marking Scheme II. Also, the labelling of vertices and points are as defined in Marking Scheme II. We remark that for the pair of reflex vertices w, w' , the line segment $\bar{L}_{ww'}$ is blocked by the terrain.

We begin with some definitions and establish some useful properties regarding them, which will be helpful in proving the lemma.

► **Definition 5.** Given a terrain T , a convex region of T is a maximal subterrain of T where every vertex is a convex vertex (see Figure 3). For a convex region C , the vertex set of C is denoted by $V(C)$. A vertex in $V(C)$ that is not one of the two (not necessarily distinct) endpoints is called an internal vertex of C . A partial convex region is a subterrain of a convex region C that contains at least one endpoint of C . We can also define internal vertices of partial convex regions as above.

Notice that the ordering of vertices given by \preceq (and \prec) naturally extends to convex regions, as two convex regions do not have common vertices. Thus, hereafter we will use \preceq (and \prec) to denote orderings among convex regions as well. In the following we state some useful observations regarding convex regions. The first two are immediate.

- **Observation 6.** The number of convex regions in T is at most $r - 1$.
- **Observation 7.** Consider a convex region C in T with endpoints $v_i \preceq v_j$. For each $u, u' \in V(C) \cup \{v_{i-1}, v_{j+1}\}$,³ u sees u' .
- **Observation 8** (♠⁴). Let C be a convex region in T with endpoints $v_i \preceq v_j$. Consider vertices $v \notin V(C)$ and $u \in V(C)$, such that v sees u and $v \preceq v_i \preceq u \preceq v_j$ (resp. $v_i \preceq u \preceq v_j \preceq v$). Then v sees each vertex u' such that $u \preceq u' \preceq v_j$ (resp. $v_i \preceq u' \preceq u$).

We are now ready to state our first marking scheme. Intuitively speaking, this marking scheme is used to identify a set of vertices where we can always find a solution.

- **Definition 9** (Marking Scheme I). We create a subset $S_1 \subseteq V$ of vertices as follows.
 1. Add all the reflex vertices of T to S_1 .
 2. For each convex region C , add its two (not necessarily distinct) endpoints to S_1 .
 3. Consider an ordered pair of distinct convex regions (C_i, C_j) such that there is $v \in V(C_i)$ that sees all vertices of C_j . If $C_i \prec C_j$ (resp. $C_j \prec C_i$), let $f(C_i, C_j)$ be the largest (resp. smallest) vertex in C_i other than the endpoints of C_i that sees C_j . Add $f(C_i, C_j)$ to S_1 .

³ If $i = 1$, then we do not consider the vertex v_{i-1} . Similarly, if $j = n$, then we do not consider v_{j+1} . Notice that if v_{i-1} or v_{j+1} exist, then they are reflex vertices.

⁴ The proofs of the results marked with ♠ and some of the figures can be found in the Appendix.

4. Consider a reflex vertex v and a convex region C with endpoints v_i, v_j , such that $v_i \preceq v_j \prec v$ (resp. $v \prec v_i \preceq v_j$). Let $f(C, v)$ be the largest (resp. smallest) vertex other than the endpoints of C that v sees. Add $f(C, v)$ to S_1 .

The following observation easily follows from the above definition and Observation 6.

► **Observation 10.** *The number of vertices in S_1 is bounded by $\mathcal{O}(r^2)$.*

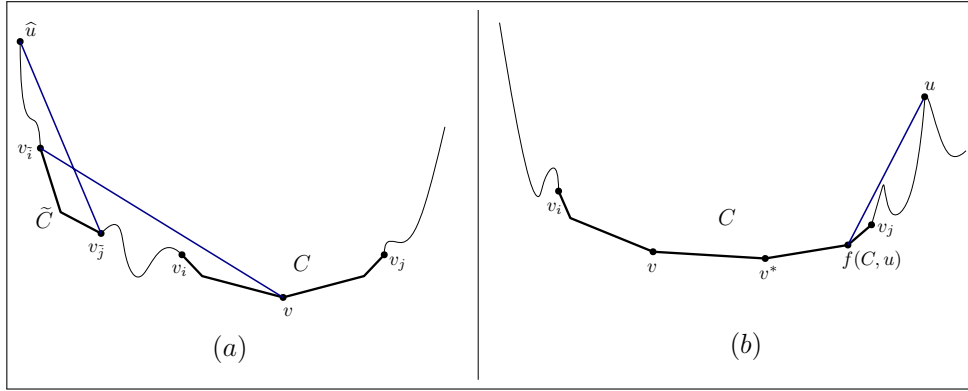
In the next lemma we show existence of a solution (for a yes-instance) contained in S_1 .

► **Lemma 11.** *(T, k, r) is a yes-instance of DISCRETE TERRAIN GUARDING if and only if there is a solution $S' \subseteq S_1$.*

Proof. If $S' \subseteq S_1$ is a solution for (T, k, r) then (T, k, r) is a yes-instance. Now suppose that (T, k, r) is a yes-instance. Consider a solution S' for (T, k, r) that maximizes the number of vertices from S_1 and is of minimum possible size. If $S' \subseteq S_1$, then we are done. Thus, we assume that there is $v \in S' \cap (V \setminus S_1)$. From Item 1 and 2 of Definition 9 we can obtain that v is neither a reflex vertex nor an endpoint of any convex region in T . Thus we assume that v belongs to a convex region, say C , with $v_i \prec v_j$ as its endpoints.

We first consider the case when there is a convex region \tilde{C} such that: i) \tilde{C} contains a vertex that is seen by v and no vertex in $S' \setminus \{v\}$, and ii) v does not see all vertices of \tilde{C} . (Note that $\tilde{C} \neq C$, from Observation 7.) Without loss of generality we assume that $\tilde{C} \prec C$. (The other case can be argued symmetrically.) For the arguments that follow, please refer to Figure 4(a). Let $v_{\tilde{i}} \prec v_{\tilde{j}}$ be the endpoints of \tilde{C} . We will argue that $S^* = (S' \setminus \{v\}) \cup \{v_j\}$ is a solution for (T, k, r) . Clearly, $|S^*| \leq k$. We will now argue that S^* sees each vertex in V . To prove the above, it is enough to show that for each $u \in V$, such that v is the only vertex in S' that sees it, there is a vertex in S^* that sees u . Consider such a vertex u . If $u \in V(C)$, then from Observation 7, v_j sees u . Now we consider the case when $u \prec v_i \prec v \prec v_j$. We will show that v_j sees u . Recall that u sees v by our assumption and v_i sees v_j (Observation 7). Thus using the Order Claim (Proposition 3) on $u \prec v_i \prec v \prec v_j$, we conclude that v_j sees u . Finally, we consider the case when $v_i \prec v \prec v_j \prec u$. As v does not see all vertices of \tilde{C} and $\tilde{C} \prec C$, using Observation 8 we conclude that v does not see $v_{\tilde{j}}$. As S' is a solution, there is some $\hat{u} \in S' \setminus \{v\}$ that sees $v_{\tilde{j}}$. By Observations 7 and 8, respectively, if $\hat{u} \in V(\tilde{C})$ or $v_{\tilde{j}} \prec \hat{u}$ then \hat{u} sees all vertices in \tilde{C} , which contradicts the choice of \tilde{C} to contain a vertex seen only by v and no other vertex in S' . Thus including the fact that $\tilde{C} \prec C$, it must be the case that $\hat{u} \prec v_{\tilde{i}} \prec v_{\tilde{j}} \prec v_i \prec v \prec v_j \prec u$. From Observation 8 we obtain that v sees $v_{\tilde{i}}$, and by assumption \hat{u} sees $v_{\tilde{j}}$. Thus, using the Order Claim for vertices $\hat{u} \prec v_{\tilde{i}} \prec v_{\tilde{j}} \prec v$, we obtain that \hat{u} sees v . Next, as $\hat{u} \prec v_i \prec v \prec v_j$, \hat{u} sees v , and v_i sees v_j (Observation 7 applied to C), using the Order Claim on $\hat{u} \prec v_i \prec v \prec v_j$ we obtain that \hat{u} sees v_j . Finally as $\hat{u} \prec v \prec v_j \prec u$, v sees u , and \hat{u} sees v_j , using the Order Claim on $\hat{u} \prec v \prec v_j \prec u$ we obtain that \hat{u} sees u . Thus, $S^* = (S' \setminus \{v\}) \cup \{v_j\}$ is a solution for the instance (T, k, r) , such that either $|S^*| < |S'|$, or $|S^*| = |S'|$ and $|S^* \cap S_1| > |S' \cap S_1|$. This contradicts the choice of S' .

Hereafter we assume that for any convex region \tilde{C} , either v sees all the vertices in \tilde{C} or sees none of its vertices. Again our goal will be to find another solution S^* by modifying S' so as to obtain a contradiction to the choice of S' . Towards the construction of S^* , we start by constructing a set X as follows. For each reflex vertex $u \in V$ such that v is the only vertex in S' that sees u , we add the vertex $f(C, u)$ to X (see Definition 9). Similarly, for each convex region C' such that v is the only vertex in S' that sees all the vertices of it, add the vertex $f(C, C')$ to X . As S' is a minimum sized solution, and hence a minimal solution, and $v \notin S_1$, we obtain that $X \neq \emptyset$. Let $v^* \in X$ be a vertex that is closest to v in



■ **Figure 4** An illustrative example of the case study in Lemma 11. Here, v is an unmarked vertex in S' that belongs to convex region C and v_i, v_j are the endpoints of the convex region C . (a) \tilde{C} is a convex region with endpoints v_i and v_j . A partial convex region C' of \tilde{C} that includes v_i is seen by v . The other endpoint v_j is seen by a vertex \hat{u} . (b) Any convex region that has some vertex seen by v has all its vertices seen by v . The vertex v^* is as defined in Lemma 11. Given a reflex vertex u , the vertex $f(C, u)$ is shown in the diagram.

(the path in) C . We assume that $v_i \preceq v \prec v^* \prec v_j$. (The case when $v_i \prec v^* \prec v \preceq v_j$ can be argued symmetrically.) For the arguments that follow, please refer to Figure 4(b). Let $S^* = (S' \setminus \{v\}) \cup \{v^*\}$. Notice that either $|S^*| < |S'|$, or $|S^*| = |S'|$ and $|S^* \cap S_1| > |S' \cap S_1|$. Thus, like previously, if we argue that S^* is a solution to (T, k, r) , then we will arrive at a contradiction to the choice of S' . Now we will show that S^* is a solution for (T, k, r) . First, we show that for each reflex vertex that v sees, the vertex v^* sees it as well. Consider a reflex vertex u that is seen by v . If $v^* = f(C, u)$, then clearly, v^* sees u . Now we assume that $v^* \neq f(C, u)$. If $v_i \prec v \prec v_j \prec u$, then by definition of $f(C, u)$ and the fact that $v^* \neq f(C, u)$ we obtain that $v^* \prec f(C, u) \prec v_j \prec u$. As v^* sees v_j (Observation 7) and $f(C, u)$ sees u , using the Order Claim on $v^* \prec f(C, u) \prec v_j \prec u$ we can obtain that v^* sees u . Next consider the case when $u \prec v_i \prec v \prec v_j$ (also we have $v^* \neq f(C, u)$ and $v \prec v^*$). In this case by definition of $f(C, u)$ and the fact that $v \notin S_1$, we obtain that $u \prec f(C, u) \prec v \prec v^*$. As v sees u and $f(C, u)$ sees v^* , we apply the Order Claim on $u \prec f(C, u) \prec v \prec v^*$ to obtain that v^* sees u .

Next, we show that for each convex region \tilde{C} that v sees (we are in the case when v sees all vertices of a convex region or none), the vertex v^* sees it as well. If $v^* = f(C, \tilde{C})$, then clearly, v^* sees \tilde{C} . Now we assume that $v^* \neq f(C, \tilde{C})$. If $C \prec \tilde{C}$ then $v_i \prec v \prec v_j \prec u$ for each vertex $u \in V(\tilde{C})$. Then by definition of $f(C, \tilde{C})$ and the fact that $v^* \neq f(C, \tilde{C})$ we obtain that $v^* \prec f(C, \tilde{C}) \prec v_j \prec u$ for each vertex $u \in \tilde{C}$. As v^* sees v_j (Observation 7) and $f(C, \tilde{C})$ sees \tilde{C} , using the Order Claim on $v^* \prec f(C, \tilde{C}) \prec v_j \prec u$ for each vertex $u \in V(\tilde{C})$, we can obtain that v^* sees each $u \in V(\tilde{C})$. Next consider the case when $\tilde{C} \prec C$. Then for each $u \in V(\tilde{C})$, $u \prec v_i \prec v \prec v_j$ (also we have $v^* \neq f(C, \tilde{C})$ and $v \prec v^*$). In this case by definition of $f(C, \tilde{C})$ and the fact that $v \notin S_1$, we obtain that for each $u \in V(\tilde{C})$, $u \prec f(C, u) \prec v \prec v^*$. As v sees u and $f(C, \tilde{C})$ sees v^* , we apply the Order Claim on $u \prec f(C, u) \prec v \prec v^*$ to obtain that v^* sees u , for each $u \in V(\tilde{C})$. Thus, v^* sees \tilde{C} . This concludes the proof. ◀

Our next two marking schemes will help us identify vertices such that guarding them will be sufficient for any vertex subset to qualify as a solution. We remark that the ordering \prec (and \preceq) of vertices of T naturally extends to the points that lie on the terrain. We will slightly abuse the notation and use \prec (and \preceq) to also denote the ordering of points on T .

► **Definition 12** (Marking Scheme II). Consider an (unordered) pair of distinct reflex vertices u, u' and let $L_{uu'}$ be the line containing them (see Figure 3). Let $\overline{L}_{uu'}$ (if it exists) be the maximal line segment with endpoints $p_{uu'}$ and $\hat{p}_{uu'}$ that contains both u and u' , and is completely contained on or above T . Let $g_{uu'}^{\text{left}}$ and $g_{uu'}^{\text{right}}$ be the (not necessarily distinct from $p_{uu'}$) vertices in V that are to the left and right of $p_{uu'}$, i.e. $g_{uu'}^{\text{left}}$ (resp. $g_{uu'}^{\text{right}}$) is the largest (resp. smallest) vertex in V , such that $g_{uu'}^{\text{left}} \preceq p_{uu'}$ (resp. $p_{uu'} \preceq g_{uu'}^{\text{right}}$). Add the vertices $g_{uu'}^{\text{left}}$ and $g_{uu'}^{\text{right}}$, and their (at most two) neighbors in T to S_2 . Similarly, let $\hat{g}_{uu'}^{\text{left}}$ (resp. $\hat{g}_{uu'}^{\text{right}}$) be the largest (resp. smallest) vertex in V , such that $\hat{g}_{uu'}^{\text{left}} \preceq p_{uu'}$ (resp. $p_{uu'} \preceq \hat{g}_{uu'}^{\text{right}}$). Add the vertices $\hat{g}_{uu'}^{\text{left}}$ and $\hat{g}_{uu'}^{\text{right}}$, and their neighbors in T to S_2 .

We design another simple marking scheme, which constructs a set of vertices S_3 which marks the neighbors of the vertices in $S_1 \cup S_2$ (excluding vertices in $S_1 \cup S_2$).

► **Definition 13** (Marking Scheme III). For each $u \in S_1 \cup S_2$ and $v \in V \setminus (S_1 \cup S_2)$, such that $\{u, v\} \in E$, add the vertex v to S_3 .

► **Observation 14.** $|S_1 \cup S_2 \cup S_3|$ is bounded by $\mathcal{O}(r^2)$. Moreover, $S_3 \cap (S_1 \cup S_2) = \emptyset$.

In the next lemma we show that guarding $S_1 \cup S_2 \cup S_3$ is enough to guard T , and the guards can be selected from the set $S_1 \cup S_2 \cup S_3$. (Although there is a solution contained in S_1 from Lemma 11, we state the lemma a bit differently to simplify its usage later.)

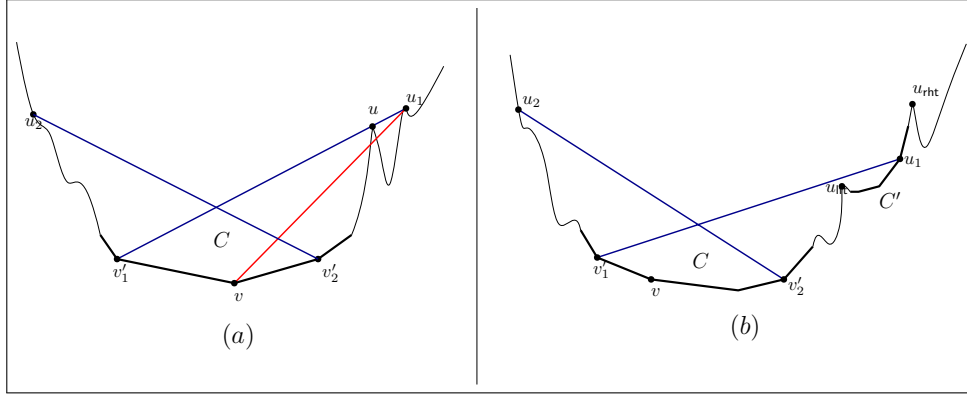
► **Lemma 15.** A set $S' \subseteq S_1 \cup S_2 \cup S_3$ of size at most k is a solution for the instance (T, k, r) of DISCRETE TERRAIN GUARDING if and only if for each $u \in S_1 \cup S_2 \cup S_3$, there is some $v \in S'$ that sees u .

Proof of Lemma 15. In one direction, suppose there is $S' \subseteq S_1 \cup S_2 \cup S_3$ that is a solution for the instance (T, k, r) . Then, clearly, for each $u \in S_1 \cup S_2 \cup S_3$, there is some $v \in S'$ that sees u .

In the other direction, consider a set $S' \subseteq S_1 \cup S_2 \cup S_3$ of minimum size that sees each vertex in $S_1 \cup S_2 \cup S_3$, and S' maximizes the number of reflex vertices it contains. We will show that S' is a solution for the instance (T, k, r) . For the sake of contradiction, suppose that there is a vertex $v \in V \setminus (S_1 \cup S_2 \cup S_3)$ that is seen by no vertex in S' . Since S_1 contains all reflex vertices (see Definition 9) and S_1 is guarded by S' , v must be a convex vertex. Let C be the convex region in T containing v . Let $v'_1 \in S_1 \cup S_2 \cup S_3$ be the largest vertex such that $v'_1 \prec v$. Similarly, let $v'_2 \in S_1 \cup S_2 \cup S_3$ be the smallest vertex such that $v \prec v'_2$. From Item 2 of Definition 9, both v'_1 and v'_2 exist, and they must belong to the convex region C . Moreover, from Definition 13, we obtain that $v'_1, v'_2 \in S_3$ (recall that $S_3 \cap (S_1 \cup S_2) = \emptyset$, see Observation 14).

If there is a $u_1 \in S'$ such that $u_1 \preceq v'_1$ and u_1 sees v'_1 , then using Observations 7 and 8 we conclude that u_1 sees v . Similarly, if there is $u_2 \in S'$, such that $v'_2 \preceq u_2$ and u_2 sees v'_2 , then we conclude that u_2 sees v . From the above, we assume that there are vertices $u_1, u_2 \in S'$, such that $u_2 \prec v'_1 \prec v \prec v'_2 \prec u_1$, u_1 sees v'_1 , and u_2 sees v'_2 . We now consider the following cases based on whether or not u_1 is a reflex vertex.

1. Suppose u_1 is a reflex vertex (see Figure 5(a)). Since u_1 sees v'_1 but not v , there must be a reflex vertex u , such that $v \prec u \prec u_1$ and the line segment $L_{u_1 u}$ intersects the subterrain C' of C between v'_1 and v (containing these vertices). Hence C' must contain a vertex from S_2 . As v'_1 is the largest vertex from $S_1 \cup S_2 \cup S_3$ with $v'_1 \prec v$, C' has no vertex from $S_1 \cup S_2 \cup S_3$ other than v'_1 . But $v'_1 \in S_3$ and $S_3 \cap (S_1 \cup S_2) = \emptyset$. This leads to a contradiction.



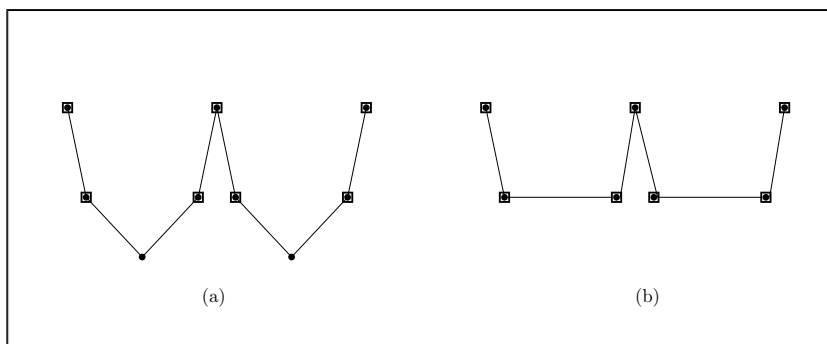
■ **Figure 5** An illustrative example of the case study in Lemma 15. Here, v'_1 and v'_2 are the nearest marked vertices to v . The vertices v'_1 and v'_2 are seen by u_1 and u_2 , respectively. (a) In Case 1, u_1 is a reflex vertex that sees v'_1 but cannot see v because of a reflex vertex u . (b) In Case 2, u_1 is a convex vertex and u_{rht} is the smallest reflex vertex to the right of u_1 . Similarly, u_{ft} is the largest reflex vertex to the left of u_1 .

2. Suppose u_1 belongs to a convex region, say C' . By our assumption u_1 does not see v , thus using Observation 7 we obtain that $C' \neq C$. Moreover, as $v \prec u_1$, we have $C \prec C'$. Let u_{rht} be the smallest reflex vertex such that $u_1 \prec u_{\text{rht}}$, and u_{ft} be the largest reflex vertex such that $u_{\text{ft}} \prec u_1$. Note that $u_{\text{ft}} \prec u_1 \prec u_{\text{rht}}$. We will show that $S^* = (S' \setminus \{u_1\}) \cup \{u_{\text{rht}}\}$ sees each vertex in $S_1 \cup S_2 \cup S_3$. Moreover, either $|S^*| < |S'|$, or $|S^*| = |S'|$ and S^* contains strictly more reflex vertices than S' . The above would lead us to a contradiction to the choice of S' . Now we focus on showing that S^* sees each vertex in $S_1 \cup S_2 \cup S_3$ (see Figure 5(b)). Consider any $u' \in S_1 \cup S_2 \cup S_3$. If u' is seen by a vertex in $S' \setminus \{u_1\}$, then clearly, S^* sees u' . If $u' \in V(C')$ or $u' \in \{u_{\text{ft}}, u_{\text{rht}}\}$, then using Observation 7 we can obtain that u_{rht} sees u' . Now we can assume that either $u' \prec u_{\text{ft}}$ or $u_{\text{rht}} \prec u'$. First consider the case when $u' \prec u_{\text{ft}}$. As $u' \prec u_{\text{ft}} \prec u_1 \prec u_{\text{rht}}$, u_1 sees u' (by assumption), and u_{ft} sees u_{rht} (Observation 7), using the Order Claim on $u' \prec u_{\text{ft}} \prec u_1 \prec u_{\text{rht}}$ we obtain that u_{rht} sees u' . Now we consider the other case, i.e., when $u_{\text{rht}} \prec u'$. Recall that $u_2 \prec v'_1 \prec v'_2 \prec u_1$, and u_1 sees v'_1 and u_2 sees v'_2 . Thus using the Order Claim on $u_2 \prec v'_1 \prec v'_2 \prec u_1$ we obtain that u_1 sees u_2 . As $u_2 \prec u_{\text{ft}} \prec u_1 \prec u_{\text{rht}}$, u_1 sees u_2 , and u_{ft} sees u_{rht} , using the Order Claim on $u_2 \prec u_{\text{ft}} \prec u_1 \prec u_{\text{rht}}$ we obtain that u_2 sees u_{rht} . Again, as $u_2 \prec u_1 \prec u_{\text{rht}} \prec u'$, u_2 sees u_{rht} , and u_1 sees u' , using the Order Claim on $u_2 \prec u_1 \prec u_{\text{rht}} \prec u'$ we obtain that u_2 sees u' . This concludes the proof. \blacktriangleleft

We define a new terrain $T' = (V' = S_1 \cup S_2 \cup S_3, E')$ (see Figure 6), where the coordinates of $S_1 \cup S_2 \cup S_3$ remain the same as in T and the edge set E' is defined as follows. Consider the restriction of the ordering, \prec , of vertices in T to the vertices in $S_1 \cup S_2 \cup S_3$. The set E' contains an edge between every consecutive pair of vertices in $S_1 \cup S_2 \cup S_3$, given by the above ordering. We have the following observations about the new terrain T' .

- **Observation 16** (\spadesuit). *A vertex is reflex in T if and only if it is a reflex vertex in T' .*
- **Observation 17** (\spadesuit). *Given two vertices $u, v \in S_1 \cup S_2 \cup S_3$, u sees v in T if and only if it sees v in T' .*

We are now ready to prove Lemma 4.



■ **Figure 6** An illustrative example of deriving from terrain $T = (V, E)$ shown in (a) the new terrain $T' = (S_1 \cup S_2 \cup S_3, E')$ shown in (b). The vertices in $S_1 \cup S_2 \cup S_3$ are denoted by boxes whereas unmarked vertices of V are denoted as circles.

Proof of Lemma 4. We show that $(T = (V, E), k, r)$ is a yes-instance of DISCRETE TERRAIN GUARDING if and only if $(T' = (V', E'), k, r)$ is a yes-instance of the problem. By Observation 16, the reflex vertices of T are reflex vertices of T' and vice versa. Therefore, the number of reflex vertices in both T and T' is r .

First, let (T, k, r) be a yes-instance of DISCRETE TERRAIN GUARDING. Following from Lemmas 11, there is a solution $S' \subseteq S_1 \cup S_2 \cup S_3$ of size at most k . In particular, S' guards all vertices in $S_1 \cup S_2 \cup S_3$. By Observation 17, S' is a k -sized solution for (T', k, r) and therefore (T', k, r) is a yes-instance.

On the other hand, let (T', k, r) be a yes-instance of DISCRETE TERRAIN GUARDING. Let S' be a k -sized solution for (T', k, r) . By Observation 17, $S' \subseteq S_1 \cup S_2 \cup S_3$ sees all vertices in $S_1 \cup S_2 \cup S_3$ in the terrain T . Thus, by Lemma 15 S' is a solution for (T, k, r) and therefore (T, k, r) is a yes-instance.

Moreover, we can construct (T', k, r) in polynomial time. Also from Observation 14 we have $|V'| \in \mathcal{O}(k^2)$. This concludes the proof. ◀

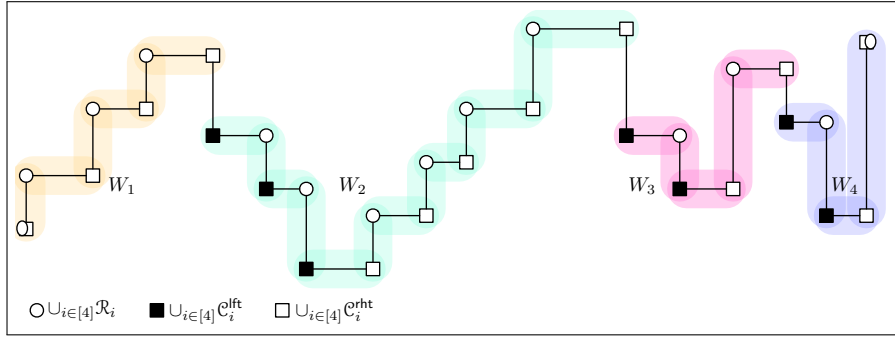
We conclude this section with the proof of Theorem 1.

Proof of Theorem 1. Let (T, k, r) be an instance of DISCRETE TERRAIN GUARDING. Using Lemma 4, in polynomial time we compute an equivalent instance $(T' = (V', E'), k, r)$ of DISCRETE TERRAIN GUARDING with $|V'| \in \mathcal{O}(k^2)$.

We now construct an instance of DOMINATING SET (G, k) as follows. We let $V(G) = V'$, and for $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and only if u and v see each other in T' . Clearly, (G, k) is a yes-instance of DOMINATING SET if and only if (T', k, r) is a yes-instance of DISCRETE TERRAIN GUARDING. Moreover, (G, k) can be constructed in polynomial time. Now we can convert the instance (G, k) of DOMINATING SET in polynomial time to an equivalent instance of DISCRETE TERRAIN GUARDING using the NP-hardness reduction from DOMINATING SET to DISCRETE TERRAIN GUARDING. (This can be explicitly achieved for example, by a chain of polynomial time reductions $\text{DOMINATING SET} \leq_{\text{poly}} \text{SAT} \leq_{\text{poly}} \text{3-SAT} \leq_{\text{poly}} \text{PLANAR 3-SAT} \leq_{\text{poly}} \text{DISCRETE TERRAIN GUARDING}$ [4, 18, 13].) This concludes the proof. ◀

4 Algorithm for Discrete Orthogonal Terrain Guarding

We design a dynamic programming based algorithm for DISCRETE ORTHOGONAL TERRAIN GUARDING running in time $4^{|\text{Min}(T)|} \cdot n^{2|\text{Min}(T)| + \mathcal{O}(1)}$, where n is the number of vertices in the input orthogonal terrain. Let (T, k) be an instance of DISCRETE ORTHOGONAL TERRAIN



■ **Figure 7** An intuitive illustration of the set of valleys $\mathcal{W} = \{W_1, W_2, W_3, W_4\}$ and different vertices in an orthogonal terrain. (The sets are presented modulo the elements $-\infty$ and $+\infty$.)

GUARDING. Intuitively speaking, in our algorithm the states for our dynamic programming table are chosen in relation to the minima of T as follows (see Figure 8). We will maintain a height, on or above which we can place guards. With respect to our minima, we will define the notion of valleys. For each such “valley”, we will have a vertex on its “left slope” in our state of the table, and we would like to guard all the vertices of the valley that appear in the “left slope” and lie on or above this vertex. Similarly, we will have such vertices for the “right slopes”. Towards formalizing the above, we begin by introducing some notations and preliminary results that will be useful later.

Notations. We let \mathcal{R} and \mathcal{C} denote the set of reflex and convex vertices of T , respectively. (For the sake of simplicity, we include the two endpoints of T in both \mathcal{R} and \mathcal{C}). In the following we state a well-known result from Claim 3.3 and 3.4 of [15], which states that guarding convex vertices of an orthogonal terrain using guards placed at reflex vertices is enough to guard the whole terrain. This property will be useful in our algorithm.

► **Proposition 18** ([15]). *(T, k) is a yes-instance of DISCRETE ORTHOGONAL TERRAIN GUARDING if and only if there is $S \subseteq \mathcal{R}$ of size at most k such that S sees each vertex of \mathcal{C} .*

► **Observation 19.** *For an orthogonal terrain T and vertices $u = (x_u, y_u) \in \mathcal{R}$ and $v = (x_v, y_v) \in \mathcal{C}$, if u sees v , then $y_v \leq y_u$.*

Next we will define the notion of valleys. Roughly speaking, a “valley” is a maximal region containing at most one minimum and at most two maxima. We will formally define the notion of valleys in an orthogonal terrain; our definitions will be formulated in a way to ensure uniqueness of the set of valleys in the given terrain (see Figure 7).

► **Definition 20.** *For an integer $i \geq 1$, the i^{th} valley, denoted by W_i (with its vertex set denoted by $V(W_i)$), of the terrain T is an (ordered) set of consecutive vertices of T that contains the smallest vertex u that is not contained in any valley W_j , where $j < i$, and the following vertices.⁵ Let $a < n$ be the smallest integer (if it exists) such that $(v_a, v_{a+1}) \in \text{Max}(T)$ and $v_a, v_{a+1} \notin \cup_{j < i} V(W_j)$. If a does not exist, then W_i contains all the vertices v where $u \preceq v \preceq v_n$. Otherwise, W_i contains all the vertices v where $u \preceq v \preceq v_{a+1}$.*

⁵ The 1st valley contains the vertex v_1 .

We let $\mathcal{W} = \{W_1, W_2, \dots, W_t\}$ be the set of valleys in T . Notice that $t \leq |\text{Min}(T)| + 2$. For a valley $W_i = (v_f, v_{f+1}, \dots, v_\ell) \in \mathcal{W}$, the vertices $\text{fst}(i) = v_f$ and $\text{lst}(i) = v_\ell$ denote the first and last vertices of W_i , respectively. For the sake of notational convenience, we will now define left/right slope convex vertices. We say that W_i contains a minimum/maximum (v_a, v_{a+1}) , if $v_a, v_{a+1} \in V(W_i)$. Note that by definition, W_i can contain at most one minimum and at most two maxima. If W_i has one minimum, say (v_a, v_{a+1}) , then the set of *left slope vertices* L_i , is the set $\{v_f, v_{f+1}, \dots, v_a\}$ and the set of *right slope vertices* R_i , is the set $\{v_{a+1}, v_{a+2}, \dots, v_\ell\}$. Otherwise, the vertices $(v_f, v_{f+1}, \dots, v_\ell)$ have either non-increasing y -coordinates or non-decreasing y -coordinates. If $(v_f, v_{f+1}, \dots, v_\ell)$ have non-increasing y -coordinates, then we have $L_i = V(W_i)$ and $R_i = \emptyset$. Otherwise, $(v_f, v_{f+1}, \dots, v_\ell)$ have non-decreasing y -coordinates, and we have $R_i = V(W_i)$ and $L_i = \emptyset$. We let $\mathcal{R}_i = V(W_i) \cap \mathcal{R}$, $\mathcal{C}_i^{\text{left}} = (L_i \cap \mathcal{C}) \cup \{-\infty\}$, $\mathcal{C}_i^{\text{right}} = (R_i \cap \mathcal{C}) \cup \{+\infty\}$ (see Figures 7 and 8).⁶ We let p_i^{left} be the *largest* vertex in $\mathcal{C}_i^{\text{left}}$. Similarly, we let \hat{p}_i^{right} be the *smallest* vertex in $\mathcal{C}_i^{\text{right}}$. We will now define the set of heights H of guards in the terrain, which will be used in defining the states of our dynamic programming routine: $H = \{y \mid v = (x, y) \in \mathcal{R}\} \cup \{+\infty\}$. For $y \in H \setminus \{+\infty\}$, by $\text{prv}(y)$ we denote the smallest element $y' \in H$ such that $y' > y$. (For the largest element, say $y^* \in H \setminus \{+\infty\}$, we have $\text{prv}(y^*) = +\infty$.) Finally for $i \in [t]$, we let $\mathcal{S}_i = \mathcal{C}_i^{\text{left}} \times \mathcal{C}_i^{\text{right}}$.

We state some useful observations that will be useful in our algorithm. We will move to the description of the states of our dynamic programming table after the stating few simple but useful observations below.

► **Observation 21.** Consider $i \in [t]$. For a vertex $v_j = (x_j, y_j) \in \mathcal{C}_i^{\text{left}}$, if $v_\ell = (x_\ell, y_\ell) \in \mathcal{R}$ sees v_j and $\ell < j$, then $\ell = j - 1$. Similarly, a vertex $v_j = (x_j, y_j) \in \mathcal{C}_i^{\text{right}}$, if $v_\ell = (x_\ell, y_\ell) \in \mathcal{R}$ sees v_j and $j < \ell$, then $\ell = j + 1$.

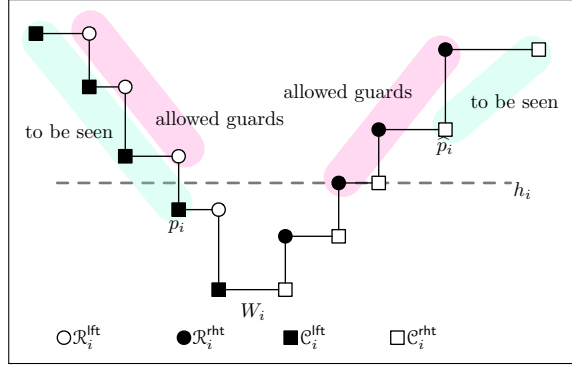
► **Observation 22.** Consider $i \in [t]$, and vertices $v_{j_1} = (x_{j_1}, y_{j_1}), v_{j_2} = (x_{j_2}, y_{j_2}) \in \mathcal{C}_i^{\text{left}}$, where $j_1 < j_2$. If $v_\ell = (x_\ell, y_\ell) \in \mathcal{R}$ sees v_{j_2} , where $j_2 < \ell$, and $y_{j_1} \leq y_\ell$, then v_ℓ sees v_{j_1} .

We define the set of heights of guards in a valley, which will be useful in stating the states of our dynamic programming routine. For $i \in [t]$, we let $\text{Hgt}(i) = \{y_a \mid v_a = (x_a, y_a) \in \mathcal{R}_i\} \cup \{+\infty\}$. Moreover, for $y_a \in \text{Hgt}(i) \setminus \{+\infty\}$, by $\text{prv}_i(y_a)$ we denote the smallest element $y_{a'} \in \text{Hgt}(i)$ such that $y_{a'} > y_a$. (For the largest element, say $y_a^* \in \text{Hgt}(i) \setminus \{+\infty\}$, we have $\text{prv}_i(y_a^*) = +\infty$.) Finally for $i \in [t]$, we let $\mathcal{S}_i = \mathcal{C}_i^{\text{left}} \times \mathcal{C}_i^{\text{right}} \times \text{Hgt}(i)$.

We let $\mathcal{R}_i^{\text{left}} = (\mathcal{R}^{\text{left}} \cap V(W_i)) \cup \{+\infty\}$, $\mathcal{R}_i^{\text{right}} = (\mathcal{R}^{\text{right}} \cap V(W_i)) \cup \{-\infty\}$, $\mathcal{C}_i^{\text{left}} = (\mathcal{C}^{\text{left}} \cap V(W_i)) \cup \{-\infty\}$ and $\mathcal{C}_i^{\text{right}} = (\mathcal{C}^{\text{right}} \cap V(W_i)) \cup \{+\infty\}$. Furthermore, we let $\mathcal{S}_i = \mathcal{C}_i^{\text{left}} \times \mathcal{C}_i^{\text{right}} \times \mathcal{R}_i^{\text{right}} \times \mathcal{R}_i^{\text{left}}$.

We are now ready to define the states of our dynamic programming algorithm. For each valley we will have the following in our dynamic programming states. Firstly, we have a pair of vertices from each valley, one from the left-side and other from the right-side of the valley. These two vertices tell us “what” vertices must be guarded (see Figure 8 for an illustration). Intuitively speaking, we want to guard all the left (resp. right) convex vertices in the valley that are on or above the left-side (resp. right-side) vertex for this valley in the state of our dynamic programming table. Additionally, we have a number denoting the height, on or above which we are allowed to place the guards. Apart from these, we will have a number $k' \leq k$ denoting the number of guards that we are allowed to use in our “partial” solution.

⁶ We use the convention that $-\infty$ and $+\infty$ are the smallest and largest elements, respectively, which are added for ease in comparison among vertices of a valley in the dynamic programming routine.



■ **Figure 8** An intuition of states of our dynamic programming algorithm.

States of the Dynamic Programming Table and their Interpretation. Consider $\tau = (\tau_1, \tau_2, \dots, \tau_t) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \dots \times \mathcal{S}_t$, where for $i \in [t]$, $\tau_i = (p_i, \hat{p}_i) \in \mathcal{S}_i$, $h \in H$, and an integer $k' \in \{0, 1, 2, \dots, k\}$. For each such triple we have an entry in our dynamic programming table denoted by $\Pi(\tau, h, k')$. For interpreting $\Pi(\tau, h, k')$, we will define $\Gamma(\tau, h, k')$; the goal of the algorithm will be to compute $\Pi(\tau, h, k')$, so as to mimic $\Gamma(\tau, h, k')$, for every triple.

► **Definition 23.** For $\tau = (\tau_1, \tau_2, \dots, \tau_t) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \dots \times \mathcal{S}_t$, where for $i \in [t]$, $\tau_i = (p_i, \hat{p}_i) \in \mathcal{S}_i$, $h \in H$, and an integer $k' \in \{0, 1, 2, \dots, k\}$, we have $\Gamma(\tau, h, k') = 1$ if and only if there is a set $S \subseteq \mathcal{R}$ of size at most k' such that the following conditions are satisfied (see Figure 8):

1. All the guards placed are at height at least h . That is, for each $v = (x, y) \in S$, we have $y \geq h$.
2. Each vertex in $\mathcal{C}_i^{\text{lft}}$ that is p_i or above it, is seen by a guard in S . Similarly, each vertex in $\mathcal{C}_i^{\text{rht}}$ that is \hat{p}_i or above it, is seen by a guard in S . So, for each $i \in [t]$ and $u \in \mathcal{C}_i^{\text{lft}} \cup \mathcal{C}_i^{\text{rht}}$, such that either $\text{fst}(i) \preceq u \preceq p_i$ or $\hat{p}_i \preceq u \preceq \text{lst}(i)$, there is $w \in S$ that sees u .

In the above, the set S is called a solution for (τ, h, k') .

Let $h^* = \min\{y \in H\}$, and $\tau_i^* = (p_i^{\text{lft}}, \hat{p}_i^{\text{rht}})$, for each $i \in [t]$. (In the above, for $i \in [t]$, as $-\infty \in \mathcal{C}_i^{\text{lft}}$ and $+\infty \in \mathcal{C}_i^{\text{rht}}$, p_i^{lft} and \hat{p}_i^{rht} can never be undefined.) From Proposition 18 we can obtain that (T, k) is a yes-instance of DISCRETE ORTHOGONAL TERRAIN GUARDING if and only if $\Gamma(\tau_1^*, \tau_2^*, \dots, \tau_t^*, h^*, k) = 1$.

Order of Computation of Entries. We describe the order in which we compute the entries of our dynamic programming table. We will use a modified form of “lexicographic” ordering for the table entries as follows. To this end we first describe how we order the vertices in the “left” and “right” sides of our valleys. For $i \in [t]$, the vertices in $\mathcal{C}_i^{\text{lft}}$ are ordered as per the ordering given by T , whereas, the vertices in $\mathcal{C}_i^{\text{rht}}$ are reverse ordered compared to the ordering given by T . (We need to do the above because when are going down the valley from right side, the vertices are decreasing.) We order the elements of H in decreasing order (with $+\infty$ being the first element in this ordering). Finally, the overall ordering is obtained by using (lexicographic) ordering of H , the ordering of vertices in $\mathcal{C}_i^{\text{lft}}$, and the ordering of vertices in $\mathcal{C}_i^{\text{rht}}$, with increasing values of i , and k' (increasing).

Next we will describe how we (recursively) compute the entries of the table. Consider $\tau = (\tau_1, \tau_2, \dots, \tau_t) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \dots \times \mathcal{S}_t$, where for $i \in [t]$, $\tau_i = (p_i, \hat{p}_i)$, $h \in H$, and an integer $k' \in \{0, 1, 2, \dots, k\}$. We compute $\Pi(\tau, h, k')$ as follows.

Base Cases. The base cases occur in the following scenarios, applied in the given order.

1. If for each $i \in [t]$, we have $p_i = -\infty$ and $\widehat{p}_i = +\infty$, then $\Pi(\tau, h, k') = 1$.
2. If $k = 0$ and for some $i \in [t]$, $p_i \neq -\infty$ or $\widehat{p}_i \neq +\infty$, then $\Pi(\tau, h, k') = 0$.
3. If $h = +\infty$ and for some $j \in [t]$, $p_j \neq -\infty$ or $\widehat{p}_j \neq +\infty$, then $\Pi(\tau, h, k') = 0$.

The correctness of the base cases directly follow from their description. Next we describe the recursive formula for computation of the other entries of our dynamic programming table.

Recursive Formula. Intuitively, we will compute the entry by taking “or” of the solutions for already computed entries, where the entries we query are based on where and at what vertices we place the lowest height guards in the partial solution.

Let $A_h = \{v = (x, y) \in \mathcal{R} \mid \text{and } y = h\}$. As Item 1 of Base Case is not applicable, we need to place at least one guard, thus we can obtain that $H \neq \emptyset$. Notice that $|A_h| \leq 2t$, as for each valley, we can have at most two vertices from \mathcal{R} that are at height h . For every $A \subseteq A_h$, we will compute ρ_A , which (intuitively speaking) corresponds to the solution S for (τ, h, k') , where $S \cap A_h = A$, i.e., A is the set of vertices of guards at height h in the solution. (We will have $\rho_A = 1$ if and only if there is a solution S for (τ, h, k') such that $S \cap A_h = A$.)

We remark that $h \neq +\infty$, as the base cases are not applicable. Consider $A \subseteq A_h$. If some $a \in A$ sees p_i , then we let $p_i[A]$ be the largest vertex in $\mathcal{C}_i^{\text{left}}$ that is not seen by any vertex in A . (If $p_i[A]$ does not exist, it is set to $-\infty$.) Otherwise, no $a \in A$ sees p_i , and we set $p_i[A] = p_i$. Similarly, if some $a \in A$ sees \widehat{p}_i , we let $\widehat{p}_i[A]$ be the smallest vertex in $\mathcal{C}_i^{\text{right}}$ that is not seen by any $a \in A$. (If $\widehat{p}_i[A]$ does not exist, it is set to $+\infty$.) Otherwise, no $a \in A$ sees \widehat{p}_i , and we set $\widehat{p}_i[A] = \widehat{p}_i$. Let $\tau_i[A] = (p_i[A], \widehat{p}_i[A])$. Finally, we let $\tau[A] = (\tau_1[A], \tau_2[A], \dots, \tau_t[A])$. Notice that $(\tau[A], \text{prv}(h), k' - |A|)$ is smaller in order compared to (τ, h, k) , and thus the entry corresponding to it in our dynamic programming table is already computed. We let $\rho_A = \Pi(\tau[A], \text{prv}(h), k' - |A|)$. Finally, we set $\Pi(\tau, h, k') = \bigvee_{A \subseteq A_h} \rho_A$.

► **Lemma 24.** *The recursive formula for computation of the entries is correct.*

Proof of Lemma 24. To establish the correctness it is enough to show that $\Gamma(\tau, h, k') = 1$ if and only if there is $A \subseteq A_h$, such that $\rho_A = \Pi(\tau[A], \text{prv}(h), k' - |A|) = 1$.

For the forward direction suppose that $\Gamma(\tau, h, k') = 1$, and $S \subseteq \mathcal{R}$ be a solution for (τ, h, k') . We let $A^* = S \cap A_h$ and $S^* = S \setminus A^*$. We will show that $\rho_{A^*} = \Pi(\tau[A^*], \text{prv}(h), k' - |A^*|) = 1$. We will show that $\Pi(\tau[A^*], \text{prv}(h), k' - |A^*|) = 1$, by proving that S^* is a solution for $(\tau[A^*], \text{prv}(h), k' - |A^*|)$. As S is a solution for (τ, h, k') , for each $v = (x, y) \in S$, we have $y \geq h$. The above together with the construction of S^* (and A_h) implies that for each $v = (x, y) \in S^*$, we have $y \geq \text{prv}(h)$, and $|S^*| \leq k' - |A^*|$. Now it remains to prove Item 2 of Definition 23, to show that S^* is a solution for $(\tau[A^*], \text{prv}(h), k' - |A^*|)$. Consider $i \in [t]$. We will show that for each $u \in \mathcal{C}_i$, such that either $\text{fst}(i) \preceq u \preceq p_i[A^*]$ or $\widehat{p}_i[A^*] \preceq u \preceq \text{lst}(i)$, there is some $s \in S^*$ that sees u . We will only prove the above statement for the case when $\text{fst}(i) \preceq u \preceq p_i[A^*]$. (We can obtain the proof for the case when $\widehat{p}_i[A^*] \preceq u \preceq \text{lst}(i)$, by following similar arguments.) Let $u = (x_u, y_u) \in \mathcal{C}_i^{\text{left}}$ be the largest vertex such that $\text{fst}(i) \preceq u \preceq p_i[A^*]$ and u is not seen by any vertex in S^* . (If such a vertex u does not exist, then the claim trivially follows.) Since S is a solution for (τ, h, k') and (by construction) $p_i[A^*] \leq p_i$, there exists $s = (x, h) \in S \setminus S^* = A^*$, such that s sees u . Furthermore, there is $s' = (x', y') \in S^*$, where $h < y'$, such that s' sees $p_i[A^*]$. From the above we can obtain that all of $u, p_i[A^*], s, s'$ are distinct and $u \prec p_i[A^*]$. From Observation 19 we have $y_u \leq h$. This together with the fact that $h < y'$ implies that $y_u < y'$. If $p_i[A^*] \prec s'$, then using Observation 22 we can conclude that s' sees u . This contradicts the choice of u that no vertex in S^* sees it. Now consider the case when $s' \prec p_i[A^*]$. In this case, using Observation 21 we

can obtain that $u \prec s' \prec p_i[A^*]$. Thus, we have $y_u \geq y'$. As $y' > h$, using Observation 19 we can obtain a contradiction to the our assumption that $s = (x, h)$ sees u . This concludes the proof of forward direction.

Now we consider the reverse direction. Consider $A^* \subseteq A_h$, such that $\rho_{A^*} = \Pi(\tau[A^*], \text{prv}(h), k' - |A^*|) = 1$, and let S^* be a solution for $(\tau[A^*], \text{prv}(h), k' - |A^*|)$. Let $S = S^* \cup A^*$. Clearly, $|S| \leq k'$, and for each $v = (x, y) \in S$, we have $y \geq h$. Also, for $i \in [t]$, by the construction of $p_i[A^*]$ and $\widehat{p}_i[A^*]$, and the fact that S^* is a solution for $(\tau[A^*], \text{prv}(h), k' - |A^*|)$, we can conclude that for each $u \in \mathcal{C}_i^{\text{ft}} \cup \mathcal{C}_i^{\text{rt}}$, such that either $\text{fst}(i) \preceq u \preceq p_i$ or $\widehat{p}_i \preceq u \preceq \text{lst}(i)$, there is $s \in S$ that sees u . From the above discussions we can obtain that S is a solution for (τ, h, k') , and thus we have $\Gamma(\tau, h, k') = 1$. This concludes the proof. \blacktriangleleft

Note that t , the number of valleys, is bounded $|\text{Min}(T)| + 2$. The number of entries in our dynamic programming table is bounded by $n^{2t + \mathcal{O}(1)}$. The entries in our base cases can be computed in $\mathcal{O}(1)$ time. The recursive formula per entry can be computed in time bounded by $2^{2t} \cdot n^{\mathcal{O}(1)}$, as $|A_h| \leq 2t$, for each $h \in H$. Thus we obtain the proof of Theorem 2.

References

- 1 Pradeesha Ashok, Fedor V. Fomin, Sudeshna Kolay, Saket Saurabh, and Meirav Zehavi. Exact algorithms for terrain guarding. *ACM Trans. Algorithms*, 14(2):25:1–25:20, 2018.
- 2 Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5d terrain guarding. *SIAM J. Comput.*, 36(6):1631–1647, 2007.
- 3 Édouard Bonnet and Panos Giannopoulos. Orthogonal terrain guarding is np-complete. In *Symposium on Computational Geometry, SoCG 2018*, pages 11:1–11:15, 2018.
- 4 Édouard Bonnet and Panos Giannopoulos. Orthogonal terrain guarding is np-complete. *JoCG*, 10(2):21–44, 2019.
- 5 Danny Z. Chen, Vladimir Estivill-Castro, and Jorge Urrutia. Optimal guarding of polygons and monotone chains. In *Proceedings of the 7th Canadian Conference on Computational Geometry, CCCG*, pages 133–138, 1995.
- 6 K. L. Clarkson and K. R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete and Computational Geometry*, 37(1):43–58, 2007.
- 7 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 9 R. Downey and M. R. Fellows. *Fundamentals of parameterized complexity*. Springer, 2013.
- 10 Stephane Durocher, Pak Ching Li, and Saeed Mehrabi. Guarding orthogonal terrains. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG*, 2015.
- 11 M. K. Elbassioni, E. Krohn, D. Matijevic, J. Mestre, and D. Severdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60(2):451–463, 2011.
- 12 S. Friedrichs, M. Hemmer, J. King, and C. Schmidt. The continuous 1.5D terrain guarding problem: Discretization, optimal solutions, and PTAS. *JoCG*, 7(1):256–284, 2016.
- 13 M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, New York, 1979.
- 14 M. Gibson, G. Kanade, E. Krohn, and K. Varadarajan. Guarding terrains via local search. *JoCG*, 5(1):168–178, 2014.
- 15 M. J. Katz and G. S. Roisman. On guarding the vertices of rectilinear domains. *Computational Geometry*, 39(3):219–228, 2008.
- 16 F. Khodakarami, F. Didehvar, and A. Mohades. A fixed-parameter algorithm for guarding 1.5D terrains. *Theoretical Computer Science*, 595:130–142, 2015.

- 17 James King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics, LATIN*, volume 3887, pages 629–640, 2006.
- 18 James King and Erik Krohn. Terrain guarding is np-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011.
- 19 Yangdi Lyu and Alper Üngör. A fast 2-approximation algorithm for guarding orthogonal terrains. In *Proceedings of the 28th Canadian Conference on Computational Geometry, CCCG*, pages 161–167, 2016.
- 20 S. Mehrabi. Guarding the vertices of an orthogonal terrain using vertex guards. *arXiv:1512.08292*, 2015.
- 21 Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., 1987.

A Remaining Details from Section 3

Proof of Observation 8. Consider the case when $v \preceq v_i \preceq u \preceq v_j$. (The other case can be proved by following similar arguments.) If $u = v_j$, then the claim trivially follows. Thus we assume that $u \prec v_j$. Consider $u' \in V(C)$, such that $u \prec u' \preceq v_j$. If $v = v_{i-1}$, then from Observation 7 it follows that v sees u' . Now we consider the case when $v \prec v_{i-1}$. From Observation 7, v_{i-1} sees u' , by our assumption v sees u , and we have $v \prec v_{i-1} \prec u \prec u'$. Thus by the Order Claim (Proposition 3) we can conclude that v sees u' . ◀

Proof of Observation 16. Consider a reflex vertex v in T . By Definition 9, $v \in V'$. We show that v is also a reflex vertex of T' . If v is the first or last vertex of T , then it is also the first or last vertex of T' and therefore is a reflex vertex by definition. Otherwise, v has two neighbours, say u_1 and u_2 . From Definition 9 we can obtain that $u_1, u_2 \in V'$. As the coordinates of u_1, v, u_2 in T are the same as that in T' , we obtain that v is a reflex vertex of T' .

Now we show that a vertex $v \in S_1 \cup S_2 \cup S_3$ that is a convex vertex in T is also a convex vertex in T' . By definition, v cannot be the first or last vertex. Let u_1 and u_2 be the two neighbours of v such that $u_1 \prec v \prec u_2$. If $u_1, u_2 \in V'$, then as the coordinates of u_1, v, u_2 in T are the same as that in T' , we obtain that v is also a convex vertex in T' . Otherwise, let $u'_1 \in V'$ be the closest such vertex to v where $u'_1 \prec v$. By definition of V' , such a vertex exists for all convex vertices v . Notice that it must hold that $u'_1 \preceq u_1 \prec v$. Also by definition of V' , if u_1 is a reflex vertex then $u'_1 = u_1$ and $u_1 \in V'$. Similarly, let $u'_2 \in V'$ be the closest such vertex to v where $v \prec u'_2$. By definition of V' , such a vertex exists for all convex vertices v . Notice that it must hold that $v \prec u_2 \preceq u'_2$. Again by definition of V' , if u_2 is a reflex vertex then $u'_2 = u_2$ and $u_2 \in V'$. Note that in T' , u'_1 and u'_2 are the neighbours of v such that $u'_1 \prec v \prec u'_2$. We are in the case that at least one of $u_1 \neq u'_1$ and $u_2 \neq u'_2$ holds. If u'_1 (u'_2) is not a reflex vertex, by construction of V' it must belong to the same convex region as v, u_1 (v, u_2). By Observation 7, u'_1 sees v (u'_2 sees v) and therefore u_1 lies below or on the line $\widehat{L}_{u'_1 v}$ (u_2 lies below or on the line $\widehat{L}_{v u'_2}$). Now consider $\angle u'_1 v u'_2$ and $\angle u_1 v u_2$ made inside the region bounded by T . It must be the case that $\angle u'_1 v u'_2 \leq \angle u_1 v u_2$. Thus, if v was a convex vertex in T then it means that in the region bounded by T $\angle u_1 v u_2 \leq 180^\circ$. This implies that in the region bounded by T' $\angle u'_1 v u'_2 \leq 180^\circ$, which means that v is a convex vertex of T' . ◀

Proof of Observation 17. For any $u, v \in S_1 \cup S_2 \cup S_3$ such the u sees v in T , each $w \in V$, such that $v \prec w \prec u$ (or $u \prec w \prec v$) must lie below or on the line \widehat{L}_{uw} , containing u and v . In particular, each $w \in S_1 \cup S_2 \cup S_3$ such that $v \prec w \prec u$ (or $u \prec w \prec v$) must lie below or on the line \widehat{L}_{uw} . Thus we can obtain that u sees v in T' .

Consider $u, v \in S_1 \cup S_2 \cup S_3$ such the u sees v in T' . Consider a $w \in V$ such that $u \prec w \prec v$ (we can give a symmetric argument for $v \prec w \prec u$). If $w \in V'$ then it must lie below or on the line \widehat{L}_{uv} . Otherwise, $w \in V \setminus V'$ and by construction of V' , w must be a convex vertex. Let $u_1 \in V'$ be the closest such vertex to w such that $u_1 \prec w$ in T . Similarly, let $u_2 \in V'$ be the closest such vertex to w such that $w \prec u_2$ in T . Note that u, v are potential candidates for u_1 and u_2 , respectively and that $u \preceq u_1 \prec u_2 \preceq v$ in T' . By definition of V' , u_1, w, u_2 all belong to a convex region C of T . By Observation 7, u_1 sees u_2 in T . Since $u_1 \prec w \prec u_2$, w lies below or on the line $\widehat{L}_{u_1 u_2}$. Coming back to the fact that u sees v in T' and $u \preceq u_1 \prec u_2 \preceq v$, the line segment $\widehat{L}_{u_1 u_2}$ must lie below or on the line segment \widehat{L}_{uv} . Putting everything together, we see that w lies below or on the line segment \widehat{L}_{uv} . Thus, u sees v in T . ◀

A.1 Extension of Theorem 1 for Continuous Terrain Guarding

Consider an instance (\widehat{T}, k, r) of CONTINUOUS TERRAIN GUARDING, where r is the number of reflex vertices in T . Using the discretization result of Friedrichs et al. (Section 2, [12]), in polynomial time we can construct a terrain $T = (V, E)$ by sub-dividing (possibly multiple times) edges of \widehat{T} , and sets X, Y , where $\widehat{V} \subseteq X \subseteq Y \subseteq V$, such that the following condition is satisfied: (T, k, r) is a yes-instance of CONTINUOUS TERRAIN GUARDING if and only if there is a set $S \subseteq X$ of size at most k that sees each vertex in Y . Equipped with the above result, we can adapt our marking schemes to consider only vertices from Y while dealing with visibilities, and marking only vertices from X for potential guard set. Using this we can obtain an instance of a restricted (NP-complete) version of DISCRETE TERRAIN GUARDING with $\mathcal{O}(r^2)$ vertices in the terrain. Also by using NP-hardness of CONTINUOUS TERRAIN GUARDING, we can obtain a polynomial kernel for the problem.

Vertex Downgrading to Minimize Connectivity

Hassene Aissi

Paris Dauphine University, France
aissi@lamsade.dauphine.fr

Da Qi Chen

Carnegie Mellon University, Pittsburgh, PA, USA
daqic@andrew.cmu.edu

R. Ravi

Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA
ravi@cmu.edu

Abstract

We consider the problem of interdicting a directed graph by deleting nodes with the goal of minimizing the local edge connectivity of the remaining graph from a given source to a sink. We introduce and study a general downgrading variant of the interdiction problem where the capacity of an arc is a function of the subset of its endpoints that are downgraded, and the goal is to minimize the downgraded capacity of a minimum source-sink cut subject to a node downgrading budget. This models the case when both ends of an arc must be downgraded to remove it, for example. For this generalization, we provide a bicriteria $(4, 4)$ -approximation that downgrades nodes with total weight at most 4 times the budget and provides a solution where the downgraded connectivity from the source to the sink is at most 4 times that in an optimal solution. We accomplish this with an LP relaxation and rounding using a ball-growing algorithm based on the LP values. We further generalize the downgrading problem to one where each vertex can be downgraded to one of k levels, and the arc capacities are functions of the pairs of levels to which its ends are downgraded. We generalize our LP rounding to get a $(4k, 4k)$ -approximation for this case.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Vertex Interdiction, Vertex Downgrading, Network Interdiction, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.5

Related Version <https://arxiv.org/pdf/1911.11229.pdf>

Funding *Hassene Aissi*: This research benefited from the support of the FMJH Program PGMO and from the support of EDF, Thales, Orange et Criteo.

R. Ravi: This material is based upon research supported in part by the U. S. Office of Naval Research under award number N00014-18-1-2099.

1 Introduction

Interdiction problems arise in evaluating the robustness of infrastructure and networks. For an optimization problem on a graph, the interdiction problem can be formulated as a game consisting of two players: an attacker and a defender. Every edge/vertex of the graph has an associated interdiction cost and the attacker interdicts the network by modifying the edges/vertices subject to a budget constraint. The defender solves the problem on the modified graph. The goal of the attacker is to hamper the defender as much as possible. Ford and Fulkerson initiated the study of interdiction problems with the maximum flow/minimum cut theorem [4, 10, 15]. Other examples of interdiction problems include matchings [17], minimum spanning trees [12, 20], shortest paths [7, 11], *st*-flows [14, 16, 18] and global minimum cuts [19, 3].



© Hassene Aissi, Da Qi Chen, and R. Ravi;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 5; pp. 5:1–5:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Most of the interdiction literature today involves the interdiction of edges while the study of interdicting vertices has received less attention (e.g.[17, 18]). The various applications for these interdiction problems, including drug interdiction, hospital infection control, and protecting electrical grids or other military installations against terrorist attacks, all naturally motivate the study of the vertex interdiction variant. In this paper, we focus on vertex interdiction problems related to the minimum st -cut (which is equal to the maximum st -flow and hence also termed network flow interdiction or network interdiction in the literature).

For st -cut vertex interdiction problems, the set up is as follows. Consider a directed graph $G = (V(G), A(G))$ with n vertices, m arcs, an arc cost function $c : A(G) \rightarrow \mathbb{N}$, and an interdiction cost function $r : V(G) \setminus \{s, t\} \rightarrow \mathbb{N}$ defined on the set of vertices $V(G) \setminus \{s, t\}$. A set of arcs $F \subseteq A(G)$ is an st -cut if $G \setminus F$ no longer contains a directed path from s to t . Define the cost of F as $c(F) = \sum_{e \in F} c(e)$. For any subset of vertices $X \subseteq V(G) \setminus \{s, t\}$, we denote its interdiction cost by $r(X) = \sum_{v \in X} r(v)$. Let $\lambda_{st}(G \setminus X)$ denote the cost of a minimum st cut in the graph $G \setminus X$.

► **Problem 1. Weighted Network Vertex Interdiction Problem (WNVIP) and its special cases.** Given two specific vertices s (source) and t (sink) in $V(G)$ and interdiction budget $b \in \mathbb{N}$, the Weighted Network Vertex Interdiction Problem (WNVIP) asks to find an interdicting vertex set $X^* \subseteq V(G) \setminus \{s, t\}$ such that $\sum_{v \in X^*} r(v) \leq b$ and $\lambda_{st}(G \setminus X^*)$ is minimum. The special case of WNVIP where all the interdiction costs are one will be termed NVIP, while the further special case when even the arc costs are one will be termed NVIP with unit costs.

In this paper, we define and study a generalization of the network flow interdiction problem in digraphs that we call **vertex downgrading**. Since interdicting vertices can be viewed as attacking a network at its vertices, it is natural to consider a variant where attacking a node does not destroy it completely but partially weakens its structural integrity. In terms of minimum st -cuts, one interpretation could be that whenever a vertex is interdicted, instead of removing it from the network we partially reduce the cost of its incident arcs. In this context, we say that a vertex is *downgraded*. Specifically, consider a directed graph $G = (V(G), A(G))$ and a downgrading cost $r : V(G) \setminus \{s, t\} \rightarrow \mathbb{N}$. For every arc $e = uv \in A(G)$, there exist four associated nonnegative costs $c_e, c_{eu}, c_{ev}, c_{euv}$, respectively representing the cost of arc e if 1) neither $\{u, v\}$ are downgraded, 2) only u is downgraded, 3) only v is downgraded, and 4) both $\{u, v\}$ are downgraded. Note that these cost functions are independent of each other so downgrading vertex v might affect each of its incident arcs differently. However, we do impose the following conditions: $c_e \geq c_{eu} \geq c_{euv}$ and $c_e \geq c_{ev} \geq c_{euv}$. These inequalities are natural to impose since the more endpoints of an arc are downgraded, the lower the resulting arc should cost. Given a downgrading set $Y \subseteq V(G) \setminus \{s, t\}$, define $c^Y : A(G) \rightarrow \mathbb{R}_+$ to be the arc cost function representing the cost of cutting e after downgrading Y .

	$u, v \notin Y$	$u \in Y, v \notin Y$	$u \notin Y, v \in Y$	$u, v \in Y$
$c^Y(e) =$	c_e	c_{eu}	c_{ev}	c_{euv}

Given a set of arcs $F \subseteq A(G)$, we define $c^Y(F) = \sum_{e \in F} c^Y(e)$.

► **Problem 2. Network Vertex Downgrading Problem (NVDP).** Let $G = (V(G), A(G))$ be a directed graph with a source s and a sink t . For every arc $e = uv$, we are given non-negative costs $c_e, c_{eu}, c_{ev}, c_{euv}$ as defined above. Given a (downgrading) budget b , find a set $Y \subseteq V(G) \setminus \{s, t\}$ and an st -cut $F \subseteq A(G)$ such that $\sum_{v \in Y} r(v) \leq b$ and minimizes $c^Y(F)$.

While it is not immediately obvious as it is for WNVIP, we can still show that detecting a zero solution for NVDP is easy.

► **Theorem 1.** *Given an instance of NVDP on graph G with budget b , there exists a polynomial time algorithm to determine if there exists $Y \subseteq V(G)$ and an st -cut $F \subseteq A(G)$ such that $\sum_{v \in Y} r(v) \leq b$ and $c^Y(F) = 0$.*

First we present some useful reductions between the above problems.

1. In the NFI (Network Flow Interdiction) problem defined in [4], the given graph is undirected instead of directed and the adversary interdicts edges instead of vertices. The goal is to minimize the cost of the minimum st -cut after interdiction. NFI can be reduced to the undirected version of WNVIP (where the underlying graph is undirected). Simply subdivide every undirected edge $e = uv$ with a vertex v_e . The interdiction cost of v_e remains the same as the interdiction cost of e while all original vertices have an interdiction cost of ∞ (or a very large number). The cut cost of the edges $uv_e, v_e v$ are equal to the original cost of cutting the edge e .
2. The undirected version of WNVIP can be reduced to the (directed) WNVIP by replacing every edge with two parallel arcs going in opposite directions. Each new arc has the same cut cost as the original edge.
3. WNVIP is a special case of NVDP with costs $c_{eu} = c_{ev} = c_{euv} = 0$ for all $e = uv$.

The first two observations above imply that any hardness result for NFI in [4] also applies to WNVIP. Based on the second observation, we prove our hardness results for the (more specific) undirected version of WNVIP. As a consequence of the third observation, all of these hardness results also carry over to the more general NVDP.

Our work also studies the following further generalization of NVDP. Every vertex has k possible levels that it can be downgraded to by paying different downgrading costs. Every arc has a cutting cost depending on what level its endpoints were downgraded to. More precisely, for each level $0 \leq i, j \leq k$, let $r_i(v)$ be the interdiction cost to downgrade v to level i and let $c_{i,j}(e)$ be the cost of cutting arc $e = uv$ if u, v were downgraded to levels i, j respectively. We assume that $0 = r_0(v) \leq r_1(v) \leq \dots \leq r_k(v)$ since higher levels of downgrading should cost more and $c_{i,j}(e) \geq c_{i',j'}(e)$ if $i \leq i', j \leq j'$ since the more one downgrades, the easier it is to cut the incident arcs. Then, given a map $L : V(G) \rightarrow \{0, \dots, k\}$, representing which level to downgrade each vertex to, one can talk about the cost of performing this downgrading: $r^L := \sum_{v \in V(G)} r_{L(v)}(v)$, and the cost of a cut F after downgrading according to L : $c^L(F) := \sum_{uv \in F} c_{L(u), L(v)}(uv)$. Now, we can formally define the most general problem we address.

► **Problem 3. Network Vertex Leveling Downgrading Problem (NVLDP).** *Let $G = (V(G), A(G))$ be a directed graph with a source s and a sink t . For every vertex v and $0 \leq i \leq k$, we have non-negative downgrading costs $r_i(v)$. For every arc $e = uv$ and levels $0 \leq i, j \leq k$, we are given non-negative cut costs $c_{i,j}(e)$. Given a (downgrading) budget b , find a map $L : V(G) \rightarrow \{0, \dots, k\}$ and an st -cut $F \subseteq A(G)$ such that $r^L \leq b$ and minimizes $c^L(F)$.*

Note that when $k = 1$ we have NVDP.

Related Works

► **Definition 2.** *An (α, β) bicriteria approximation for the interdiction (or downgrading) problem returns a solution that violates the interdiction budget b by a factor of at most β and provides a final cut (in the interdicted graph) with cost at most α times the optimal cost of a minimum cut in a solution of interdiction budget at most b .*

Chestnut and Zenklusen [4] study the network flow interdiction problem (NFI), which is the undirected and edge interdiction version of WNVIP. NFI is also known to be essentially equivalent to the budgeted minimum st cut problem [13]. NFI is also a recasting of the k -route st -cut problem [5, 9], where a minimum cost set of edges must be deleted to reduce the node or edge connectivity between s and t to be k . The results of Chestnut and Zenklusen, and Chuzhoy et al. [5] show that an $(\alpha, 1)$ -approximation for WNVIP implies a $2(\alpha)^2$ -approximation for the notorious Densest k -Subgraph (DkS) problem. The results of Chuzhoy et al. [5] (Theorem 1.9 and Appendix section B) also imply such a hardness for NVIP even with unit edge costs. For the directed version, WNVIP is equivalent to directed NFI (by subdividing arcs or splitting vertices). As noted in [18], there is a symmetry between the interdicting cost and the capacity on each arc and thus it is also hard to obtain a $(1, \beta)$ -approximation for WNVIP. Furthermore, Chuzhoy et al. [5] also show that there is no $(C, 1 + \gamma_C)$ -bi-criteria approximation for WNVIP assuming Feige’s Random κ -AND Hypothesis (for every C and sufficiently small constant γ_C). For example, under this hypothesis, they show hardness of $(\frac{11}{10} - \epsilon, \frac{25}{24} - \epsilon)$ approximation for WNVIP.

Chestnut and Zenklusen give a $2(n - 1)$ approximation algorithm for NFI for any graph with n vertices. In the special case where the graph is planar, Philips [14] gave an FPTAS and Zenklusen [18] extended it to handle the vertex interdiction case.

Burch *et al.* [2] give a $(1 + \epsilon, 1), (1, 1 + \frac{1}{\epsilon})$ pseudo-approximation algorithm for NFI. Given any $\epsilon > 0$, this algorithm returns either a $(1 + \epsilon)$ -approximation, or a solution violating the budget by a factor of $1 + \frac{1}{\epsilon}$ but has a cut no more expensive than the optimal cost. However, we do not know which case occurs *a priori*. In this line of work, Chestnut and Zenklusen [3] have extended the technique of Burch *et al.* to derive pseudo-approximation algorithms for a larger class of NFI problems that have good LP descriptions (such as duals that are box-TDI). Chuzhoy et al. [5] provide an alternate proof of this result by subdividing edges with nodes of appropriate costs.

Our Contributions

1. We define and initiate the study of multi-level node downgrading problems by defining the Network Vertex Leveling Downgrading Problem (NVLDP) and provide the first results for it. This problem extends the study in [18] of the vertex interdiction problem so as to consider a richer set of interdiction functions.
2. For the downgrading variant NVDP, we show that the problem of detecting whether there exists a downgrading set that gives a zero cost cut can be solved in polynomial time. (Section 2)
3. We design a new LP rounding approximation algorithm that provides a $(4, 4)$ -approximation to NVDP. We use a carefully constructed auxiliary graph so that the level-cut algorithm based on ball growing for showing integrality of st -cuts in digraphs (See. e.g. [6]) can be adapted to choose nodes to downgrade and arcs to cut based on the LP solution. (Section 3)
4. For the most general version NVLDP with k levels of downgrading each vertex and k^2 possible downgraded costs of cutting an edge, we generalize the LP rounding method for NVDP to give a $(4k, 4k)$ -approximation. The direct extension of the NVDP rounding to this case only gives an $O(k^2)$ approximation. However, we exploit the sparsity properties of a vertex optimal solution to our LP formulation to improve this guarantee to match that for the case of $k = 1$. Details are in the full version [1].
5. As noted before, many previous works showed hardness in obtaining a unicriterion approximation for WNVIP, which motivates the focus on finding bicriteria approximation results. We push the hardness result further to show that it is also “DkS hard” to obtain a

$(1, \beta)$ -approximation for NVIP and NVIP with unit costs even in *undirected* graphs. Note that this is in sharp contrast to the edge interdiction case. NFI in undirected graphs with unitary interdiction cost and unitary cut cost can be solved by first finding a minimum cut and then interdicting b edges in that cut [19]. Details are in the full version [1].

6. Burch *et al.* [2] gave a polynomial time algorithm that finds a $(1 + 1/\epsilon, 1)$ or $(1, 1 + \epsilon)$ -approximation for any $\epsilon > 0$ for WNVIP in digraphs. This was reproved more directly by Chuzhoy et al [5] by converting both interdiction and arc costs into costs on nodes. We show that this strategy can also be extended to give a simple $(4, 4(1 + \epsilon))$ -bicriteria approximation for the multiway cut generalization in directed graphs and a $(2(1 + \epsilon) \ln k, 2(1 + \epsilon) \ln k)$ -approximation for the multicut vertex interdiction problem in undirected graphs, for any $\epsilon > 0$, where k is the number of terminal nodes in the multicut problem. Details are in the full version [1].

2 Detecting Zero in NVDP in Polynomial Time

In this section, we provide an algorithm to detect, in a given instance of NVDP, whether there exists nodes to downgrade such that the downgrading cost is less than the budget and the min cut after downgrading is zero, and hence prove Theorem 1.

In order to demonstrate the main idea of the proof, we first work on a special case of NVDP. Suppose for every arc $e = uv$, $c_e = c_{eu} = c_{ev} = 1$ and $c_{euw} = 0$. In other words, every arc is unit cost and requires the downgrading of both ends in order to reduce the cost down to zero. For every vertex $v \in V(G)$, we assume the interdiction cost $r(v) = 1$. We call this the **Double-Downgrading Network Problem** (DDNP). We first prove the following.

► **Lemma 3.** *Given an instance of DDNP on graph G with budget b , there exist a polynomial time algorithm to determine if there exists $Y \subseteq V(G)$ and an st -cut $F \subseteq A(G)$ such that $|Y| \leq b$ and $c^Y(F) = 0$.*

Proof. Let $X \subseteq V(G)$ be a minimum set of vertices to downgrade such that the resulting graph contains a cut of zero cost. Let F be the set of arcs in the graph induced by X (i.e., with both ends in X). Note that F are the only arcs with cost zero and hence F is an arc cut in G . Furthermore, since X is optimal, X is the set of vertices incident to F (there are no isolated vertices in the graph induced by X). Let V_s, V_t be the set of vertices in the component of $G \setminus F$ that contains s, t respectively.

Consider the graph G^2 where we add arc uv to G if there exists $w \in V(G)$ such that $uw, vw \in A(G)$. First we claim that X is a vertex cut in G^2 . Suppose there is an st path in $G^2 \setminus X$ where the first arc crossing over from V_s to V_t is uv . Note that any such $u \in V_s \setminus X$ and $v \in V_t \setminus X$ are distance 3 apart and hence do not have an arc between them in G^2 , a contradiction.

Given any vertex cut Y in G^2 , we claim that downgrading Y in G creates an st -cut of zero cost, by deleting the arcs induced by Y from G . Suppose for a contradiction there is an st -path after downgrading Y and deleting the zero-cost arcs induced by Y . Then the path cannot have two consecutive nodes in Y . Let $y \in Y$ be a single node along the path with neighbors $y^-, y^+ \notin Y$. Note that $(y^-, y^+) \in G^2$, and shortcutting over all such single node occurrences from Y in the path gives us an st -path in $G^2 \setminus Y$, a contradiction.

This proves that a minimum size downgrading vertex set Y in G whose downgrading produces a zero-cost st -cut is also a minimum vertex-cut in G^2 . Then, one can check if a zero-cut solution exists with budget b for DDNP by simply checking if the minimum vertex-cut in G^2 is at most b . ◀

5:6 Vertex Downgrading to Minimize Connectivity

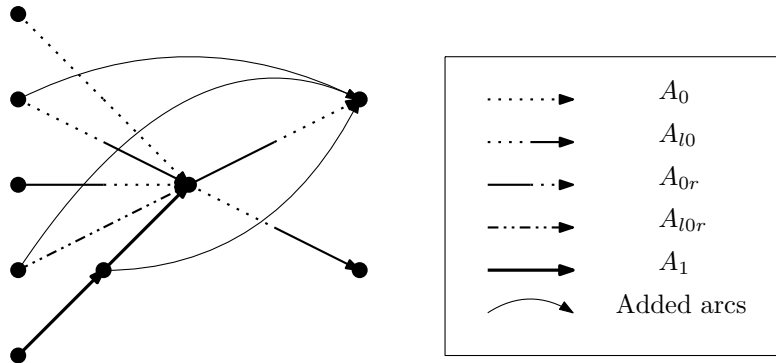


Figure 1 Example of Added Arcs in H .

Now, to prove Theorem 1, we have to slightly modify the graph G and the construction of G^2 in order to adapt to the various costs. Our goal is still to look for a minimum vertex cut in an auxiliary graph using $r(v)$ as vertex cost.

Proof. Given an instance of NVDP on G with a budget b , vertex downgrading costs $r(v)$ and arc costs $c_e, c_{eu}, c_{ev}, c_{euv}$, consider the following auxiliary graph H . First, we delete any arc e where $c_e = 0$ since they are free to cut anyways. For every arc $e = uv$ where $c_{euv} > 0$, subdivide e with a vertex t_e and let $r(t_e) = \infty$. In some sense, since $c_e, c_{eu}, c_{ev} \geq c_{euv} > 0$, downgrading u, v cannot reduce the cost of e to zero. Then, we should never be allowed to touch the vertex t_e . Let T be the set of all newly-added subdivided vertices. To finish constructing H , our next step is to properly simulate H^2 .

We classify arcs into five types based on which of its costs are zero. Note that we no longer have any arcs where $c_e = 0$. Let $A_0 := \{e = uv : c_{eu} = c_{ev} = c_{euv} = 0\}$, the arcs where downgrading either ends reduce its cost to zero. Let $A_{l0} := \{e = uv : c_{eu} = c_{euv} = 0, c_{ev} > 0\}$, $A_{0r} := \{e = uv : c_{ev} = c_{euv} = 0, c_{eu} > 0\}$, $A_{l0r} := \{e = uv : c_{euv} = 0, c_{eu}, c_{ev} > 0\}$ respectively represent arcs that require the downgrading of its left tail, its right head, or both in order to reduce its cost. Let A_1 be all remaining arcs, those incident to the newly subdivided vertex t_e . Now, for every path uvw of length two, we consider adding the arc uw based on the following rules (see Figure 1 for example of newly added arcs):

		If $v \notin T$				
Add uw ?	$vw \in$					
$uv \in$		A_0	A_{l0}	A_{0r}	A_{l0r}	A_1
A_0		No	No	No	No	No
A_{l0}		No	No	Yes	Yes	Yes
A_{0r}		No	No	No	No	No
A_{l0r}		No	No	Yes	Yes	Yes
A_1		No	No	Yes	Yes	Yes
If $v = t_e \in T$, do not add uw						

The idea is similar to the proof for DDNP. If $uv, vw \in A_{l0r}$, downgrading v is not enough to cut uv, vw for free. Thus we add arc uw to keep the connectivity. If $uv \in A_{0r}$, then downgrading v should reduce the cost of uv to 0. Thus, we do not want to bypass v by adding an arc uw . If $v = t_e \in T$, since $r(v)$ has high cost, we never cut it so we do not need to strengthen the connectivity by adding arcs uw .

Let (X, F) be a solution to NVDP where $\sum_{v \in X} r(v)$ is minimum, F is an st -cut and $c^X(F) = 0$. Let V_s be all vertices connected to s in $G \setminus F$. We claim that X is a vertex cut in H . Suppose not and there exists an st -path in H and let uv be the first arc of the path leaves V_s . If $v = t_e \in T$, then arc $e \in F$, contradicting $c^X(F) = 0$. If $uv \in A(G)$, then $uv \in F$. Since $u, v \notin X$, $c^X(uv) > 0$, a contradiction. If uv is a newly added arc, then there exist $v' \in V(G)$ such that $uv'v$ is a path in G . By definition, $V_s \cap T = \emptyset$ so $u, v \notin T$. Then, there are only four cases where we add arc uv to create H . In all cases, downgrading v' does not reduce the cost of $uv'', v'v$ to 0. Since at least one of $uv', v'v \in F$, it contradicts $c^X(F) = 0$.

Given a minimum vertex cut Y in H , we claim downgrading Y in G creates an st -cut of zero cost. Note that $Y \cap T = \emptyset$ since any vertex in T is too expensive to cut. Suppose for a contradiction there is an st -path P that does not cross an arc with cost 0 after downgrading Y . Let P' be the corresponding path in H . If P contains two consecutive vertices $u, v \in Y$, then $c_{eu} > 0$ and it would have been subdivided. This implies there are no consecutive vertices of Y in P' . Let uvw be a segment of P' where $v \in Y$. Since downgrading v does not reduce its incident arcs to a cost of 0, it follows that $uv \in A_{l_0} \cup A_{l_{0r}} \cup A_1$ and $vw \in A_{0r} \cup A_{l_{0r}} \cup A_1$. Then it follows that $uw \in A(H)$. Then, every vertex $v \in Y \cap V(P')$ can be bypassed, a contradiction.

This implies a minimum vertex cut in H is a downgrading set that creates a zero-cost cut in G . Then, by checking the min-vertex cut cost of H , we can determine whether a zero-solution exists for G with budget b . ◀

3 Approximating Network Vertex Downgrading Problem (NVDP)

As an introduction and motivation to the LP model and techniques used to solve NVLDP, in this section, we focus on the special case NVDP, where there is only one other level to downgrade each vertex to. Our main goal is to show the following theorem.

► **Theorem 4.** *There exists a polynomial time algorithm that provides a (4, 4)-approximation to NVDP on an n -node digraph.*

3.1 LP Relaxation and Rounding

LP Model for Minimum st -cut. To formulate the NVDP as a LP, we begin with the following standard formulation of minimum st -cuts [8].

$$\min \quad \sum_{e \in A(G)} c(e)x_e$$

$$\text{s.t.} \quad d_v \leq d_u + x_{uv} \quad \forall uv \in A(G) \quad (1)$$

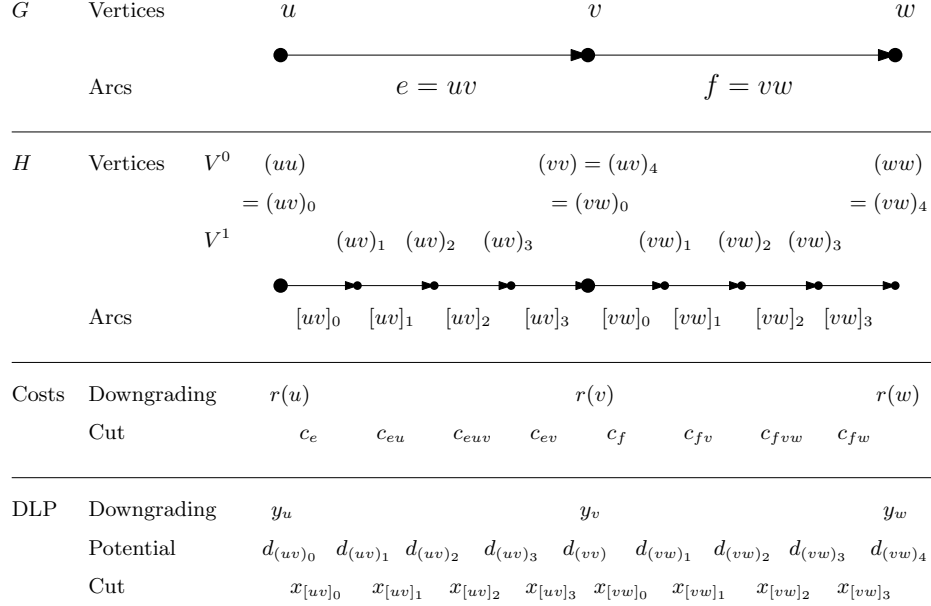
$$d_s = 0, d_t \geq 1$$

$$x_{uv} \geq 0 \quad \forall uv \in A(G) \quad (2)$$

An integer solution of this problem can be seen as setting d to be 0 for nodes in the s shore and 1 for nodes in the t shore of the cut. Constraints (1) then insist that the x -value for arcs crossing the cut to be set to 1. The potential d_v at node v can also be interpreted as a distance label starting from s and using the nonnegative values x_{uv} as distances on the arcs. Any optimal solution of the above LP can be rounded to an optimal integer solution of no greater value by using the x -values on the arcs as lengths, growing a ball around s , and cutting it at a random threshold between 0 and the distance to t (which is 1 in this case). The expected cost of the random cut can be shown to be the LP value (See e.g., [6]), and the minimum such ball can be found efficiently using Dijkstra's algorithm. Our goal in this section is to generalize this formulation and ball-growing method to NVDP.

5:8 Vertex Downgrading to Minimize Connectivity

One difficulty in NVDP comes from the fact that every arc has four associated costs and we need to write an objective function that correctly captures the final cost of a chosen cut. One way to overcome this issue is to have a distinct arc associated with each cost. In other words, for every original arc $uv \in A(G)$, we create four new arcs $[uv]_0, [uv]_1, [uv]_2, [uv]_3$ with cost $c_e, c_{eu}, c_{euw}, c_{ev}$ respectively. Then, every arc has its unique cost and it is now easier to characterize the final cost of a cut. We consider the following auxiliary graph. See Figure 2.



■ **Figure 2** Construction of the auxiliary graph H .

Constructing the Auxiliary Graph H

Let $V(H) = V^0(H) \cup V^1(H)$ where $V^0(H) = \{(vv) : v \in V(G)\}$ and $V^1(H) = \{(uv)_i : uv \in A(G), i = 1, 2, 3\}$. Define $A(H) = \{[uv]_0 = (uu)(uv)_1, [uv]_1 = (uv)_1(uw)_2, [uv]_2 = (uw)_2(uw)_3, [uv]_3 = (uw)_3(vv) : uv \in A(G)\}$. Essentially, the vertices $(uu) \in V^0(H)$ correspond to the original vertices $u \in V(G)$ and for every arc $uv \in A(G)$, we replace it with a path $(uu)(uv)_1(uw)_2(uw)_3(vv)$ where the four arcs on the path are $[uv]_0, [uv]_1, [uv]_2, [uv]_3$. For convenience and consistency in notation, we define $(uv)_0 := (uu)$, $(uv)_4 := (vv)$. Note that the vertices of H will always be denoted as two lowercase letters in parenthesis while arcs in H will be two lowercase letters in square brackets with subscript $i = 0, 1, 2, 3$. The cost function $c : A(H) \rightarrow \mathbb{R}_{\geq 0}$ is as follows: $c([uv]_0) = c_e$, $c([uv]_1) = c_{eu}$, $c([uv]_2) = c_{euw}$, $c([uv]_3) = c_{ev}$. Since we can only downgrade vertices in V^0 , to simplify the notation, we retain $r(v)$ as the cost to downgrade vertex $(vv) \in V^0$. Note that $|V(H)| = 3|A(G)| + |V(G)| = O(n + m)$.

Downgrading LP

Given the auxiliary graph H , we can now construct an LP similar to the one for st -cuts. For vertices $(vv) \in V^0(H)$ corresponding to original vertices of G , we define a downgrading variable y_v representing whether vertex v is downgraded or not in G . For every arc $[uv]_i \in A(H)$, we have a cut variable $x_{[uv]_i}$ to indicate if the arc belongs in the final cut of the graph. Lastly for all vertices $(uv)_i \in V(H)$, we have a potential variable $d_{(uv)_i}$ representing its distance from the source (ss).

The idea is to construct an LP that forces s, t to be at least distance 1 apart from each other as before. This distance can only be contributed from the arc variables $x_{[uv]_i}$. The downgrading variables y_v imposes limits on how large these distances $x_{[uv]_i}$ of some of its incident arcs can be. The motivation is that the larger y_u and y_v are, the more we should allow arc $[uv]_2$ to appear in the final cut over the other arcs $[uv]_0, [uv]_1, [uv]_3$ in order to incur the cheaper cost of c_{euv} . We consider the following downgrading LP henceforth called DLP.

Figure 2 includes the list of variables associated with H . In the LP, our objective is to minimize the cost of the final cut. Constraint (3) corresponds to the budget constraint for the downgrading variables. Constraint (4) is analogous to Constraint (1) in the LP for min-cuts.

Constraint (5) relates cut and downgrade variables. If we do not consider any constraint related to downgrading variables for a moment, the LP will naturally always want to choose the cheapest arc $[uv]_2$ over $[uv]_0, [uv]_1, [uv]_3$ when cutting somewhere between (uu) and (vv) . However, the cut should not be allowed to go through $[uv]_2$ if one of u, v is not downgraded. In other words $x_{[uv]_2}$ should be at most the minimum of y_u, y_v . This reasoning gives the constraint $x_{[uv]_2}, x_{[uv]_3}, x_{[vw]_1}$ and, $x_{[vw]_2}$ all need to be $\leq y_v$ for in-arcs uv and out-arcs vw . Now consider an arc $f = vw \in E(G)$. In an integral solution, if v is downgraded, the arc vw incurs a cost of either c_{fv} or c_{fvw} but not both, since v must lie on one side of the cut. This translates to a LP solution where only one of the arcs $[vw]_1, [vw]_2$ is in the final cut. Thus, a better constraint to impose is $x_{[vw]_1} + x_{[vw]_2} \leq y_v$. We can also similarly insist that $x_{[uv]_2} + x_{[uv]_3} \leq y_v$ for in-arcs uv . To push this even further, consider a path uvw in G . In an integral solution, at most one of the arcs uv, vw appears in the final cut. This implies that if v is downgraded, then only one of the costs $c_{ev}, c_{euv}, c_{fv}, c_{fvw}$ is incurred. This corresponds to the tighter constraint (5). Note that for every vertex $v \in V(G)$, for every pair of incoming and outgoing arcs of v , we need to add one such constraint. Then for every vertex in G , we potentially have to add up to n^2 many constraints. In total, the number of constraints would still only be $O(n^3)$. The last few constraints in DLP make sure s and t are 1 distance apart and cannot themselves be downgraded. The final LP relaxation is given below.

$$\begin{aligned}
 \min \quad & \sum_{[uv]_i \in A(H)} c([uv]_i)x_{[uv]_i} \\
 \text{s.t.} \quad & \sum_{(vv) \in V^0(H)} r(v)y_v \leq b \tag{3} \\
 & d_{(uv)_{i+1}} \leq d_{(uv)_i} + x_{[uv]_i} \quad \forall \text{arc } [uv]_i, 0 \leq i \leq 3 \tag{4} \\
 & x_{[uv]_2} + x_{[uv]_3} + x_{[vw]_1} + x_{[vw]_2} \leq y_v \quad \forall \text{path } (uv)_3(vv)(vw)_1 \tag{5} \\
 & d_{(ss)} = 0, d_{(tt)} = 1, y_s = 0, y_t = 0
 \end{aligned}$$

The following lemmas shows the validity of our defined DLP for NVDP.

► **Lemma 5.** *An optimal solution to NVDP provides a feasible integral solution to DLP with the same cost.*

Proof. Given a digraph G with cost functions $c_e, c_{eu}, c_{ev}, c_{euv}$, a source s and a sink t , let $Y \subseteq V(G), F \subseteq A(G)$ be an optimal solution to NVDP where $r(Y) \leq b, F$ is an st -cut and $c^Y(F)$ is minimum. Then, a feasible solution (x, y, d) to DLP on the graph H can be constructed as follows:

- For the cut variables x , let
 - $x_{[uv]_0} = 1$ if $uv \in F$ and $u, v \notin Y$, 0 otherwise,
 - $x_{[uv]_1} = 1$ if $uv \in F$ and $u \in Y, v \notin Y$, 0 otherwise,

5:10 Vertex Downgrading to Minimize Connectivity

- $x_{[uv]_2} = 1$ if $uv \in F, u, v \in Y$, 0 otherwise,
- $x_{[uv]_3} = 1$ if $uv \in F, u \notin Y, v \in Y$, 0 otherwise.
- For the downgrading variables y , let $y_u = 1$ if $u \in Y$, 0 otherwise.
- For the potential variables d , let $d_{[uv]_i} = 0$ if $[uv]_i \in S$ and 1 otherwise,

where we define S, T as follows. Let F^* be the set of arcs in H whose x variable is 1. We claim that F^* is an st -cut in H . Note that every st -path Q in H corresponds to an st -path P in G . Then, there is an arc uv in P that is also in F . Then, it follows from construction that the x value for one of $[uv]_0, [uv]_1, [uv]_2, [uv]_3$ is 1 and thus there exists $i = 0, 1, 2, 3$ such that $[uv]_i \in F^*$. Note that $[uv]_i$ is also in Q . Therefore F^* is an st -cut in H . Then, let S be the set of vertices in $H \setminus F^*$ that is connected to the source s and let $T = V(H) \setminus S$.

Note that by construction, (x, y, d) is integral and is a feasible solution to DLP. The final objective value $\sum_{[uv]_i \in A(H)} c([uv]_i) x_{[uv]_i} = \sum_{[uv]_i \in F^*} c([uv]_i)$ and by construction, it matches the cost $c^Y(F^*)$. ◀

► **Lemma 6.** *An integral solution (x^*, y^*, d^*) to DLP with objective value c^* corresponds to a feasible solution (Y^*, E^*) to NVDP such that $c^{Y^*}(E^*) \leq c^*$.*

Proof. Given a directed graph G and its auxiliary graph H , let (x^*, y^*, d^*) be an optimal integral solution to DLP with an objective value of c^* . Let $F^* \subseteq A(H)$ be the set of arcs whose x^* value is 1. Let $Y^* \subseteq V^0(H)$ whose y value is 1. Let $E^* \subseteq A(G) = \{uv \in A(G) : [uv]_i \in F^* \text{ for some } i = 0, 1, 2, 3\}$ be the set of original arcs of those in F^* .

Note that by construction, Y^* does not violate the budget constraint. Every st -path in G corresponds directly to an st -path in H . Since F^* is an st -cut in H , it follows that E^* is an st -cut in G . Then it remains to show that $c^* \geq c^{Y^*}(E^*)$.

Note that

$$c^* = \sum_{[uv]_i \in A(H)} c([uv]_i) x_{[uv]_i}^* = \sum_{e=uv \in A(G)} c_e x_{[uv]_0}^* + c_{eu} x_{[uv]_1}^* + c_{euv} x_{[uv]_2}^* + c_{ev} x_{[uv]_3}^*.$$

Meanwhile, note that $c^{Y^*}(E^*) = \sum_{e=uv \in A(G)} c^{Y^*}(e)$. Thus, it suffices to prove the following claim.

▷ **Claim 7.** For every arc $e = uv \in A(G)$, $\sum_{i=0}^3 c([uv]_i) x_{[uv]_i}^* \geq c^{Y^*}(e)$.

First, note that if $e = uv \notin E^*$, then $c^{Y^*}(e) = 0$ by definition of E^* . Then, the inequality is trivially true. Thus, we may assume $uv \in E^*$ which implies there exist $i = 0, 1, 2, 3$ such that $[uv]_i \in F^*$ and $x_{[uv]_i}^* = 1$. We will now break into cases depending on whether $u, v \in Y^*$.

Suppose $u, v \notin Y^*$. Then, $y_u^* = y_v^* = 0$ and by constraint (5) in DLP, it follows that the x^* value for $[uv]_1, [uv]_2, [uv]_3$ are all 0. Then, $[uv]_0 \in F^*$ and $\sum_{i=0}^3 c([uv]_i) x_{[uv]_i}^* = c_e = c^{Y^*}(e)$.

Now, assume $u \in Y^*, v \notin Y^*$. By constraint (5), $x_{[uv]_2}^* + x_{[uv]_3}^* \leq y_v^* = 0$ and thus only the x^* value for $[uv]_0, [uv]_1$ can be 1. Since we have an integral solution, it follows that $x_{[uv]_0}^* + x_{[uv]_1}^* \geq 1$, since $e \in E^*$. Note that $c_e \geq c_{eu}$. Then $\sum_{i=0}^3 c([uv]_i) x_{[uv]_i}^* = c_e x_{[uv]_0}^* + c_{eu} x_{[uv]_1}^* \geq c_{eu} (x_{[uv]_0}^* + x_{[uv]_1}^*) \geq c_{eu} = c^{Y^*}(e)$. Note that a similar argument can be made for the case when $u \notin Y^*, v \in Y^*$.

Lastly, assume both $u, v \in Y^*$. Then $c^{Y^*}(e) = c_{euv}$. Note that $c_e, c_{eu}, c_{ev} \geq c_{euv}$. Then, $\sum_{i=0}^3 c([uv]_i) x_{[uv]_i}^* \geq \sum_{i=0}^3 c_{euv} x_{[uv]_i}^* \geq c_{euv}$. The last inequality is due to the fact that there exists $i = 0, 1, 2, 3$ such that $[uv]_i \in F^*$. This completes the proof of claim and thus also our lemma. ◀

Bicriteria Approximation for NVDP

We now prove Theorem 4. We will work with an optimal solution of DLP defined on the auxiliary graph H . The idea is to use a ball-growing algorithm that greedily finds cuts until one with the promised guarantee is produced. The reason this algorithm is successful is proved by analyzing a randomized algorithm that picks a number $0 \leq \alpha \leq 1$ uniformly at random and chooses a cut at distance α from the source s . Then we choose vertices to downgrade and arcs to cut based on arcs in this cut at distance α . By computing the expected downgrading cost and the expected cost of the cut arcs, the analysis will show the existence of a level cut that satisfies our approximation guarantee.

To achieve the desired result, we cannot work with the graph H directly. This is because the ball-growing algorithm works only if the probability of cutting some arc can be bounded within some range. This bound exists for the final cut arcs (as in the proof for st -cuts) but not for the final downgraded vertices. Consider a vertex v ; it is downgraded if any arc of the form $[uv]_2, [uv]_3, [vw]_1, [vw]_2$ is cut in H . Thus it has the potential of being cut anywhere between the range of the vertices $(uv)_2$ and $(vw)_3$. We would like to use Constraint (5) to bound this range but we cannot do this directly since we do not know the length of the arc $[vw]_0$ which also lies in this range. To circumvent this difficulty and properly employ Constraint (5), we will construct a reduced graph H' obtained by contracting some arcs.

Let (x^*, y^*, d^*) be an optimal solution to DLP where the optimal cost is c^* . It follows from the validity of our model that c^* is at most the cost of an optimal integral solution.

Constructing Graph H'

For every arc $uv \in A(G)$, we compare the value $x_{[uv]_0}^*$ and $x_{[uv]_1}^* + x_{[uv]_2}^* + x_{[uv]_3}^*$. The reason we separate this way is because the variables in the second term are influenced by the downgrading values on u, v . Thus the more we downgrade u and v , the larger we are allowed to increase the second sum, and the more length we can place between u and v in these variables. For an arc $uv \in A(G)$, if $x_{[uv]_0}^* < x_{[uv]_1}^* + x_{[uv]_2}^* + x_{[uv]_3}^*$, we say uv is an *aided* arc since the majority of its length is contributed by the downgrading values on the u, v and thus the downgrading values help to generate its length. For all other arcs, we say uv is an *unaided* arc since more of its length would be contributed by the arc $[uv]_0$, corresponding to simply paying for the original cost of deletion c_e without the aid from downgrading. To construct H' , if uv is an aided arc, then contract $[uv]_0$. Otherwise, contract $[uv]_1, [uv]_2, [uv]_3$.

Consider a path $P = (uv)_0(uv)_1(uv)_2(uv)_3(uv)_4$ in H . Note that the length of this path is shortened in H' depending on whether uv is an aided/unaided arc. However, since we always retain the larger of $x_{[uv]_0}^*$ and $x_{[uv]_1}^* + x_{[uv]_2}^* + x_{[uv]_3}^*$ in H' , the path's length is at most halved. Then it follows that the distance between any two vertices in H' is reduced to at most half its original value in H . In particular, it follows that the shortest path between the source and the sink is at least $1/2$. This property will be crucial in arguing that the solution chosen by our algorithm has low cost relative to the LP optimum.

We make one last adjustments to the weight of aided arcs. Let $D^*((uv)_i)$ be the shortest path distance from the source (ss) to the vertex $(uv)_i$ viewing x^* as lengths in H' . Consider a path $[uv]_1[uv]_2[uv]_3$ of an aided arc. Note that the distances of nodes $(uu), (uv)_2, (uv)_3$ are strictly increasing but $D^*((vv))$ might be strictly less than $D^*((uv)_3)$ (e.g. via an alternate shorter path to (vv) avoiding (uu)). In fact $D^*((vv))$ might even be smaller than $D^*((uu))$. This makes the analysis of the usage of arcs of the form $[uv]_i$ in the cutting procedure difficult. To avoid this difficulty, for any aided arcs uv where $D^*((vv)) < D^*((uu))$, replace the path $[uv]_1[uv]_2[uv]_3$ with a single dummy arc. Then we redefine a new weight variable x' : for every

5:12 Vertex Downgrading to Minimize Connectivity

aided arc uv , where $0 \leq D^*((vv)) - D^*((uu)) < \sum_{i=1}^3 x^*_{[uv]_i}$, let $x'_{[uv]_i} = x^*_{[uv]_i} \frac{D^*((vv)) - D^*((uu))}{\sum_{i=1}^3 x^*_{[uv]_i}}$. The weight of all dummy arcs are 0. For all other arcs, the x' variables stay the same. This step guarantees that for all aided arcs that have not been replaced by dummy arcs, the distances of (uu) , $(uv)_2$, $(uv)_3$, (vv) are in non-decreasing order. For those that have been replaced by the dummy arc, we will ensure that these arcs do not occur in any cut chosen in our algorithm. Two important things to keep in mind: $x' \leq x^*$ for any arc while the distance of any vertex from the source remains unchanged. In particular $D^*((tt))$ remains at least $1/2$. Our ball growing algorithm uses the modified distances x' .

■ **Algorithm 1** Ball-Growing Algorithm for NVDP.

Require: a graph G and its auxiliary graph H' with non-negative arc-weights $x'_{[uv]_i}$, source (ss) , sink (tt) , arc cut costs $c([uv]_i)$ and vertex downgrading costs $r(v)$

Ensure: a vertex set V' and an arc cut E' of G such that $\sum_{v \in V'} r(v) \leq 4b$, $c^{V'}(E') \leq 4c^*$

- 1: initialization $V = \{(ss)\}$, $D((uv)_i) = 1$ for all $(uv)_i \in V(H')$
- 2: **repeat**
- 3: let $X' \subseteq A(H')$ be the cut induced by V
- 4: find $[uw]_i = (uv)_i(uv)_{i+1} \in X'$ minimizing $D((uv)_i) + x'_{[uw]_i}$
- 5: update by adding $(uv)_{i+1}$ to V , update $D((uv)_{i+1}) = D((uv)_i) + x'_{[uw]_i}$
- 6: let $E' = \{uv \in A(G) : \{[uv]_0, [uv]_1, [uv]_2, [uv]_3\} \cap X' \neq \emptyset\}$ and $V' = \{v \in V(G) : \{[uv]_2, [uv]_3, [vw]_1[vw]_2\} \cap X' \neq \emptyset \text{ for some } u, w \in V(G)\}$
- 7: **until** $\sum_{v \in V'} r(v) \leq 4b$ and $c^{V'}(E') \leq 4c^*$
- 8: output the set V', E'

Algorithm 1 is simply a restatement of Dijkstra's algorithm run on H' . It follows the general ball-growing technique and looks at cuts X' at various distances from the source. Note that the algorithm adds at least one vertex to a node set V at each iteration so it runs for at most $|V(H')| = O(m)$ steps when applied to the graph H' (Recall that m denotes the number of arcs in the original graph G).

At each iteration, the algorithm computes a cut $X' \subseteq A(H')$ and considers the set E' of original arcs associated to those in X' and the vertex set V' representing the set of vertices we should downgrade based on the arcs in X' . For example, if $[uv]_2 \in X'$, then we should downgrade both u and v . Note that every chosen cut only contains arcs $[uv]_i$ where $D((uv)_i) \leq D((uv)_{i+1})$ so they do not contain any dummy arcs. Thus we can essentially ignore dummy arcs in accounting for the cost of the chosen cut. Furthermore, since X' is a cut in H' , it follows that E' is a cut in G .

To argue the validity of the algorithm, we show that there exists a cut X' at some distance $\alpha \leq D(tt)$ from the source such that the associated sets V', E' provides the approximation guarantee.

► **Lemma 8.** *There exists X', V', E' such that $\sum_{v \in V'} r(v) \leq 4b$, $c^{V'}(E') \leq 4c^*$*

The main idea of the proof is to pick a distance uniformly at random between zero and the distance of (tt) (which is at least half) and study the cut at that distance. We claim that the extent to which an arc is cut (chosen in E' above) in the random cut is at most twice its x^* -value, using the fact that the range of this arc is at most its x^* -value and the range of the cutting threshold is at least half. When nodes are chosen in the random cut (in V' above) to be downgraded, we argue that the range of cutting any node is at most the maximum of the values in the left hand side of the constraints (5) corresponding to this node, which in turn is at most its y^* -value. Again, since the range of the cutting threshold

is at least half, we infer that the probability of downgrading a node in the cutting process is at most twice its y^* -value. To obtain a cut where we simultaneously do not exceed both bounds, we use Markov's inequality to argue a probability of at least half of being within twice these respective expectations, hence giving us a single cut with both bounds within four times their respective LP values. The detailed proof follows.

Proof of Lemma 8. Let $D((uv)_i)$ be the shortest-path distance from the source (ss) to any vertex $(uv)_i \in V(H')$ viewing the x' variables as lengths. Note that $D((tt)) \geq 1/2$ since the original distance is at least 1 and H' reduces the distance by at most $1/2$. Note that the triangle-inequality holds under this distance metric where $D((uv)_i) - D((u'v')_{i'})$ is at most the distance between $(uv)_i$ and $(u'v')_{i'}$.

Defining the Random Variables. Let α be chosen uniformly at random from the interval $[0, D((tt))]$. Consider $X_\alpha := \{[uv]_i \in A(H') : D((uv)_i) \leq \alpha < D((uv)_{i+1})\}$, the cut at distance α in H' . Let $E_\alpha = \{uv \in A(G) : [uv]_i \in X_\alpha \text{ for some } i = 0, 1, 2, 3\}$, representing the original arcs corresponding to those in X_α . Let $V_\alpha = \{v \in V(G) : \{[uv]_2, [uv]_3, [vw]_1, [vw]_2\} \cap X_\alpha \neq \emptyset \text{ for some } u, w \in V(G)\}$, representing the set of vertices we should downgrade so that the final cost of the arcs E_α matches the cost associated to X_α . More precisely, we want $c^{V_\alpha}(E_\alpha) = \sum_{[uv]_i \in X_\alpha} c([uv]_i)$. Note that by construction E_α is an st -cut in G . Let $\mathcal{V} = \sum_{v \in V_\alpha} r(v)$, $\mathcal{E} = c^{V_\alpha}(E_\alpha)$. Our goal is to show that these two random variables \mathcal{V}, \mathcal{E} have low expectations and obtain our approximation guarantee using Markov's inequality. In particular, we will prove that $\mathbb{E}[\mathcal{V}] \leq 2b$, and that $\mathbb{E}[\mathcal{E}] \leq 2c^*$ where c^* is the optimal value of DLP.

To understand \mathcal{E} , for every arc $e = uv \in A(G)$, we introduce the indicator variables \mathcal{E}_e to be 1 if arc $e \in E_\alpha$ and 0 otherwise. Then $\mathcal{E} = \sum_{e \in A(G)} \mathcal{E}_e c^{V_\alpha}(e)$. To study the value of $\mathcal{E}_e c^{V_\alpha}(e)$, we can break into several cases depending on which arc $[uv]_i \in X_\alpha$. Note that if $[uv]_i \notin X_\alpha$ for $i = 0, 1, 2, 3$, then $e \notin E_\alpha$ and $\mathcal{E}_e c^{V_\alpha}(e) = 0$. Next, if we assume $[uv]_i \in X_\alpha$, then one can check that $c^{V_\alpha}(e) \leq c([uv]_i)$ as in the proof of Claim 7.

Slightly abusing the notation, define the indicator variable $\mathcal{E}_{[uv]_i}$ for arc $[uv]_i \in A(H)$ to be 1 if $[uv]_i \in X_\alpha$ and 0 otherwise. Then, we can upper-bound the expectation of \mathcal{E} using conditional expectations of the events $\mathcal{E}_{[uv]_i} = 1$ as follows.

$$\begin{aligned} \mathbb{E}[\mathcal{E}] &= \sum_{e \in A(G)} \mathbb{E}[\mathcal{E}_e c^{V_\alpha}(e)] \\ &= \sum_{e \in A(G)} \sum_{i=0}^3 \mathbb{E}[c^{V_\alpha}(e) | \mathcal{E}_{[uv]_i} = 1] \cdot \Pr[\mathcal{E}_{[uv]_i} = 1] \\ &\leq \sum_{e \in A(G)} \sum_{i=0}^3 c([uv]_i) \Pr[\mathcal{E}_{[uv]_i} = 1] \end{aligned}$$

To understand the probability of $\mathcal{E}_{[uv]_i} = 1$, note that an arc $[uv]_i \in X_\alpha$ if and only if $D((uv)_i) \leq \alpha < D((uv)_{i+1})$. Then, $\Pr[[uv]_i \in X_\alpha] \leq (D((uv)_{i+1}) - D((uv)_i)) / D((tt)) \leq 2x'_{[uv]_i} \leq 2x^*_{[uv]_i}$ since $D((tt)) \geq 1/2$. Combining with the previous inequalities, we see that

$$\begin{aligned} \mathbb{E}[\mathcal{E}] &\leq \sum_{uv \in A(G)} \sum_{i=0}^3 c([uv]_i) \Pr[\mathcal{E}_{[uv]_i} = 1] \\ &\leq \sum_{uv \in A(G)} \sum_{i=0}^3 c([uv]_i) 2x^*_{[uv]_i} = 2c^*. \end{aligned}$$

Next, we show a similar result for \mathcal{V} . Note that $\mathbb{E}[\mathcal{V}] = \sum_{v \in V(G)} r(v) \cdot \Pr[v \in V_\alpha]$. Recall that $v \in V_\alpha$ if and only if there exists a vertex u or w such that at least one of $[uv]_2, [uv]_3, [vw]_1, [vw]_2 \in X_\alpha$. Note that if uv is an unaided arc, then $[uv]_2, [uv]_3$ would have been contracted in H' and would never be chosen in X_α . If uv is an aided arc that was turned into a dummy arc, it would also never be chosen in the final cut. Therefore, we only need to consider aided arcs that have not been turned into dummy arcs. In order

to upper-bound the probability of choosing v into V_α , we thus need to find the range of possible α that might affect v . For any vertex $v \in V(G)$, it follows that we only need to examine aided arcs incident to the vertex v . Let $u \in V(G)$ such that $uv \in A(G)$, uv is an aided arc and $D((uv)_2)$ is minimum. Let $w \in V(G)$ such that vw is an aided arc and $D((vw)_3)$ is maximum. Note that for any aided arcs zv, vz' that are not replaced by a dummy arc, $D((zz)) \leq D((zv)_2) \leq D((zv)_3) \leq D((vv)) \leq D((vz')_2) \leq D((vz')_3) \leq D((z'z'))$ by our choice of x' . For all such arcs of the form $[zv]_2, [zv]_3, [vz']_1, [vz']_2$, their extremities are in the distance range $D((uv)_2), D((vw)_3)$. Then, v is chosen only if α is between $D((uv)_2)$ and $D((vw)_3)$. The distance between $(uv)_2$ and $(vw)_3$ is upper-bounded by the length of a shortest path in H' . Since vw is an aided arc, $[vw]_0$ is contracted in H' . Then $(uv)_2(uv)_3(vv)(vw)_1(vw)_2$ is a path in H' . Thus $D((vw)_3) - D((uv)_2) \leq x'_{[uv]_2} + x'_{[uv]_3} + x'_{[vw]_1} + x'_{[vw]_2} \leq x^*_{[uv]_2} + x^*_{[uv]_3} + x^*_{[vw]_1} + x^*_{[vw]_2} \leq y_v^*$ where the last inequality follows from Constraint (5)¹. Thus, $Pr[D((uv)_2) \leq \alpha < D((vw)_3)] \leq y_v^*/D((tt)) \leq 2y_v^*$. Therefore

$$\begin{aligned} \mathbb{E}[\mathcal{V}] &= \sum_{v \in V(G)} r(v) \cdot Pr[v \in V_\alpha] \\ &\leq \sum_{v \in V(G)} r(v) 2y_v^* \leq 2b. \end{aligned}$$

Lastly, by Markov's inequality, $Pr[\mathcal{V} \leq (2 + \epsilon)2b] \geq 1 - 1/(2 + \epsilon)$, $Pr[\mathcal{E} \leq 4c^*] \geq 1/2$ for any $\epsilon > 0$. Then it follows there exists $0 \leq \alpha \leq D((tt))$ such that $\sum_{v \in V_\alpha} r(v) \leq 4b + 2\epsilon b$ and $c^{V_\alpha}(E_\alpha) \leq 4c^*$. One can choose ϵ such that $2\epsilon b < 1$. Since $r(v)$ is always integral, it follows that $\sum_{v \in V_\alpha} r(v) \leq 4b$, proving Lemma 8. \blacktriangleleft

It is well known that the Ball Growing algorithm (which is Dijkstra's algorithm run on H') selects a linear number of nested cuts that represent the set of all cuts at all distances between zero and $D((tt))$ from the source. It follows from Lemma 8 that one of these cuts meets the desired guarantees. Theorem 4 is then proved by simply running Algorithm 1 on the auxiliary graph H' .

References

- 1 Hassene Aissi, Da Qi Chen, and R. Ravi. Downgrading to minimize connectivity, 2019. [arXiv:1911.11229](https://arxiv.org/abs/1911.11229).
- 2 C. Burch, R. Carr, S. Krumke, M. Marathe, C. Phillips, and E. Sundberg. A decomposition-based pseudoapproximation algorithm for network flow inhibition. In Woodruff D. L., editor, *Network Interdiction and Stochastic Integer Programming*, volume 26, pages 51–68. Springer, 2003.
- 3 Stephen R Chestnut and Rico Zenklusen. Interdicting structured combinatorial optimization problems with $\{0, 1\}$ -objectives. *Mathematics of Operations Research*, 42(1):144–166, 2016.
- 4 Stephen R Chestnut and Rico Zenklusen. Hardness and approximation for network flow interdiction. *Networks*, 69(4):378–387, 2017.
- 5 Julia Chuzhoy, Yury Makarychev, Aravindan Vijayaraghavan, and Yuan Zhou. Approximation algorithms and hardness of the k-route cut problem. *ACM Transactions on Algorithms (TALG)*, 12(1):2, 2016.
- 6 Julia Chuzoy. Flows, cuts and integral routing in graphs - an approximation algorithmist's perspective. In *Proc. of the International Congress of Mathematicians*, pages 585–607, 2014.

¹ This is the main reason why we distinguish between aided and unaided arcs and contract the appropriate one to construct H' . Without the contraction, the distance between $(uv)_2$ and $(vw)_3$ includes the arc $[vw]_0$ and thus could be arbitrarily larger than y_v^* . Also, without rescaling x^* to x' , it is possible that some $D((zv)_3) > D((vw)_3)$. Then, the range in which v is downgraded can go much further past y_v^* .

- 7 Bruce Golden. A problem in network interdiction. *Naval Research Logistics Quarterly*, 25(4):711–713, 1978.
- 8 Bertrand Guenin, Jochen Könnemann, and Levent Tuncel. *A gentle introduction to optimization*. Cambridge University Press, 2014.
- 9 Guru Guruganesh, Laura Sanita, and Chaitanya Swamy. Improved region-growing and combinatorial algorithms for k-route cut problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 676–695. Society for Industrial and Applied Mathematics, 2015.
- 10 T. E. Harris and F. S. Ross. Fundamentals of a method for evaluating rail net capacities. Technical report, RAND CORP SANTA MONICA CA, Santa Monica, California, 1955.
- 11 Eitan Israeli and R Kevin Wood. Shortest-path network interdiction. *Networks: An International Journal*, 40(2):97–111, 2002.
- 12 André Linhares and Chaitanya Swamy. Improved algorithms for mst and metric-tsp interdiction. *Proceedings of 44th International Colloquium on Automata, Languages, and Programming*, 32:1–14, 2017.
- 13 Christos H Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92. IEEE, 2000.
- 14 Cynthia A. Phillips. The network inhibition problem. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 776–785, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167286.
- 15 Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.
- 16 R. Wood. Deterministic network interdiction. *Mathematical and Computer Modeling*, 17(2):1–18, 1993.
- 17 R. Zenklusen. Matching interdiction. *Discrete Applied Mathematics*, 145(15), 2010.
- 18 R. Zenklusen. Network flow interdiction on planar graphs. *Discrete Applied Mathematics*, 158(13), 2010.
- 19 R. Zenklusen. Connectivity interdiction. *Operations Research Letters*, 42(67):450–454, 2014.
- 20 R. Zenklusen. An $\mathcal{O}(1)$ approximation for minimum spanning tree interdiction. *Proceedings of 56th Annual IEEE Symposium on Foundations of Computer Science*, pages 709–728, 2015.

Sea-Rise Flooding on Massive Dynamic Terrains

Lars Arge

MADALGO, Aarhus University, Denmark
large@cs.au.dk

Mathias Rav

SCALGO, Aarhus, Denmark
mathias@scalgo.com

Morten Revsbæk

SCALGO, Aarhus, Denmark
morten@scalgo.com

Yujin Shin

MADALGO, Aarhus University, Denmark
yujinshin@cs.au.dk

Jungwoo Yang

SCALGO, Aarhus, Denmark
jungwoo@scalgo.com

Abstract

Predicting floods caused by storm surges is a crucial task. Since the rise of ocean water can create floods that extend far onto land, the flood damage can be severe. By developing efficient flood prediction algorithms that use very detailed terrain models and accurate sea-level forecasts, users can plan mitigations such as flood walls and gates to minimize the damage from storm surge flooding.

In this paper we present a data structure for predicting floods from dynamic sea-level forecast data on dynamic massive terrains. The forecast data is dynamic in the sense that new forecasts are released several times per day; the terrain is dynamic in the sense that the terrain model may be updated to plan flood mitigations.

Since accurate flood risk computations require using very detailed terrain models, and such terrain models can easily exceed the size of the main memory in a regular computer, our data structure is *I/O-efficient*, that is, it minimizes the number of *I/Os* (i.e. block transfers) between main memory and disk. For a terrain represented as a raster of N cells, it can be constructed using $O(\frac{N}{B} \log \frac{M}{B} \frac{N}{B})$ *I/Os*, it can compute the flood risk in a given small region using $O(\log_B N)$ *I/Os*, and it can handle updating the terrain elevation in a given small region using $O(\log_B^2 N)$ *I/Os*, where B is the block size and M is the capacity of main memory.

2012 ACM Subject Classification Information systems → Geographic information systems

Keywords and phrases Computational geometry, I/O-algorithms, merge tree, dynamic terrain

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.6

Funding The work in this paper was supported in part by the Danish National Research Foundation and Innovation Fond Denmark.

1 Introduction

Predicting floods caused by storm surges is a crucial task. Since the rise of ocean water can create floods that extend far onto land, the flood damage can be severe. By developing efficient flood prediction algorithms, we hope to minimize the damage from storm surge flooding by allowing users to plan mitigations such as flood walls and gates, or evacuation of affected areas.



© Lars Arge, Mathias Rav, Morten Revsbæk, Yujin Shin, and Jungwoo Yang;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 6; pp. 6:1–6:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Due to the advancement of remote sensing technology and meteorology, nowadays very detailed terrain models and accurate sea-level forecasts can be obtained, and these datasets can be used for designing accurate flood prediction algorithms. For example, the publicly available detailed raster terrain model of Denmark [13] (where each cell represents a 0.4 by 0.4 meter region) contains 267 billion cells. Furthermore, the Danish Meteorological Institute releases a sea-level forecast of the Danish territorial waters every 6 hours, containing 81 thousand values (each value corresponding to the forecasted sea-level in a 1 km² region).

Designing an efficient flood prediction algorithm is a challenging task. The algorithm must be fast enough that the computation can finish before the actual disastrous event happens or a new forecast appears, while to guarantee the accuracy of the prediction it is required to use very detailed data that is larger than the main memory in a typical computer. For example, with the terrain model and sea-level forecast datasets mentioned above, the algorithm must process the terabyte-sized terrain model and complete well within 6 hours before a new sea-level forecast is released. Existing flood prediction algorithms process the entire terrain model to compute the flood risk when a new forecast is released. Moreover, if a user modifies the terrain model to e.g. examine the effect of planned mitigations, the entire terrain model must be processed again. This is critical in practice since even a simple scan of a detailed terrain model such as the model of Denmark easily takes a few hours. However, users examine flood risk and plan flood mitigations not for the entire terrain but only for small regions of the terrain. Therefore, supporting efficient computations for a small region in the terrain would make flood prediction algorithms more practically relevant.

In this paper, we consider the problem of predicting floods from dynamic sea-level forecast data on dynamic massive terrains. The forecast data is dynamic in the sense that new forecasts can appear; the terrain is dynamic in the sense that the terrain model may be updated locally, to e.g. incorporate planned flood mitigations. We present a data structure that allows updating respectively the forecast and the terrain, and a query algorithm to report the flood height in a given query window (i.e. a small region of the terrain examined by the user). The data structure is *I/O-efficient*, meaning it can efficiently handle terrain models much larger than main memory.

The dynamic sea-level flooding problem. We use the *I/O-model* by Aggarwal and Vitter [4] to design and analyze our algorithms. In this model, the computer is equipped with a two-level memory hierarchy consisting of an internal memory and a (disk-based) external memory. The internal memory is capable of holding M data items, while the external memory is of conceptually unlimited size. All computation has to happen on data in internal memory. Data is transferred between internal and external memory in blocks of B consecutive data items. Such a transfer is referred to as an *I/O-operation* or *I/O*. The cost of an algorithm is the number of I/Os it performs.

A terrain is typically represented using a digital elevation model (DEM) as a two-dimensional array with N cells (a *raster*). Note that by storing a raster of N cells in $O(\frac{N}{B})$ tiles of size $\sqrt{B} \times \sqrt{B}$, for any $s \geq B$ a \sqrt{s} -by- \sqrt{s} square of the raster can be read or written in $O(\frac{s}{B})$ I/Os. Each cell in a raster terrain T is either a *terrain cell*, meaning that the corresponding location is on land, or an *ocean cell*. We denote the elevation of cell u in T by $h_T(u)$; the height of an ocean cell is undefined. For two cells u and v , we say that u and v are *adjacent* (or that v is a *neighbor of u*) if u and v share at least one point on their boundary. A terrain cell is a *coastal cell* if it is adjacent to an ocean cell. Since the resolution of terrain models is typically much greater than the resolution of forecasts, we partition the coastal cells into a set C of connected *coastal regions*, each region corresponding to the coastal cells

associated with a single cell of the forecast. We denote the region containing a coastal cell u by $C(u)$. A *sea-level forecast* is a function $F : C \rightarrow \mathbb{R}$ that assigns a sea-level elevation value to each coastal region. We denote the sea-level elevation of a region $R \in C$ by $F(R)$, and define a function $h_F(v) = F(C(v))$ that assigns a forecast value to each coastal cell v .

A terrain cell u is *flooded through* a coastal cell v if there exists a path $p = u \rightsquigarrow v$ of adjacent terrain cells such that each cell x in p has $h_T(x) < h_F(v)$. We call p a *flood path* of u with respect to v . Let S_u be the set of coastal cells that have flood paths to u . For a flooded cell u , we define its *flood height* as $f(u) = \max_{v \in S_u} (h_F(v) - h_T(u))$. The *flood source* of u is the region R containing the coastal cell $v \in S_u$ that has the highest sea-level value, that is, $f(u) = h_F(v) - h_T(u)$; for simplicity, we assume that the flood source is unique.

The *dynamic sea-level flooding* problem we consider in this paper consists of constructing an I/O-efficient data structure on a raster terrain T , a partition C of the coastal cells and a forecast F , and supporting the following operations I/O-efficiently, where Q_B and U is a query and an update of $\sqrt{B} \times \sqrt{B}$ cells, respectively:

- FLOOD-HEIGHT(Q_B): Return the flood height $f(u)$ of each terrain cell u in Q_B .
- FORECAST-UPDATE(F): Update the data structure with the new forecast F .
- HEIGHT-UPDATE(Q_B, U): Set the heights of terrain cells in Q_B to the values given by U .

Previous work. Previously, a large number of results on I/O-efficient algorithms have been obtained. Aggarwal and Vitter [4] showed that reading and sorting N items require $\Theta(\text{Scan}(N)) = \Theta(\frac{N}{B})$ and $\Theta(\text{Sort}(N)) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os, respectively. A set of N items can be maintained in an $O(\frac{N}{B})$ -block search tree such that updates and queries can be performed in $O(\log_B N)$ I/Os. Refer e.g. to the surveys [6, 20].

I/O-efficient algorithms for modeling flooding on terrains have been studied extensively (e.g. [2, 5, 9, 12, 7, 10, 8, 11, 14]). A number of results have also been obtained for flooding from sea-level rise. However, to our knowledge, the problem of computing flood heights while I/O-efficiently supporting updates of sea-level forecasts and the heights of terrain cells has not been studied before.

When the sea level rises *uniformly* with the same amount h_r , that is, all coastal cells belong to the same region R and $F(R) = h_r$, then it is easy to see that there is a threshold ℓ_u for each terrain cell u so that u is flooded with flood height $h_r - h_T(u)$ if and only if $h_r \geq \ell_u$. The thresholds ℓ_u for all cells u in the terrain can be computed in $O(\text{Sort}(N))$ I/Os [7], after which the flood heights in any square of B cells can be easily reported for an arbitrary h_r in $O(1)$ I/Os.

Arge et al. [11] introduced an $O(\text{Sort}(N))$ -I/O algorithm for computing flood heights when the sea-level rises non-uniformly. Their algorithm relies on the so-called *merge tree* that captures the nesting topology of depressions in T [14, 15]. Their algorithm has been incorporated into a real-time storm surge flood warning system in a pilot project between Danish Meteorological Institute (DMI) and the research spin-out company SCALGO. This system maps the extent of any flood risk resulting from the current sea-level forecast for the Danish territorial waters (updated by DMI every six hours) in full resolution on the 0.4-meter terrain model of Denmark. As part of the pilot project, the algorithm has been engineered to support efficient recomputation when a new forecast is released. However, it is unable to handle updates to the terrain without incurring $O(\text{Sort}(N))$ I/Os, which is the main motivation for the work in the present paper.

In addition to rasters, TINs are commonly used to represent terrain models. A TIN T_Δ consists of a planar triangulation of N vertices in the plane, each vertex v having an associated height $h_{T_\Delta}(v)$. The height of a point interior to a face is a linear interpolation

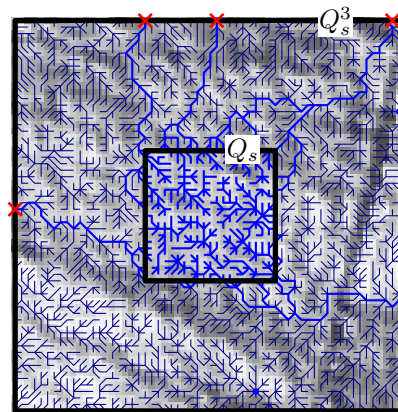
of the face vertices, so that h_{T_Δ} is a continuous piecewise linear function. Since a raster can be triangulated into a TIN, algorithms for TIN representations can be applied to raster representations as well, but the converse is not true. However, GIS applications typically implement algorithms for rasters directly, as rasters are often easier to process with simple algorithms. Furthermore, often data, such as the terrain data that we consider, is available as rasters.

To maintain dynamic terrains, Agarwal et al. [3] presented an internal-memory so-called kinetic data structure for maintaining the so-called *contour tree* of a TIN terrain T_Δ with a time-varying height function. Whereas the merge tree represents how the depressions of T are nested, the contour tree represents the nested topology of the *contours* defined by T_Δ . This result was extended to an I/O-efficient data structure by Yang [21]. He showed that for a TIN terrain with N vertices, the contour tree can be constructed in $O(\text{Sort}(N))$ I/Os and the elevation of a TIN vertex can be updated in $O(\log_B^2 N)$ I/Os.

Our results. In this paper we introduce the first data structure for the dynamic sea-level flooding problem. Our data structure can be constructed in $O(\text{Sort}(N))$ I/Os and uses $O(\frac{N}{B})$ blocks, where N is the number of cells in T . FLOOD-HEIGHT(Q_B) can be performed in $O(\log_B N)$ I/Os, FORECAST-UPDATE(F) in $O(\text{Scan}(F))$ I/Os, and HEIGHT-UPDATE(Q_B, U) in $O(\log_B^2 N)$ I/Os. Note that the number of I/Os needed to update a forecast does not depend on N , and that the terrain update bound matches the update bound of Yang [21].

Our result assumes that the size of partition set C is smaller than M (which implies that the forecast F is smaller than M), and that the number of local minima and maxima in the terrain T is also smaller than M . As the number of local minima and maxima in the terrain data for Denmark (after removing all depressions and hills with volume less than 1 m^3 , which is customary in flood computations) is 60 million, and the sea-level forecast contains 81 thousand values, both of the assumptions hold for the data for Denmark that we described above. It also requires the so-called *confluence assumption* [17] on the flow network that models how water flows on a raster terrain T . In such a network, a *flow direction* is assigned to each terrain cell u , which is a lower neighbor of u that water will flow to, and the confluence assumption intuitively says that flowing water quickly combines to larger flows at all scales. Formally, the *confluence parameter* γ is defined as follows: Let Q_s be a square of $\sqrt{s} \times \sqrt{s}$ cells and Q_s^3 be the square of $3\sqrt{s} \times 3\sqrt{s}$ cells that has Q_s in the center. Let $\gamma(Q_s)$ be the number of cells on the boundary of Q_s^3 reached from the boundary of Q_s when following flow directions without leaving Q_s^3 . Refer to Figure 1. The confluence parameter is $\gamma = \max_{s>0} \max_{Q_s} \gamma(Q_s)$ where the maximum is taken over all squares Q_s of all sizes. The *confluence assumption* then states that γ is a constant independent of the size and resolution of the terrain model.

Our work is inspired by the work of Arge et al. [11] and Yang [21]. As described previously, Arge et al. [11] compute the flood risk by using the topological features of T encoded in the merge tree. However, this structure does not support efficient terrain updates. On the other hand, Yang [21] presented an I/O-efficient data structure for maintaining the contour tree of a dynamic TIN terrain. In Section 2 we show how a raster terrain T can be transformed into a TIN T_Δ while maintaining the topology pertaining to the dynamic sea-level flooding problem. In Section 3 we then show how the merge tree of T , which is needed when computing flood risk using the approach of Arge et al. [11], can be constructed from the contour tree of T_Δ , which can be maintained under terrain updates using the data structure of Yang [21]. Note that standard techniques for triangulating a raster terrain have several issues, because they do not necessarily preserve flood paths and they do not allow the merge tree to be



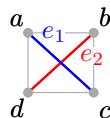
■ **Figure 1** Confluence parameter. Hillshaded terrain shown in greyscale. Blue lines show the flow directions of the terrain. The thicker, lighter blue lines are flow directions reachable from Q_s^3 . There are only 5 cells on the boundary of Q_s^3 that are reached from Q_s (red crosses), which implies that $\gamma(Q_s) = 5$ in this example.

constructed from the contour tree. Thus we believe that our transformation algorithm is of independent interest. In Section 4 we then describe the *sea-level flooding data structure* and show how the three operations are performed efficiently using the confluence assumption and the assumption on C and the number of minima and maxima.

2 Reducing raster problem to TIN

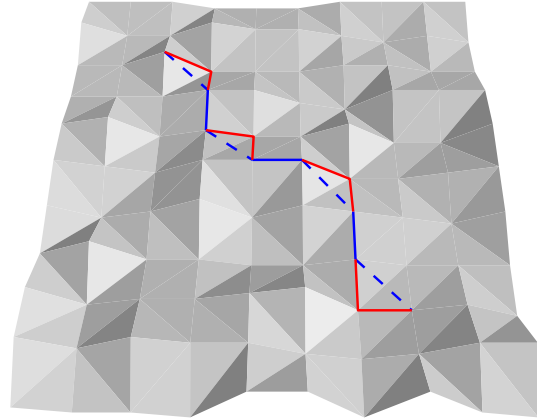
Problem definition for TINs. As mentioned, a TIN consists of a planar triangulation of a set of N vertices in the plane along with a continuous height function h_{T_Δ} that is linear on each face of the triangulation. We assume that the boundary of the triangulation is a simple polygon, with the interior corresponding to land and the exterior corresponding to ocean. A subset of the vertices are *coastal vertices*, and like for raster terrains, the coastal vertices are partitioned into a set C of connected *coastal regions*. For two vertices u, v in T_Δ , we say u and v are *adjacent* (or u is a *neighbor* of v) when there exists an edge in T_Δ that connects u and v . We define *flood path*, *flood height* and *flood source* on TINs as for raster terrains.

Transforming the raster terrain. The *incidence graph* G_T of a raster terrain T is a graph on the terrain cells of T , where two vertices u_Δ and v_Δ are connected in G_T if u and v are adjacent in T . If u and v are connected diagonally we call u_Δ a *diagonal neighbor* of v_Δ , otherwise a *cardinal neighbor* of v_Δ . Note that the natural planar embedding of G_T , where the vertex u_Δ corresponding to a cell u is placed at the center of u , is almost a triangulation, except for the intersecting edges corresponding to diagonal neighbors. We turn G_T into a



triangulation T_Δ by assigning u_Δ in T_Δ the same height as u in T , that is, $h_{T_\Delta}(u_\Delta) = h_T(u)$, and by removing a diagonal edge in T_Δ corresponding to each two-by-two square of terrain cells in T as follows: For each two-by-two square of cells a, b, c, d in clockwise order, there

5	5	5	4	4	4	5	5	5	5
6	4	4	6	5	5	4	3	4	3
4	4	6	3	3	3	4	4	3	3
4	4	4	5	5	3	2	5	5	5
5	6	3	4	3	3	5	2	2	5
5	4	3	5	5	5	2	2	3	4
4	2	3	5	2	4	3	2	1	3
3	5	2	2	1	1	2	1	1	1
5	2	3	3	4	5	1	1	2	0
5	3	5	4	4	2	1	2	4	4
2	1	1	0	5	5	0	5	1	0



■ **Figure 2** Example of flood path preservation. On the triangulated terrain T_Δ , each missing diagonal (dotted blue) is replaced by two cardinal edges (red).

are two intersecting incidence edges $e_1 = \{a, c\}$ and $e_2 = \{b, d\}$ in G_T . We triangulate the square by removing whichever edge has the higher midpoint, where the midpoint height of an edge $e = \{u, v\}$ is $s(e) = \frac{1}{2}(h_T(u) + h_T(v))$. If $s(e_1) = s(e_2)$, then we pick an arbitrary edge to remove.

► **Theorem 1.** u is flooded through v in T if and only if u_Δ is flooded through v_Δ in T_Δ .

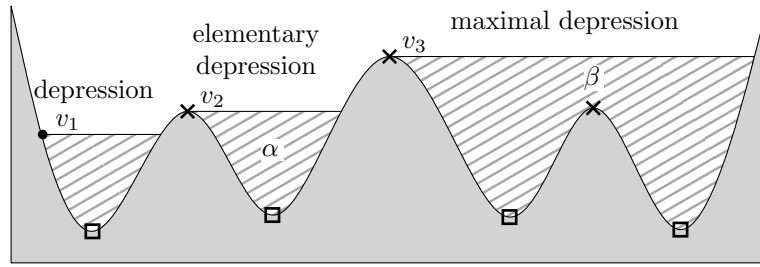
Proof. First, if u_Δ is flooded through v_Δ in T_Δ , that is, there exists a flood path $p_\Delta : u_\Delta \rightsquigarrow v_\Delta$, then there is a flood path $p : u \rightsquigarrow v$ in T corresponding to p_Δ , since each edge in T_Δ has a corresponding edge in G_T . Thus u is flooded through v in T .

Next, we show that if $p : u \rightsquigarrow v$ is a flood path in T , then there is a corresponding flood path $p_\Delta : u_\Delta \rightsquigarrow v_\Delta$ in T_Δ . If no edge in G_T corresponding to adjacent cells in p was removed by the TIN construction, then we are done; p_Δ is the sequence of vertices that correspond to the cells in p . Otherwise, we show how to obtain p_Δ by replacing each edge e in G_T corresponding to adjacent cells in p that is not in T_Δ as follows: Since the TIN construction only removes diagonal edges, e is a diagonal edge connecting two raster cells a and c . Recall that the definition of flood path means that $\max\{h_T(a), h_T(c)\} \leq h_F(u)$. Let e' be the diagonal edge connecting cells b and d such that e and e' intersect and e' is included in T_Δ . Then $s(e') \leq s(e)$, which implies that $\min\{h_T(b), h_T(d)\} \leq \max\{h_T(a), h_T(c)\} \leq h_F(u)$. Without loss of generality assume $h_T(b) \leq h_T(d)$, in which case we replace the edge e with the two edges $\{a, b\}$ and $\{b, c\}$. These edges are both non-diagonal edges and thus were not discarded when we triangulated T into T_Δ , that is, we replace the adjacent cells a and c in p with cells abc . Refer to Figure 2 for an example. Since $h_T(b) \leq h_F(u)$, p is still a flood path after adding b . After handling all relevant edges this way, we have obtained a flood path p such that p_Δ is the sequence of vertices that correspond to the cells in p . ◀

3 Connecting topology of T and T_Δ

3.1 Local topology and depressions

Flow directions, sinks, peaks, upper and lower sequences, and depressions. As mentioned previously, water flow on a raster terrain T can be modeled by assigning a *flow direction* on each terrain cell u in T , which is a lower neighbor of u that water will flow to from u .



■ **Figure 3** An example terrain seen from the side, showing cells v_1 - v_3 along with depressions defined by the cells. The cell v_2 defines an elementary depression α . The cell v_3 defines a maximal, but not elementary, depression β .

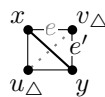
When a cell u does not have any lower neighbors, then u is not assigned a flow direction and we call it a *sink*. Similarly, a cell u is a *peak* if there is no neighbor of u that has higher elevation than u . We assume that no pair of adjacent cells have the same height. That is, a flow direction can be assigned to all terrain cells except sinks. The assumption can be removed using standard techniques [7].

As we traverse the neighbors of a cell u in T that is not a sink or peak in clockwise order, there are sequences of cells that are lower or higher than u . Each continuous sequence of lower (resp. higher) neighbors of u is called a *lower sequence* (resp. *upper sequence*) of u .

A raster terrain cell u defines a *depression* that is the maximal connected component of terrain cells containing u such that all cells v in the depression have $h(v) \leq h(u)$ [8]. Note that each depression contains at least one sink. A depression β_1 is *maximal* if every depression $\beta_2 \supset \beta_1$ contains strictly more sinks than β_1 . If a maximal depression β contains exactly one sink, then we call β an *elementary depression*. Refer to Figure 3.

TIN construction preserves lower sequences, sinks, and depressions. On TINs, *flow direction*, *sink*, *peak*, *lower/upper sequence*, and (*maximal/elementary*) *depression* are defined as for rasters, but with the vertex adjacency defined by edges of the triangulation T_Δ rather than the incidence graph G_T . The following lemmas show that lower sequences, sinks, and depressions in T are preserved by our raster to TIN transformation. Note that the lemmas say nothing about peaks or upper sequences, as it is easy to verify that they are not preserved by the transformation. Refer to Figure 4 for an example.

► **Lemma 2.** *For any lower sequence J of any cell u in T , u has a neighbor v in J such that u_Δ and v_Δ are connected in T_Δ .*

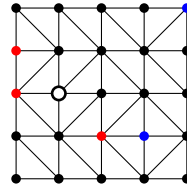


Proof. Pick any $v \in J$. If the edge $e = (u_\Delta, v_\Delta)$ is in T_Δ , then we are done. Otherwise, e is removed by our TIN construction, so v must be a diagonal neighbor of u in T . Let $e' = (x, y)$ be the edge that was chosen to remain instead of e in the TIN construction. Then we have that x and y are the common neighbors of u and v and $s(e') \leq s(e)$, which implies that $\min\{h_T(x), h_T(y)\} \leq h_T(u)$. Without loss of generality assume $h_T(x) \leq h_T(y)$. Then x is a lower neighbor of u in J , and since x is a cardinal neighbor of u , T_Δ contains $\{u_\Delta, x_\Delta\}$. ◀

► **Corollary 3.** *A cell u has a lower neighbor in T if and only if u_Δ has a lower neighbor in T_Δ .*

20	21	22	24	25
16	7	5	1	23
15	14	13	2	19
11	12	17	18	3
10	9	8	6	4

(a) Raster terrain.



(b) Triangulated terrain.

■ **Figure 4** Example showing that upper sequences and peaks are not necessarily preserved by our TIN construction. In (a) the raster terrain cell with height 14 has two upper sequences (red), but in the TIN terrain, the corresponding vertex (marked with a circle in (b)) has only one upper sequence. The raster terrain has only a single peak, the cell with height 25 (blue), but in the TIN terrain the vertex corresponding to the cell with height 18 is also a peak (blue).

► **Lemma 4.** *For any cell u in T , the cells in the depression β defined by u in T are in one-to-one correspondence with the vertices in the depression β_Δ defined by u_Δ in T_Δ .*

Proof. First, we show that for each $v_\Delta \in \beta_\Delta$, v is in β : As $v_\Delta \in \beta_\Delta$, there is a path $p : u_\Delta \rightsquigarrow v_\Delta$ in T_Δ of vertices with height below $h_{T_\Delta}(u)$. This path corresponds to a path in T (since the TIN construction only removes edges) and therefore $v \in \beta$.

To show that the cells in β correspond to a subset of the vertices in β_Δ , it suffices to show that the vertices corresponding to cells in β are connected in T_Δ , since β_Δ is a maximal connected component of vertices with height $\leq h_{T_\Delta}(u_\Delta)$ (and each vertex in T_Δ has the same height as it has in T). To show this we proceed by induction in the list of cells u in T ordered by height $h_T(u)$. Suppose that for any cell u' in T with $h_T(u') < h_T(u)$, the depression defined by u' is connected in T_Δ . If u is a sink, then $\beta = \{u\}$ which is trivially connected in T_Δ . Otherwise, we consider the set $L = \beta \setminus \{u\}$ and make the following two observations for each connected component β' of L .

- Since β' is a depression in T defined by the highest cell in β' , it follows by induction that β' is connected in T_Δ .
- It is easy to see that β' contains a lower neighbor v of u and thus contains all cells in the lower sequence J containing v ; by Lemma 2, it follows that u_Δ is adjacent in T_Δ to a vertex that corresponds to a cell in J .

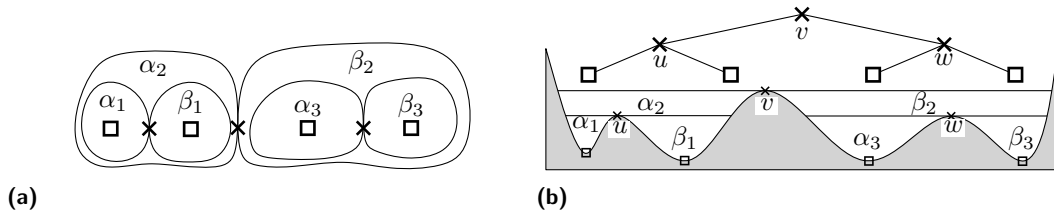
From this it follows that u_Δ is connected to all of β' in T_Δ , so β is connected in T_Δ . ◀

► **Corollary 5.** *A depression β is a maximal (resp. elementary) depression in T if and only if β_Δ is a maximal (resp. elementary) depression in T_Δ .*

3.2 Merge trees and contour trees

Merge tree of raster T . As mentioned, the merge tree \mathcal{M} of a raster terrain T is a rooted tree that represents the nested topology of the maximal depressions [14]. Each node in \mathcal{M} represents a maximal depression in T , and we refer to the maximal depression represented by a merge tree node x as β_x . Each elementary depression is represented by a leaf node, and a node y is the parent of a node x when $\beta_x \subset \beta_y$ and there exists no maximal depression β_z such that $\beta_x \subset \beta_z \subset \beta_y$.

Now consider sweeping the raster terrain with a plane of height ℓ from $-\infty$ to ∞ while maintaining the set of depressions that consist of cells with elevation less than or equal to ℓ . If the number of depressions decreases when the sweeping plane crosses a cell u , then u is called a *negative saddle*; it is easy to see that a negative saddle u has at least two lower



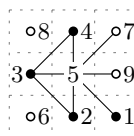
■ **Figure 5** An example terrain with negative saddle cells u, v and w . (a) Terrain seen from above. Sinks are marked with a square and saddles are marked with a cross. The maximal depressions $\alpha_1, \beta_1, \alpha_3,$ and β_3 are elementary. (b) Terrain seen from the side along with the merge tree \mathcal{M} .

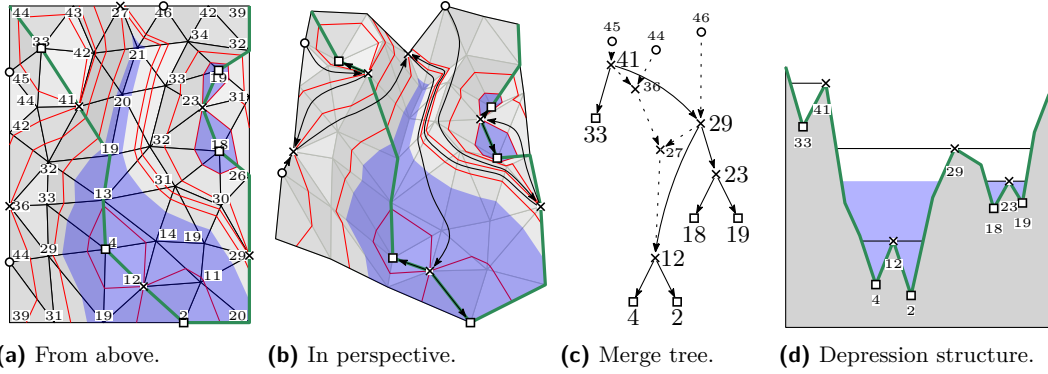
sequences. Then note that for any two maximal depressions β_x and β_y whose corresponding nodes x and y in \mathcal{M} share the same parent, there exists a negative saddle where β_x and β_y are merged. Using this we associate a terrain cell to each merge tree node as follows: To a leaf node x we associate the sink in the elementary depression β_x ; to an internal node x we associate the negative saddle where the maximal depressions of its children are merged. Refer to Figure 5.

Contour tree of TIN T_Δ . For $\ell \in \mathbb{R}$, the ℓ -level set of T_Δ is defined to consist of points $x \in \mathbb{R}^2$ with $h_{T_\Delta}(x) = \ell$. A *contour* of T_Δ is a connected component of a level set of T_Δ [3]. We define a *down-contour* of u_Δ as any contour with elevation $h_{T_\Delta}(u) - \epsilon$, for a value ϵ smaller than the height difference between any pair of vertices, such that the contour intersects an edge incident to u_Δ [3]. Similarly, we define an *up-contour* as a contour with elevation $h_{T_\Delta}(u) + \epsilon$ that intersects an edge incident to u_Δ .

Traversing the neighbors of a vertex u_Δ in clockwise order, we say that u_Δ is a *saddle* in T_Δ if there are multiple sequences of lower neighbors of u_Δ disconnected by higher neighbors of u_Δ [3]. For simplicity we assume that every saddle has exactly two such sequences of lower neighbors. This assumption can be removed [16]. We say that a vertex u_Δ is *critical* if u_Δ is a peak, sink, or saddle. If a saddle u_Δ has one up-contour (down-contour) and two down-contours (up-contours) then u_Δ is called a *negative (positive) saddle*. For any two contours C_1 and C_2 with level ℓ_1 and ℓ_2 , respectively, we say C_1 and C_2 are *equivalent* if they belong to the same connected component of $\Gamma = \{x \in \mathbb{R}^2 \mid \ell_1 \leq h_{T_\Delta}(x) \leq \ell_2\}$ that does not contain any critical vertex. When sweeping T_Δ with a plane from $-\infty$ to ∞ , an equivalence class of contours starts and ends at critical vertices. That is, the contours deform continuously as the sweeping plane changes its height, but the number of contours does not change as long as the plane varies between two critical vertices. A contour appears and disappears at a sink and a peak, respectively. Two contours merge into one at a negative saddle, and a contour splits into two at a positive saddle.

The *contour tree* \mathcal{A} of a TIN T_Δ is a tree on the critical vertices in T_Δ that encodes the topological changes of the contours [3, 21]. Two critical vertices u_Δ, v_Δ are connected in \mathcal{A} if and only if an equivalence class of contours starts at u_Δ and ends at v_Δ . That is, an edge (u_Δ, v_Δ) in \mathcal{A} represents the equivalence class of contour that appears at u_Δ and disappears at v_Δ . Refer to Figure 6. Note that the contour tree is not a rooted tree; an internal node has two lower (higher) neighbors and one higher (lower) neighbor if it corresponds to a negative (positive) saddle.

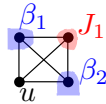




■ **Figure 6** Example TIN terrain. Sinks are marked with squares, peaks with circles, and saddles with crosses. (a, b) Everything below $\ell = 23$ is marked as blue. Contours defined by saddle vertices are marked with red lines. (b) Edges of the contour tree \mathcal{A} are shown as arcs pointing downwards in height. (c) Merge tree derived from the contour tree (dashed), representing how the maximal depressions in (d) are nested.

TIN construction preserves negative saddles. Note that for raster terrains we have only defined negative saddles, and not *saddles* or *positive saddles*. The reason is, as shown in the example in Figure 6, that a cell in T with multiple lower and upper sequences can become a regular vertex in T_Δ after our transformation. However, we can show that negative saddles are preserved by the transformation. To do so we need the following lemma.

► **Lemma 6.** *For any negative saddle u in a raster terrain T that merges two depressions β_1 and β_2 , let J_1 and J_2 be the two higher sequences of u that separate β_1 and β_2 in a clockwise traversal of the neighbors of u . Then J_1 and J_2 each contain a cardinal neighbor of u .*



Proof. Assume for contradiction that a higher sequence separating β_1 and β_2 does not contain a cardinal neighbor of u , that is, that it consists of a single diagonal neighbor. This implies that a cell in β_1 has a diagonal neighbor in β_2 , which violates the definition of depressions as maximal connected components. ◀

Using Lemma 6 we can prove the following.

► **Lemma 7.** *A cell u is a negative saddle in T if and only if u_Δ is a negative saddle in T_Δ .*

Proof. First, we show that if u is a negative saddle in T , then u_Δ is a negative saddle in T_Δ . Let β_1 and β_2 be the two depressions that merge at u . By Lemma 2, u_Δ is connected in T_Δ to a lower neighbor u^1_Δ in β_1 and a lower neighbor u^2_Δ in β_2 . By Lemma 6, u has two higher cardinal neighbors v^1 and v^2 separating u^1 and u^2 in a clockwise traversal of the neighbors of u . Since the construction of T_Δ only removes diagonal edges, u_Δ is connected to both v^1_Δ and v^2_Δ in T_Δ . Thus u_Δ has four alternating lower and higher neighbors $u^1_\Delta, v^1_\Delta, u^2_\Delta, v^2_\Delta$ in clockwise order, so u_Δ is a saddle in T_Δ . By Lemma 4, β_1 and β_2 are preserved in T_Δ . Thus the down-contour of u_Δ intersecting u^1_Δ is distinct from the down-contour intersecting u^2_Δ , so u_Δ is a negative saddle.

Next, we show that if u is not a negative saddle in T , then u_Δ is not a negative saddle in T_Δ . If the number of lower sequences of u is less than 2, then it is easy to see that u_Δ is not a saddle in T_Δ . If the number of lower sequences of u is at least 2, then these lower sequences must be from the same depression β . By Lemma 4, β is connected in T_Δ and thus u_Δ is not a negative saddle. \blacktriangleleft

Constructing merge tree \mathcal{M} of T from contour tree \mathcal{A} of T_Δ . We now show how the merge tree \mathcal{M} of T can be constructed from the contour tree \mathcal{A} of the TIN T_Δ obtained after applying our TIN construction to T . From Corollary 3 and Lemma 7 it follows that the nodes of \mathcal{M} , which correspond to the sinks and negative saddles of T , are encoded in \mathcal{A} . In Appendix A.1 we show (Lemma 12) that if v is the highest node on the path between u and v in \mathcal{A} , then there is a path between u and v in T_Δ where v is the highest vertex. The following lemma is then the key to constructing the edges of \mathcal{M} from \mathcal{A} .

► **Lemma 8.** *For two distinct nodes u_Δ and v_Δ in the contour tree \mathcal{A} of T_Δ that correspond to negative saddles or sinks, $\beta_{u_\Delta} \subset \beta_{v_\Delta}$ if and only if there is a path $p_{\mathcal{A}} : u_\Delta \rightsquigarrow v_\Delta$ in \mathcal{A} such that all vertices in $p_{\mathcal{A}}$ have height less than or equal to the height of v_Δ .*

Proof. Suppose $\beta_{u_\Delta} \subset \beta_{v_\Delta}$. Then there is a path $p_\Delta : u_\Delta \rightsquigarrow v_\Delta$ in T_Δ such that all vertices in p_Δ have height less than or equal to $h_{T_\Delta}(v_\Delta)$. The edges and vertices in \mathcal{A} corresponding to all contours through points on p_Δ in T_Δ form a connected subtree of \mathcal{A} . Thus there is a path $p_{\mathcal{A}} : u_\Delta \rightsquigarrow v_\Delta$ in \mathcal{A} with height less than or equal to the height of v_Δ .

Now, suppose $p_{\mathcal{A}}$ is a path in \mathcal{A} from u_Δ to v_Δ such that v_Δ is the highest vertex in $p_{\mathcal{A}}$. By Lemma 12 there is a path $p_\Delta : v_\Delta \rightsquigarrow u_\Delta$ in T_Δ such that the highest vertex on p_Δ is v_Δ . This implies that u_Δ is contained in the depression defined by v_Δ , so $\beta_{u_\Delta} \subseteq \beta_{v_\Delta}$. \blacktriangleleft

► **Theorem 9.** *The merge tree \mathcal{M} of T can be constructed from the contour tree \mathcal{A} of T_Δ .*

Proof. For each negative saddle u_Δ in \mathcal{A} we define a *key descendant* of u_Δ for each child v_Δ of u_Δ as follows: If v_Δ is a sink or a negative saddle, then v_Δ is the key descendant of u_Δ . Otherwise, it is a positive saddle, and the key descendant is found by following a downward path from v_Δ in \mathcal{A} until reaching a negative saddle or sink; since a vertex in \mathcal{A} that is not a negative saddle or sink has exactly one lower neighbor, this downward path following lower neighbors until encountering a negative saddle or sink is unique. Since u_Δ has exactly two children, it has two key descendants.

To construct \mathcal{M} we start with a forest containing all the sinks of \mathcal{A} (leaves of \mathcal{M}), and we maintain a union-find data structure that maps each vertex u of \mathcal{M} to the root of u in the forest \mathcal{M} constructed so far. Next, we insert the negative saddles of \mathcal{A} (internal nodes) into \mathcal{M} in increasing order of height using the union-find data structure as follows: When processing a negative saddle u_Δ with key descendants v_Δ^1 and v_Δ^2 , we query the union-find data structure to obtain $v^1 = \text{FIND}(v_\Delta^1)$ and $v^2 = \text{FIND}(v_\Delta^2)$. From Lemma 8, v^1 and v^2 are the children of u in \mathcal{M} , so we insert u into \mathcal{M} with children v^1 and v^2 . We then update the union-find structure using $\text{UNION}(v^1, v^2)$. When we have processed all negative saddles in this way, we have constructed \mathcal{M} . \blacktriangleleft

4 Sea-level flooding data structure

We are now ready to describe our sea-level flooding data structure. Intuitively, our structure maintains the result of a flood computation for a forecast F on a terrain T by a *flood instance* I_F , which stores for each sink cell u of T the flood source of u . When answering a query

the flood height of any cell can then be computed from I_F using the observation that if a non-sink cell v is flooded by F , then the flood source of v is the same as the flood source of any sink cell u in the depression defined by v , and it is easy to compute the flood height of v from such a sink u [11]. Furthermore, we maintain the contour tree \mathcal{A} of T_Δ using the data structure of Yang [21]. As required by this data structure, our data structure also maintains a so-called *descent tree* Π_\downarrow and a so-called *ascent tree* Π_\uparrow that store descending and ascending connectivity on T_Δ , respectively. More precisely, the descent (ascent) tree is an I/O-efficient data structure that maintains a forest on the vertices in T_Δ , where each vertex is connected to one of its lower (upper) neighbors in T_Δ , and where each root in the forest corresponds to a sink (peak). The descent (ascent) tree Π_\downarrow (Π_\uparrow) supports finding for a cell u , a sink (peak) that can be reached from u by a decreasing (increasing) path in $O(\log_B N)$ I/Os. Thus, Π_\downarrow can be queried to find a sink in the depression defined by v in $O(\log_B N)$ I/Os. The ascent and descent trees also support the following operations in $O(\log_B^2 N)$ I/Os: Disconnect the subtree rooted at u from its parent, and link a root u to a vertex v . For details we refer to [21].

While Yang described how to maintain \mathcal{A} , Π_\downarrow and Π_\uparrow when updating the height of a single vertex in $O(\log_B^2 N)$ I/Os [21], we need to update heights in a square Q_B of B cells in the same bound. Yang classifies the possible changes to \mathcal{A} as a result of changing the height of a vertex of T_Δ as either adding a sink or peak (*birth event*), removing a sink or peak (*death event*), or reordering saddles (*interchange event*). When updating a number of vertices in a region Q_B of \sqrt{B} by \sqrt{B} cells, birth and death events are conceptually simple to handle, as the involved sink or peak is either in Q_B or adjacent to a vertex in Q_B . On the other hand, an interchange event involves a saddle u in Q_B and a saddle v adjacent to u in \mathcal{A} , but not necessarily in the vicinity of Q_B in T_Δ . Yang [21] handles such an interchange event by querying Π_\downarrow and Π_\uparrow with the neighbors of u and v in T_Δ . However, this is a problem for our data structure, as updating the heights of B vertices in Q_B can cause $\Theta(B)$ interchange events and thus $\Theta(B)$ queries to Π_\downarrow and Π_\uparrow , which would require $\Theta(B \log_B N)$ I/Os. In Appendix A.2 we describe (Lemma 13) how to answer the queries to Π_\downarrow and Π_\uparrow without I/Os, by maintaining in addition a set L of vertices of Π_\downarrow and Π_\uparrow in main memory. In this way, \mathcal{A} can be updated without using I/Os (since we have assumed that the critical vertices, and thus \mathcal{A} , fits in memory).

Yang also showed how to augment his data structure such that all vertices in T_Δ are represented in \mathcal{A} [21]. Intuitively, a vertex v is added to the edge representing the contour through v . The augmented data structure can be maintained in the same bounds as described above [21]. In our sea-level flooding data structure, we similarly augment the contour tree of T_Δ with the so-called *coastal minima* of T_Δ that are the coastal vertices u with no lower neighbors inside their coastal region, that is, a coastal vertex u is a coastal minimum if there is no vertex $v \in C(u)$ adjacent to u that is lower than u . We denote by \mathcal{A}^+ this augmented contour tree extended to represent the coastal minima. In Appendix A.3 (Lemma 14) we show that \mathcal{A}^+ contains enough information to determine which sinks are flooded by a forecast F . More precisely, we show that if a coastal vertex u in coastal region R floods a terrain vertex v , then there is a coastal minimum w in R that also floods v . Furthermore, we show that the number of coastal minima is bounded by the number of coastal regions $|R|$ and the number of critical vertices in T_Δ .

In summary, our sea-level flooding data structure consists of the following components:

- Contour tree \mathcal{A}^+ containing the sinks, peaks, saddles and coastal minima of T_Δ ;
- Descent tree Π_\downarrow and ascent tree Π_\uparrow on T_Δ , as well as a set L of vertices of Π_\downarrow and Π_\uparrow ;
- Terrain T , forecast F and flood instance I_F .

Space. Recall that we assume that the number of sinks, peaks and saddles in T_Δ , as well as the number of coastal regions, is smaller than M . Thus \mathcal{A}^+ and L fit in main memory. As I_F stores a coastal region for each sink of T_Δ , it also fits in main memory. By assumption, F fits in main memory. Finally, T and the descent and ascent trees are stored in external memory where they use $O(N/B)$ blocks of space [21]. It is easy to see that our data structure can be constructed in $O(\text{Sort}(N))$ I/Os [21].

Flood-Height(Q_B) query. To compute the flood height for all cells in Q_B , we first associate each cell $v \in Q_B$ with a sink in the depression defined by v as follows: Let Q_B^3 be the square of $3\sqrt{B} \times 3\sqrt{B}$ cells that has Q_B in the center. For each $v \in Q_B$, we follow flow directions from v until reaching either the boundary of Q_B^3 or a sink u . By assumption, the number of times we reach a cell on the boundary of Q_B^3 is constant. For each such cell w , we query Π_\downarrow to find a sink u in the depression defined by w , and we associate u with the cell $v \in Q_B$ that reached w when following flow directions. After this way having associated each $v \in Q_B$ with a sink u in the depression defined by v , the flood source of each $v \in Q_B$ can be found using I_F as discussed above.

Following flow directions can be done by loading the query cells of T in Q_B^3 into main memory and computing the flow directions in $O(1)$ I/Os. Making the $O(1)$ queries to Π_\downarrow requires $O(\log_B N)$ I/Os [21]. Then, the flood sources can be found using I_F without using any I/Os, since I_F is stored in memory. Thus, FLOOD-HEIGHT(Q_B) requires $O(\log_B N)$ I/Os in total.

Forecast-Update(F). To update the flood instance I_F given a new forecast F , we first construct \mathcal{M} from \mathcal{A}^+ using Theorem 9. Then for each coastal minimum v in a coastal region R , we compute $h = h_F(v) - h_T(v)$. If $h > 0$, it means that v is flooded by the forecast value $F(R) = h_F(v)$. Note that this also means that all sinks in the depression defined by v are flooded. As \mathcal{A}^+ contains the coastal minimum v , we can then use \mathcal{A}^+ to find one of these sinks u by following a decreasing path in \mathcal{A}^+ from v until reaching a sink as follows: At a negative saddle w , we pick an arbitrary lower neighbor of w , and at a vertex w that is not a negative saddle or sink (i.e. a peak, positive saddle, or coastal minimum), w has a unique lower neighbor in \mathcal{A}^+ . Eventually we reach a sink u and since we have followed a strictly decreasing path from v to u in \mathcal{A}^+ it follows by Lemma 8 that u is in the depression defined by v . Note that u is also a node in \mathcal{M} , and that in any instance of the sea-level flooding problem in which u is flooded with forecast value $F(R)$, the other sinks in the depression defined by v will also be flooded. After having found at most one flooded sink for each coastal minimum in this way, we can identify the remaining flooded sinks in the terrain using the algorithm by Arge et al. [11] that takes as input a list of forecast values for sinks in a merge tree \mathcal{M} of T and computes the flood source and flood height for all sinks in \mathcal{M} . This in turn gives us the new flood instance I_F .

After the forecast F is read into memory using $O(\text{Scan}(F))$ I/Os, no further I/Os are required for FORECAST-UPDATE, since \mathcal{A}^+ , \mathcal{M} and I_F fit in main memory.

Height-Update(Q_B, U). To update the heights of the cells in Q_B we need to update Π_\downarrow , Π_\uparrow , \mathcal{A}^+ and I_F .

Intuitively, we update Π_\downarrow and Π_\uparrow by removing subtrees containing the vertices whose heights were updated, reconstructing a new forest corresponding to the new descending or ascending connectivity after the update, and then linking the forest into the structure. More precisely, let Q_B^3 and Q_B^5 be the squares of $3\sqrt{B} \times 3\sqrt{B}$ and $5\sqrt{B} \times 5\sqrt{B}$ cells, respectively,

such that Q_B^3 and Q_B^5 both have Q_B in the center. We describe how the descent tree Π_\downarrow is updated; the ascent tree can be updated in an analogous way. First, we will disconnect a number of edges on the boundary of Q_B^3 to ensure that no tree in Π_\downarrow contains both a vertex inside Q_B and a vertex outside Q_B^5 . In other words, we isolate a set of vertices V in Π_\downarrow such that $Q_B \subset V \subset Q_B^5$, as follows: First we disconnect edges so that no vertex in Q_B is connected to a vertex on the boundary of Q_B^3 by following edges of Π_\downarrow from each vertex in Q_B until reaching either a sink inside Q_B^3 or a vertex on the boundary of Q_B^3 . By the confluence assumption, the number of times we reach a vertex on the boundary of Q_B^3 is constant. For each such vertex w , we disconnect w from its parent in Π_\downarrow . Next, we disconnect edges so that no vertex outside Q_B^5 is connected to a vertex on the boundary of Q_B^3 by following edges of Π_\downarrow from all cells on the boundary of Q_B^5 towards Q_B^3 until we reach a sink or the boundary of Q_B^3 . It is easy to show that the number of times we reach the boundary of Q_B^3 is a constant, by covering each of the four sides of Q_B^5 with five translated copies of Q_B , and bounding the number of times each copy can reach the boundary of Q_B^3 using the confluence parameter. As previously, we disconnect each vertex w reached on the boundary of Q_B^3 from its parent in Π_\downarrow . Let u in Q_B and v outside Q_B^5 be vertices that were in the same tree of Π_\downarrow before we disconnected edges on the boundary of Q_B^3 , and let w be the lowest common ancestor of u and v in Π_\downarrow at that time. If w is outside Q_B^3 , then u is no longer connected to w as we disconnected the first edge on the path from u to w that is on the boundary of Q_B^3 . If w is inside Q_B^3 , a similar argument shows that v is no longer connected to w . Thus we have isolated a set V of $O(B)$ vertices in Π_\downarrow such that $Q_B \subseteq V \subseteq Q_B^5$. We can then simply reconstruct new subtrees for V in Π_\downarrow according to the new descending connectivity resulting from the height updates in Q_B and link back the disconnected edges on the boundary of Q_B^3 into Π_\downarrow .

After updating $\Pi_\downarrow/\Pi_\uparrow$ we update \mathcal{A}^+ with the new topology of the terrain using the update algorithm in [21], and we use FORECAST-UPDATE(F) to update the flood instance I_F .

As we disconnect and reconnect $O(1)$ edges of Π_\downarrow and Π_\uparrow , updating the heights of cells in Q_B can be done using $O(\log_B^2 N)$ I/Os [21]. Updating \mathcal{A}^+ requires querying Π_\downarrow and Π_\uparrow with neighbors of saddle vertices; as discussed previously (Lemma 13), this part can be handled without I/Os since \mathcal{A}^+ and L fit in memory, where L contains the vertices of Π_\downarrow and Π_\uparrow required to perform the update of \mathcal{A}^+ . Since F is already in main memory, FORECAST-UPDATE does not require any I/Os. Thus, HEIGHT-UPDATE(Q_B, U) requires $O(\log_B^2 N)$ I/Os in total.

► **Theorem 10.** *Given a terrain of N cells, a partition of the coastal cells of the terrain into a set of coastal regions C , and a forecast $F : C \rightarrow \mathbb{R}$, for which the following assumptions hold:*

- *the confluence parameter γ is constant,*
- *the number of local minima and maxima is smaller than M ,*
- *$|C|$ is smaller than M ,*

a data structure for the dynamic sea-level flooding problem can be constructed in $O(\text{Sort}(N))$ I/Os using $O(\frac{N}{B})$ blocks of space, such that

- *FLOOD-HEIGHT(Q_B) can be performed in $O(\log_B N)$ I/Os,*
- *FORECAST-UPDATE(F) can be performed in $O(\text{Scan}(F))$ I/Os, and*
- *HEIGHT-UPDATE(Q_B, U) can be performed in $O(\log_B^2 N)$ I/Os.*

References

- 1 Pankaj K. Agarwal, Lars Arge, Gerth Stølting Brodal, and Jeffrey S. Vitter. I/O-efficient Dynamic Point Location in Monotone Planar Subdivisions. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 11–20, 1999.

- 2 Pankaj K Agarwal, Lars Arge, and Ke Yi. I/O-efficient batched union-find and its applications to terrain analysis. *ACM Trans. Algorithms*, 7(1):11, 2010.
- 3 Pankaj K. Agarwal, Thomas Mølhave, Morten Revsbæk, Issam Safa, Yusu Wang, and Jungwoo Yang. Maintaining Contour Trees of Dynamic Terrains. In *31st International Symposium on Computational Geometry*, volume 34, pages 796–811, 2015.
- 4 Alok Aggarwal and Jeffrey Vitter. The Input/output Complexity of Sorting and Related Problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
- 5 Cici Alexander, Lars Arge, Peder Klith Bøcher, Morten Revsbæk, Brody Sandel, Jens-Christian Svenning, Constantinos Tsirogiannis, and Jungwoo Yang. Computing River Floods Using Massive Terrain Data. In *Geographic Information Science*, pages 3–17. Springer International Publishing, 2016.
- 6 Lars Arge. External memory data structures. In *Handbook of massive data sets*, pages 313–357. 2002.
- 7 Lars Arge, Jeffrey S Chase, Patrick Halpin, Laura Toma, Jeffrey S Vitter, Dean Urban, and Rajiv Wickremesinghe. Efficient Flow Computation on Massive Grid Terrain Datasets. *GeoInformatica*, 7(4):283–313, 2003.
- 8 Lars Arge, Mathias Rav, Sarfraz Raza, and Morten Revsbæk. I/O-Efficient Event Based Depression Flood Risk. In *Proc. 9th Workshop on Algorithm Engineering and Experiments*, pages 259–269. SIAM, 2017.
- 9 Lars Arge and Morten Revsbæk. I/O-efficient contour tree simplification. In *International Symposium on Algorithms and Computation*, pages 1155–1165. Springer, 2009.
- 10 Lars Arge, Morten Revsbæk, and Norbert Zeh. I/O-efficient computation of water flow across a terrain. In *Proceedings of the 26th annual Symposium on Computational Geometry*, pages 403–412. ACM, 2010.
- 11 Lars Arge, Yujin Shin, and Constantinos Tsirogiannis. Computing Floods Caused by Non-Uniform Sea-Level Rise. In *Proc. 20th Workshop on Algorithm Engineering and Experiments*, pages 97–108. SIAM, 2018.
- 12 Lars Arge, Laura Toma, and Jeffrey Scott Vitter. I/O-efficient Algorithms for Problems on Grid-based Terrains. *Journal of Experimental Algorithmics*, 6:1, 2001.
- 13 Danish Geodata Agency. Elevation Model of Denmark:Terræn (0.4 meter grid). <http://eng.gst.dk>, 2015.
- 14 Andrew Danner. *I/O Efficient Algorithms and Applications in Geographic Information Systems*. PhD thesis, Department of Computer Science, Duke University, 2006.
- 15 Andrew Danner, Thomas Mølhave, Ke Yi, Pankaj K Agarwal, Lars Arge, and Helena Mitásová. TerraStream: from Elevation Data to Watershed Hierarchies. In *Proc. 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, page 28. ACM, 2007.
- 16 Herbert Edelsbrunner, John Harer, and Afra Zomorodian. Hierarchical morse-smale complexes for piecewise linear 2-manifolds. *Discrete and computational Geometry*, 30(1):87–107, 2003.
- 17 Herman J. Haverkort and Jeffrey Janssen. Simple I/O-efficient Flow Accumulation on Grid Terrains. *CoRR*, abs/1211.1857, 2012. URL: <http://arxiv.org/abs/1211.1857>.
- 18 Robert E Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.
- 19 Robert Endre Tarjan and Uzi Vishkin. Finding biconnected components and computing tree functions in logarithmic parallel time. In *25th Annual Symposium on Foundations of Computer Science*, pages 12–20. IEEE, 1984.
- 20 Jeffrey Scott Vitter. Algorithms and data structures for external memory. *Foundations and Trends® in Theoretical Computer Science*, 2(4):305–474, 2008.
- 21 Jungwoo Yang. *Efficient Algorithms for Handling Massive Terrains*. PhD thesis, Department of Computer Science, University of Aarhus, 2015.

A Appendices

A.1 Connecting paths in T_Δ and \mathcal{A}

In this section we show an important relation between paths along the edges of a TIN T_Δ and paths along the edges of the contour tree \mathcal{A} of T_Δ . Recall that we have defined a *flood path* from a coastal vertex u to a vertex v as a path $p : u \rightsquigarrow v$ along the edges of T_Δ such that no vertex in v has height greater than $h_F(u)$. Thus, one way of showing that flood paths are preserved is by showing the more general statement that paths that stay below some height level ℓ are preserved. The following lemmas show that a path in \mathcal{A} that stays below ℓ corresponds to a path along the edges of T_Δ that stays below ℓ , and vice versa.

► **Lemma 11.** *For any critical vertex $v_\Delta \in T_\Delta$, sink $u_\Delta \in T_\Delta$ and strictly decreasing path $p_\Delta : v_\Delta \rightsquigarrow u_\Delta$ along the edges of T_Δ , there is a corresponding path $p : v_\Delta \rightsquigarrow u_\Delta$ in \mathcal{A} that is strictly decreasing in height.*

Proof. Consider lowering a plane from $\ell = h_{T_\Delta}(v_\Delta)$ to $h_{T_\Delta}(u_\Delta)$. Since p_Δ is strictly decreasing in height, the plane intersects p_Δ at a single point $x(\ell)$ for all ℓ . The path p consists of edges of \mathcal{A} corresponding to the contours containing $x(\ell)$ for all ℓ . ◀

► **Lemma 12.** *For any pair of nodes $v, w \in \mathcal{A}$ such that v is the highest vertex on the path $p : v \rightsquigarrow w$ in \mathcal{A} , there exists a path $p_\Delta : v \rightsquigarrow w$ along the edges of T_Δ such that the highest vertex on p_Δ is v .*

Proof. First, we observe that for each vertex $v \in \mathcal{A}$ and down-contour c of v , there exists a strictly decreasing path $\pi_\Delta(v, c)$ in T_Δ from v through c to a sink u . By applying Lemma 11, we obtain a path $\pi_\mathcal{A}(v, c) : v \rightsquigarrow u$ in \mathcal{A} . We show how to find a path in T_Δ between any pair $v, w \in \mathcal{A}$ using the paths π_Δ , as follows: We proceed by induction in the list of contour tree nodes sorted in increasing height order. Fix $v \in \mathcal{A}$ and suppose that, for all pairs $v', w \in \mathcal{A}$ such that $h_{T_\Delta}(v') < h_{T_\Delta}(v)$ and v' is the highest vertex on the path from v' to w in \mathcal{A} , there exists a path from v' to w in T_Δ having v' as the highest vertex. We have to show that for any $w \in \mathcal{A}$ and path $p : v \rightsquigarrow w$ where v is the highest vertex on p , there is a corresponding path along the edges of T_Δ having v as the highest vertex. Let c be the down-contour represented by the edge of p incident to v , and let u be the sink such that $\pi(v, c)$ connects v to u in T_Δ . By induction, u is connected to w in T_Δ by a path p_Δ such that w is the highest vertex. By concatenating $\pi_\Delta(v, c) : u \rightsquigarrow w$ and the reverse of $p_\Delta : w \rightsquigarrow u$, we obtain a path from v to w such that v is the highest vertex on the path. ◀

A.2 Updating \mathcal{A} without using I/Os

In this section we describe how updates to \mathcal{A} in HEIGHT-UPDATE can be handled without I/Os. First we give a brief description of how the data structure of Yang [21] handles updates to \mathcal{A} . Then we describe how our sea-level flooding data structure avoids I/Os when the contour tree is updated.

Yang describes the *dynamic forest* data structure that is an I/O-efficient data structure used to store the descent tree Π_\downarrow and ascent tree Π_\uparrow . The data structure represents a forest of rooted trees, and for a forest of N vertices the following operations are supported: Returning the *root* of a vertex u in $O(\log_B N)$ I/Os and *linking* a root u to a vertex v or *disconnecting* a non-root u from its parent in $O(\log_B^2 N)$ I/Os. The dynamic forest is represented by its *Euler tour* [19, 18], using the observation that cutting or linking an edge corresponds to a constant number of splits and merges in the Euler tour. The Euler tour is stored in a

level-balanced B -tree [1] that supports split and merge in $O(\log_B^2 N)$ I/Os. Each vertex in the forest stores a pointer to its first and last occurrences in the Euler tour, which allows cutting or linking an edge in $O(\log_B^2 N)$ I/Os.

It is straightforward to augment the level-balanced B -tree to support the *rank* operation, which returns the number of elements before a given element in the sequence in $O(\log_B N)$ I/Os. The ranks of Euler tour occurrences can be used to answer subtree queries: Given vertices u and v , u is in the subtree rooted at v if and only if the rank of the first occurrence of u is contained in the interval spanned by the ranks of the first and last occurrences of v .

Let L be the set of vertices adjacent to saddle vertices in a TIN T_Δ ; the size of L is at most $8X$, where X is the number of critical vertices of T_Δ . Our sea-level flooding data structure stores, for each vertex v in L , the rank of the first occurrence of v in Π_\downarrow and Π_\uparrow , and it stores for each sink (peak) u the ranks of the first and last occurrences of u in Π_\downarrow (Π_\uparrow). As described above, from the ranks of u and v it can be determined whether v is in the subtree of u in Π_\downarrow or Π_\uparrow .

When the data structure of Yang [21] handles an interchange event between saddles u and v in \mathcal{A} , that is, the event that u and v swap height order due to an update to the height of u , Π_\downarrow (Π_\uparrow) is queried with a vertex from each lower (higher) sequence of u and v to determine the new nesting structure of contours. By answering the queries to Π_\downarrow and Π_\uparrow using the ranks stored in main memory, the queries require no I/Os; since this is the only case in which Π_\downarrow or Π_\uparrow is queried by the update algorithm for \mathcal{A} , our data structure may update \mathcal{A} without I/Os.

During our update algorithm, Π_\downarrow and Π_\uparrow are updated to reflect the new descending and ascending connectivity of the vertices in the updated square Q_B . Whenever an edge is disconnected or reconnected in Π_\downarrow or Π_\uparrow , this operation is translated into a constant number of splits and merges to the level-balanced B -trees underlying Π_\downarrow and Π_\uparrow , and these splits and merges can cause the ranks of vertices in L to change. By using the *rank* operation on the level-balanced B -tree before and after each such split or merge operation, it is straightforward to update the stored ranks of vertices in L accordingly.

As we assume that X is less than M , L fits in main memory, and thus querying and updating L incurs no I/Os.

► **Lemma 13.** *For Π_\downarrow (Π_\uparrow), the ranks of L and terrain sinks (peaks) can be maintained in internal memory by the sea-level flooding data structure such that queries to the sink (peak) reached when following edges in Π_\downarrow (Π_\uparrow) from a vertex $v \in L$ can be answered without I/Os.*

A.3 Analyzing the coastal minima

In this section we show that the contour tree augmented with coastal minima, denoted by \mathcal{A}^+ , contains enough information to determine which sinks are flooded. Recall that a *coastal minimum* is a coastal vertex u such that no other coastal vertex in $C(u)$ is below u . We define *coastal maxima* analogously to coastal minima. A coastal minimum (maximum) is a *true coastal minimum (maximum)* if it has no lower (higher) neighbor that is a coastal vertex in any coastal region; otherwise it is a *region minimum (maximum)*.

► **Lemma 14.** *Let X be the number of sinks, saddles and peaks in the terrain, and let $|C|$ be the number of coastal regions.*

- (i) *If a coastal vertex u in coastal region R floods a terrain vertex v , then there is a coastal minimum w in R that also floods v .*
- (ii) *The number of coastal minima is $O(X + |C|)$.*

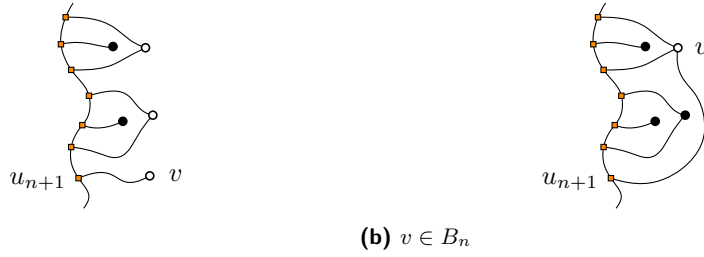
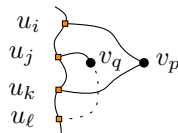


Figure 7 The two cases when u_{n+1} is added. Open and closed vertices are marked with circles and disks, respectively.

Proof. First, we show (i). Suppose a coastal vertex u in coastal region R floods a terrain vertex v ; we have to show that there exists a coastal minimum w in R that also floods v . Let $p : u \rightsquigarrow v$ be a flood path from u to v . Now we construct another flood path $p' : u \rightsquigarrow w \rightsquigarrow u \rightsquigarrow v$ that goes through the coastal minimum w in R reached when following a descending path in R from u until reaching a coastal minimum. Then the path $w \rightsquigarrow u \rightsquigarrow v$ is a flood path from the coastal minimum w to v .

Next, we show (ii). We define the *lowest descent path* p from a given vertex u as the path along the edges of T_Δ starting in u and ending in a sink, such that for each edge (v, w) in p , w is the lowest neighbor of v ; the *highest ascent path* is defined analogously. If two lowest descent paths or two highest ascent paths p, q intersect, then they share a common suffix; as such, p and q do not cross. From each true coastal minimum, we follow the lowest descent path to reach a sink, and from each true coastal maximum, we follow the highest ascent path to reach a peak. Note that an ascending path and a descending path cannot cross, and if two paths share a common suffix, we can separate them so that all paths form a planar bipartite graph $G = (A \cup B, E)$, where A is the set of true coastal minima and maxima, and B is the set of sinks and peaks in the terrain. Let $u_1, \dots, u_{|A|}$ be the vertices in A labeled in the order that they appear as we traverse the coastline. Each vertex $u_i \in A$ is connected to exactly one vertex in B by an edge $e(u_i) \in E$. Since any two true coastal minima are separated by a true coastal maximum and vice-versa, we assume without loss of generality that u_1, u_3, \dots are minima and



u_2, u_4, \dots are maxima. Observe that for all $n \geq 1$, u_n and u_{n+1} are connected to distinct vertices in B . Consider three vertices u_i, u_j , and u_k with $i < j < k$, such that u_i and u_k are connected to the same vertex v_p and u_j is connected to vertex v_q . Then v_q cannot be connected to any vertex u_ℓ with $k < \ell$.

To show that $|A| < 2|B|$, we consider constructing G incrementally by adding the vertices of A in the order they appear on the coastline; for each $n \geq 0$ let $G_n = (A_n \cup B_n, E_n)$ be the subgraph of G consisting of $A_n = \{u_1, \dots, u_n\}$, $E_n = \{e(u_1), \dots, e(u_n)\}$, and B_n being the set of B -vertices connected by E_n . For each G_n , we call a vertex $v \in B_n$ *open* if it can be reached from u_{n+1} via a path on the terrain that does not intersect any edge in G_n , and *closed* otherwise. Let C_n (O_n) be the set of closed (open) vertices of B_n . We show by induction in n that $|A_n| \leq |O_n| + 2|C_n|$, from which it follows that $|A_n| < 2|B_n|$ (since not

all vertices in B_n can be closed). Initially, $A_1 = \{u_1\}$, $|O_1| = 1$ and $|C_1| = 0$. When u_{n+1} is added with an edge to a vertex $v \in B_{n+1}$, there are two cases. If $v \notin B_n$ (Figure 7a), then $|C_{n+1}| = |C_n|$ and $|O_{n+1}| = |O_n| + 1$, so

$$|A_{n+1}| = |A_n| + 1 \leq |O_n| + 2|C_n| + 1 = |O_{n+1}| + 2|C_{n+1}|. \quad \blacktriangleleft$$

Otherwise, $v \in B_n$ (Figure 7b). The edge from u_{n+1} to v moves k vertices in O_n to C_{n+1} , so $|O_{n+1}| = |O_n| - k$, $|C_{n+1}| = |C_n| + k$. Since both u_n and u_{n+1} cannot be maxima or minima, they cannot have the same neighbor and therefore the neighbor of u_n is open in G_n and closed in G_{n+1} , which implies that $k \geq 1$. Thus it follows that

$$|A_{n+1}| = |A_n| + 1 \leq |O_n| + 2|C_n| + 1 \leq (|O_n| - k) + 2(|C_n| + k) = |O_{n+1}| + 2|C_{n+1}|. \quad \blacktriangleleft$$

Computing β -Stretch Paths in Drawings of Graphs

Esther M. Arkin

Stony Brook University, NY, USA

Faryad Darabi Sahneh

University of Arizona, Tucson, AZ, USA

Alon Efrat

University of Arizona, Tucson, AZ, USA

Fabian Frank

University of Arizona, Tucson, AZ, USA

Radoslav Fulek

University of Arizona, Tucson, AZ, USA

Stephen Kobourov

University of Arizona, Tucson, AZ, USA

Joseph S. B. Mitchell

Stony Brook University, NY, USA

Abstract

Let f be a drawing in the Euclidean plane of a graph G , which is understood to be a 1-dimensional simplicial complex. We assume that every edge of G is drawn by f as a curve of constant algebraic complexity, and the ratio of the length of the longest simple path to the length of the shortest edge is $\text{poly}(n)$. In the drawing f , a path P of G , or its image in the drawing $\pi = f(P)$, is β -stretch if π is a simple (non-self-intersecting) curve, and for every pair of distinct points $p \in P$ and $q \in P$, the length of the sub-curve of π connecting $f(p)$ with $f(q)$ is at most $\beta \|f(p) - f(q)\|$, where $\|\cdot\|$ denotes the Euclidean distance. We introduce and study the β -stretch Path Problem (β SP for short), in which we are given a pair of vertices s and t of G , and we are to decide whether in the given drawing of G there exists a β -stretch path P connecting s and t . The β SP also asks that we output P if it exists.

The β SP quantifies a notion of “near straightness” for paths in a graph G , motivated by gerrymandering regions in a map, where edges of G represent natural geographical/political boundaries that may be chosen to bound election districts. The notion of a β -stretch path naturally extends to cycles, and the extension gives a measure of how gerrymandered a district is. Furthermore, we show that the extension is closely related to several studied measures of local fatness of geometric shapes.

We prove that β SP is strongly NP-complete. We complement this result by giving a quasi-polynomial time algorithm, that for a given $\varepsilon > 0$, $\beta \in O(\text{poly}(\log |V(G)|))$, and $s, t \in V(G)$, outputs a β -stretch path between s and t , if a $(1 - \varepsilon)\beta$ -stretch path between s and t exists in the drawing.

2012 ACM Subject Classification Mathematics of computing \rightarrow Paths and connectivity problems; Theory of computation \rightarrow Computational geometry

Keywords and phrases stretch factor, dilation, geometric spanners

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.7

Funding This work is supported in part by NSF grants CCF-1526406, CCF-1740858, CCF-1712119, and DMS-1839274.

1 Introduction

We study an optimal path problem in planar drawings of graphs, in which we represent edges as curves of constant algebraic complexity. We seek a path in a graph G from a given vertex s to another given vertex t that is, in a precise sense, as close as possible to the straight-line



© Esther M. Arkin, Faryad Darabi Sahneh, Alon Efrat, Fabian Frank, Radoslav Fulek, Stephen Kobourov, and Joseph S. B. Mitchell;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 7; pp. 7:1–7:20



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

segment from s to t . We formalize this notion by saying that an $s - t$ path is a β -stretch path if the distance between any two points along the path (not only the endpoints) is at most β times the Euclidean distance between them.

The notion of “ β -stretch” in this definition is similar to the notion of stretch in a multiplicative β graph spanner [17], where we want to remove edges from the graph while ensuring that the shortest path distance in the spanner is at most β times the length of a shortest path in the original graph. Thorough reviews of existing results for geometric spanners are available in [4, 9, 16]. In our problem we are not sparsifying the graph; instead, we try to find the most “natural” path connecting two given vertices s and t in a given embedded graph. If we interpret the embedded graph as the road network of a country, such paths can be used as an initial step to partition the country into regions with natural shapes. One of our motivations, in fact, is the problem of computing natural regions that, in a precise sense, avoid gerrymandering. A few definitions have been proposed in the literature to characterize what a “natural” path could entail. For example, a path in a drawing of a graph is defined to be *self-approaching* [1, 12] if for any two points p and q on the path, when moving from p to q along the path, the Euclidean distance to q is decreasing. Icking et al. [12] proved that a self-approaching path is 5.3332-stretch.

The problem of computing β -stretch paths bears similarities to the graph dilation problem, where for every pair of vertices s and t in a geometric graph, we compare the shortest-path distance between s and t to their actual Euclidean distance in the plane, and return the largest ratio of these two values over all pairs (s, t) . In the special case of cycles this problem is known as computing the maximum detour of a polygonal chain [8]. Klein and Kutz show that computing a minimum-dilation graph that connects a given n -point set in the plane with at most m edges is NP-hard [14]. In one direction, if we are given an embedded geometric graph with a dilation ratio that is at most as large as our target stretch factor, a weaker variant of a β -stretch path exists between every pair of vertices $s - t$, in which we consider only pairs of vertices along the path rather than points. However, since the dilation is a global property an $s - t$ path that is β -stretch in the given graph might still exist even if the dilation is more than β . We elaborate on other connections to our problem in Section 1.3.

We naturally extend the notion of β -stretch paths to β -stretch cycles. Interestingly, we show that a β -stretch cycle bounds a locally “fat” shape in the sense as defined by De Berg [7], with the parameter of fatness depending on β . The converse is easily seen not to be true. Our notion of β -stretch cycles may have applications to computing geographic partitions into regions whose shapes are well shaped in a sense that cannot be captured with fatness criteria.

The rest of the paper is organized as the following. We formally define the β -stretch path problem in Section 1.1, followed by key main results and an overview of related results in the literature in Section 1.2 and 1.3, respectively. In Section 2, we prove a relation between β -stretch cycles and locally γ -fat shapes. Section 3 proves that β -stretch path problem is strongly NP-complete. Section 4 develops a quasi-polynomial approximation scheme algorithms for β -stretch path problem and its extension to computing β -stretch cycles. We conclude with open problems and future directions in Section 5. Omitted proofs are in the Appendix (Section 6).

1.1 Problem Statement

Let $G = (V, E)$ be a finite simple graph, with vertex set V and edge set $E \subseteq \binom{V}{2}$. A *drawing* of a graph is a representation of G in the Euclidean plane \mathbb{R}^2 , in which vertices are distinct points and edges are Jordan arcs represented as curves of *constant algebraic complexity*, i.e., described by a constant number of polynomial equations (inequalities), whose maximum

degree is bounded by a fixed constant.

Formally, a drawing of a graph is a continuous map $f : G \rightarrow \mathbb{R}^2$, where we treat G as a 1-dimensional simplicial complex. The representation of a vertex $v \in V$, an edge $e \in E$, and a path $P \subseteq G$ in the drawing f is $f(v)$, $f(e)$, and $f(P)$, respectively. Here, we consider a *generalized path* that can end in a midpoint of an edge.

We will distinguish paths in a graph from paths in a drawing of a graph. The reason is that we will consider “paths” in a drawing that end in relative interiors of edges. Treating G as a 1-dimensional simplicial complex, a *path* in a drawing f of G is $f(P)$, where P is a generalized path in G . We will be denoting paths in a drawing by lower case Greek letters.

Let $\|\cdot\|$ be the Euclidean norm. Let $P \subseteq G$ denote a path between p and $q \in G$. If both p and q are vertices of G then P corresponds to a usual path in G . Let f be a drawing of G . Then $\pi = f(P)$ is the path *between p and q* in f . Let $\pi(p', q')$ denote the sub-path of π between $p', q' \in G$, that is, $\pi(p', q') = f(P(p', q'))$, where $P(p', q') \subseteq P$ is the path between p' and q' . If we want to specify a path π together with its endpoints s and t we denote it by $\pi(s, t) = \pi$. The path π *passes through* all of the vertices and edges of G intersecting P . The *length* of the path π , denoted by $\|\pi\|$, is the usual Euclidean length, which can be computed as $\int_P \|f'(x)\| dx$. The *distance* between $s \in P$ and $t \in P$ along π , denoted by $d_\pi(s, t)$, is the length of the sub-curve of π between $f(s)$ and $f(t)$.

β -stretch path. Let π be a path in f free of self-intersections. For $\beta \geq 1$, path π is a *β -stretch path* if for every $p, q \in P$ we have

$$\frac{d_\pi(p, q)}{\|f(p) - f(q)\|} \leq \beta. \quad (1)$$

β -stretch cycle. Let C be a simple cycle in G so that $\gamma = f(C)$ is free of self-intersections. The cycle γ in f is a *β -stretch cycle* if for every pair of points p and q on C we have

$$\frac{d_\gamma(p, q)}{\|f(p) - f(q)\|} = \frac{\min\{d_\pi(p, q), d_{\pi'}(p, q)\}}{\|f(p) - f(q)\|} \leq \beta, \quad (2)$$

where $\pi = \pi(p, q)$ and $\pi' = \pi'(p, q)$ are the two paths between q and p whose union is γ .

The left hand side of (1) and (2) is the *stretch factor of p and q along π and γ* , respectively. The maximum of the stretch factor of p and q over distinct $p, q \in P$ and $p, q \in C$ is the *stretch factor* of π and γ , respectively. Note that a β -stretch path (cycle) is a β' -stretch path (cycle), for every $\beta' \geq \beta$. If a path π or a cycle γ is self-intersecting, its stretch factor is undefined.

► **Problem 1.** *β -STRETCH PATH PROBLEM (β SP).* We are given a drawing f of a graph G , $\beta \geq 1$, $s \in V(G)$ and $t \in V(G)$. Decide whether there exists a β -stretch path in f between s and t . The instance of the problem is denoted by (G, f, β, s, t) .

A self-intersection-free cycle γ in a drawing f of G *separates* $s \in G \setminus C$ from $t \in G \setminus C$ if $f(s)$ and $f(t)$ are contained in different connected components of the complement of γ in \mathbb{R}^2 .

► **Problem 2.** *β -STRETCH CYCLE PROBLEM (β CP).* We are given a drawing f of a graph G , $\beta \geq 1$, $s \in V(G)$ and $t \in V(G)$. Decide whether there exists a β -stretch cycle in f separating s from t . The instance of the problem is denoted by (G, f, β, s, t) .

1.2 Main Results

Our main results proved in Sections 3, 4.2 and 4.3, respectively, are the following.

► **Theorem 1.** *β SP is strongly NP-complete.*

► **Theorem 2.** *Let (G, f, β, s, t) be an instance for β SP with $\text{poly}(\log n) \geq \beta \geq 1$. Suppose that the shortest edge length in f is 1, and that there exists $c > 0$ such that the longest simple path in f has length at most n^c . Under the above assumptions there exists a QPTAS for β SP. In other words, there exists a quasi-polynomial-time algorithm that for a fixed $\text{poly}(\log n) \geq \beta \geq 1$ and $\varepsilon > 0$ returns a β -stretch path between s and t if a $\beta(1 - \varepsilon)$ -stretch path between s and t exists in f .*

► **Theorem 3.** *Let (G, f, β, s, t) be an instance for β SC with $\text{poly}(\log n) \geq \beta \geq 1$. Suppose that the shortest edge length in f is 1, and that there exists $c > 0$ such that the longest path in f has the length at most n^c . Under the above assumptions there exists a QPTAS for β SC. In other words, there exists a quasi-polynomial-time algorithm that for a fixed $\text{poly}(\log n) \geq \beta \geq 1$ and $\varepsilon > 0$ returns a β -stretch cycle separating s from t if a $\beta(1 - \varepsilon)$ -stretch cycle separating s from t exists in f .*

1.3 Related Work

Dilation or stretch factor [16] is perhaps the most common measure for the quality of a geometric graph. There is a subtle difference between the stretch factor of a path versus the stretch factor of a graph. For a path, the stretch factor only pertains to its endpoints, while for a graph the stretch factor pertains to every pair of the graph vertices. Our definition of β -stretch path falls in the middle as it pertains to all pairs of points belonging to the path.

It is worth mentioning that a line of existing results in the literature is not about designing a geometric graph with desired stretch factor, but about the fast computation of the stretch factor, given the graph. Narasimhan and Smid [15] considered the problem of computing the stretch factor of a Euclidean graph, defined as the maximum ratio of graph distance and Euclidean distance between any two vertices of the graph. Using Callahan and Kosaraju's well-separated pair decomposition, they showed that there exists a EPTAS for computing the stretch factor running in $O(|V|^{3/2})$ time, which is much faster than computing all-pairs-shortest-path distances. For general weighted graphs, Cohen proposed fast algorithms to compute paths with a desired stretch factor [6]. The stretch factor, in this case, is the ratio of the path length to the graph distance. Farshi *et al.* studied the problem of adding an edge to a Euclidean graph that lowers its stretch factor as much as possible [11].

Chen *et al.* [5] recently proposed a new straightness measure for a path. A polygonal chain (p_1, p_2, \dots, p_n) is a c -chain if for all $1 \leq i < j < k \leq n$, we have $\|p_i - p_j\| + \|p_j - p_k\| \leq c\|p_i - p_k\|$. There is a connection between the notion of c -chain and our proposed notion of β -stretch paths. On the one hand, if a chain is β -stretch, it is trivial to show that it is also a β -chain according to the definition in [5]. On the other hand, a c -chain bounds the possible stretch of the chain according to [5, Theorem 1–3]. Even though the analysis is only for the endpoints of the path, the results readily follow for any pair of points on the chain. Hence, it indeed implies the chain has β -stretch (with the difference of only checking pairs of vertices, not the points on the connecting segments).

A closely related notion to our β -stretch path is the notion of quasiconvexity as defined by Azzam and Schul [3]. A connected subset Γ of the Euclidean space is said to be *quasiconvex* if any two points x and y in Γ can be connected via a path in Γ whose length is bounded by a constant times the Euclidean distance between x and y [3]. According to this definition, a

β -stretch path is quasiconvex with constant β . The problem studied by Azzam and Schul is in some sense opposite to ours. Given a connected set Γ and a target set of points K , they compute a superset $\tilde{\Gamma} \supset \Gamma$ that connects the K points, has Hausdorff length comparable to that of Γ , and is quasiconvex. We, instead, look for a path that is a subset of the given connected set (graph) and that is quasiconvex with a constant stretch factor β . While a short quasiconvex set always exists [3, Theorem 1], we show that determining whether a β -path exists is strongly NP-complete.

One measure of “compactness” designed to quantify gerrymandering in political districting is the Polsby-Popper score, based on the ratio of the area of a district to the square of the district’s perimeter [18]. See [19] for a discussion of shape measures used in the study of gerrymandering.

2 β -Stretch Curves and Locally γ -Fat Shapes

In order to model inputs that represent realistic objects, computational geometers introduced the notion of *fat shapes*. The aim of this section is to argue that our notion of β -stretch cycles captures a local variant of fatness.

Roughly speaking, a planar shape, understood as a closed topological disk T , is locally γ -fat if every disk that is centered in T and is not containing the whole T has at least a γ -fraction of its area in T . Let $D \subset \mathbb{R}^2$ denote a disk. Let $D \cap S$, for $S \subseteq \mathbb{R}^2$, denote the path connected component of $D \cap S$ containing the center of D .

Locally γ -fat shape [2, 7]. For $0 \leq \gamma \leq \frac{1}{2}$, a closed topological disk $T \subseteq \mathbb{R}^2$ is locally γ -fat if for every disk D centered in T that does not contain D in its interior, we have $\text{area}(T \cap D) \geq \gamma \cdot \text{area}(D)$.

We remark that there exists a variant of local γ -fatness that considers $\text{area}(T \cap D)$ rather than $\text{area}(T \cap D)$ [20, 21]. The following applies also to this weaker notion of local γ -fatness.

The notion of β -stretch cycles extends to any measurable Jordan curve, in particular, boundaries of “nice” topological disks. In the following theorem, we show that by controlling the stretch factor of the boundary of a topological disk, we also control its local fatness. In particular, lowering the stretch factor increases the fatness. The corresponding lower bound on the local fatness is the inverse of a linear function of the stretch factor with the leading constant factor 2π . We also show that the stretch factor of the boundary cannot be bounded by a function of its local fatness.

► **Theorem 4.** *Every closed topological disk $T \subset \mathbb{R}^2$, whose boundary ∂T is measurable and β -stretch, is locally $\frac{1}{2\pi\beta}$ -fat. For every $\beta > 1$, there exists a locally $\frac{1}{32\pi}$ -fat topological disk whose boundary is not a β -stretch cycle.*

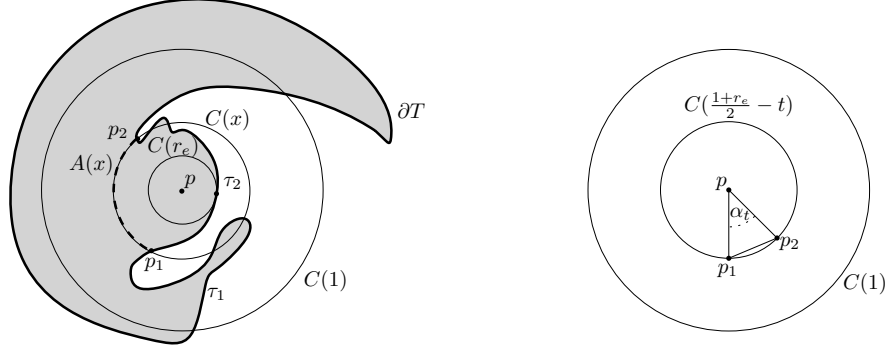
Proof. Let D denote a disk, centered at a point $p \in T$, that does not contain T in its interior. We need to show that $\frac{1}{2\pi\beta} \text{area}(D) \leq \text{area}(T \cap D)$.

Let $D(r)$ and $C(r)$, for $r \geq 0$, denote the disk and circle, respectively, with radius r centered at p . By rescaling, we assume that $D = D(1)$ is a unit disk. Let $r_e = \min\{r \geq 0, (C(r) \cap \partial T) \neq \emptyset\}$. Hence, r_e is the radius of the largest disk $D(r_e)$, whose interior does not intersect ∂T . Since D does not contain T in its interior, we have $r_e \leq 1$.

We will presently show that $\left(r_e^2 + \frac{(1-r_e)^2}{2\pi\beta}\right) \text{area}(D) = \left(r_e^2 + \frac{(1-r_e)^2}{2\pi\beta}\right) \pi \leq \text{area}(T \cap D)$. Then optimizing over the value of r_e , such that $0 \leq r_e \leq 1$, in the previous two inequalities gives the desired lower bound $\frac{1}{2\pi\beta} \text{area}(D)$ on $\text{area}(T \cap D)$. The lower bound is minimized for $r_e = 0$. It remains to show that $\left(r_e^2 + \frac{(1-r_e)^2}{2\pi\beta}\right) \pi \leq \text{area}(T \cap D)$. The first term is due to the fact that $D(r_e) \subseteq T$ since $p \in T$.

7:6 Computing β -Stretch Paths in Drawings of Graphs

To get the second term we consider slices $S(r) = T \cap C(r)$, for $r_e \leq r \leq 1$. First, we treat $r \in [r_e, \frac{1+r_e}{2}]$. We claim that $S(\frac{1+r_e}{2} - t)$, for $0 \leq t \leq \frac{1-r_e}{2}$, contains a circular arc of angular length greater than or equal to $\frac{1}{\beta} \cdot 2 \frac{1-r_e-2t}{1+r_e-2t}$. The claim is proved with the help of the following lemma; see Figure 1 for an illustration.



■ **Figure 1** An illustration of Lemma 5 (left) and inequality (3) (right).

► **Lemma 5.** *The slice $S(x)$, $r_e < x \leq 1$, contains a circular arc $A(x)$, whose relative interior is contained in the interior of $T \cap D$, and whose endpoints $p_1 \in \partial T$ and $p_2 \in \partial T$ split ∂T into two parts τ_1 and τ_2 sharing p_1 and p_2 , such that $\tau_2 \cap C(r_e) \neq \emptyset$ and $\tau_1 \cap C(1) \neq \emptyset$.*

Proof. Refer to Figure 1 (left). First, we perturb ∂T a little bit to eliminate touchings between $C(x)$ and ∂T without increasing the total length of $C(x)$ contained in the interior of T . Let p'_1 and p'_2 denote a point in $\partial T \cap C(r_e)$ and $\partial T \cap C(1)$, respectively. Let τ'_1 and τ'_2 denote the two parts of ∂T connecting p'_1 and p'_2 . We assume that τ'_2 is shortest possible. In particular, τ'_2 is contained in $\partial(T \cap D)$. Note that both τ'_1 and τ'_2 intersect $C(x)$ in an odd number of path connected components.

Let A_1, \dots, A_k denote the path connected components of $T \cap C(x)$. Note that none of A_i 's is a point since we eliminated touchings between ∂T and $C(x)$. It must be that there exists A_j , $1 \leq j \leq k$, such that one endpoint of A_j belongs to τ'_1 and the other to τ'_2 . Indeed, otherwise the number of path connected components in $\tau'_1 \cap C(x)$ and $\tau'_2 \cap C(x)$ would be even.

By the choice of τ'_2 , putting $A(x) = A_j$ concludes the proof. ◀

We show that $A(\frac{1+r_e}{2} - t)$ from Lemma 5 is an arc of the desired angular length, which is at least $\frac{1}{\beta} \cdot 2 \frac{1-r_e-2t}{1+r_e-2t}$. Let τ_1 and τ_2 , and p_1 and p_2 be as in Lemma 5 for $x = \frac{1+r_e}{2} - t$. Note that due to the choice of t and the fact that $C(r_e) \cap \tau_2 \neq \emptyset$, we have $d_{\tau_2}(p_1, p_2) \geq 2(\frac{1-r_e}{2} - t)$. The same inequality holds for $d_{\tau_1}(p_1, p_2)$, since $\tau_1 \cap C(1) \neq \emptyset$. Let α_t denote the smaller angle defined by the rays emanating from p through p_1 and p_2 . Since ∂T is β -stretch, we have, see Figure 1 (right),

$$\beta \geq \frac{2(\frac{1-r_e}{2} - t)}{\|p_1 - p_2\|} = \frac{2(\frac{1-r_e}{2} - t)}{2 \sin \frac{\alpha_t}{2} (\frac{1+r_e}{2} - t)}. \quad (3)$$

The desired lower bound $\frac{1}{2\beta} \cdot \frac{1-r_e-2t}{1+r_e-2t}$ on the angular length of $A(\frac{1+r_e}{2} - t)$ follows since this is lower bounded by $2 \sin \frac{\alpha_t}{2}$.

Similarly we prove that $S(\frac{1+r_e}{2} + t)$, for $0 \leq t \leq \frac{1-r_e}{2}$, contains a circular arc of angular length at least $\frac{1}{\beta} \cdot 2 \frac{1-r_e-2t}{1+r_e+2t}$.

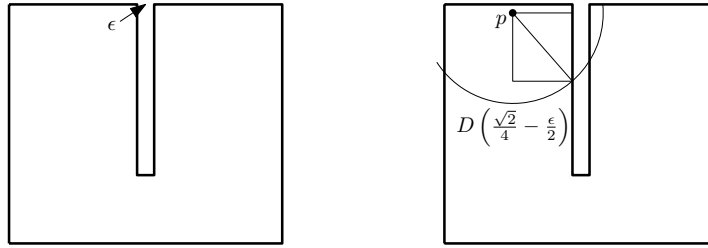
Finally, by summing up infinitesimal thickenings of the slices of width dt we get

$$\begin{aligned} \text{area}(D \cap T) &\geq \frac{1}{2\beta} \int_0^{\frac{1-r_e}{2}} 2 \frac{1-r_e-2t}{1+r_e-2t} \left(\left(\frac{1+r_e}{2} - t \right)^2 - \left(\frac{1+r_e}{2} - t - dt \right)^2 \right) + \\ &+ \frac{1}{2\beta} \int_0^{\frac{1-r_e}{2}} 2 \frac{1-r_e-2t}{1+r_e+2t} \left(\left(\frac{1+r_e}{2} + t \right)^2 - \left(\frac{1+r_e}{2} + t - dt \right)^2 \right), \end{aligned}$$

which simplifies to

$$\text{area}(D \cap T) \geq \frac{2}{\beta} \int_0^{\frac{1-r_e}{2}} (1-r_e-2t) dt.$$

It follows that $\frac{(1-r_e)^2}{2\beta} \leq \text{area}(T \cap D)$, concluding the proof of the first part of the theorem.



■ **Figure 2** A family of topological disks T witnessing that a locally $\frac{1}{32\pi}$ -fat shape can have boundary with an arbitrarily large stretch factor, which is achieved by choosing ϵ arbitrarily small.

Refer to Figure 2. For the second part of the theorem, consider a topological disk T , that is a unit square with an $\epsilon > 0$ wide slit from the middle of an edge to the center as in Figure 2. Clearly, if we choose $\epsilon < \frac{1}{\beta}$ then ∂T is not a β -stretch cycle. However, T stays locally $\frac{1}{32\pi}$ -fat for $\epsilon > 0$. Indeed, it is not hard to see that for $r < \frac{\sqrt{2}}{4} - \frac{\epsilon}{2}$, a disk $D(r)$ centered at a point p in T of radius r has $\text{area}(T \cap D(r)) \geq \left(\frac{r}{\sqrt{2}} \right)^2 > \frac{r^2}{32} = \frac{\text{area}(D(r))}{32\pi}$. For $r \geq \frac{\sqrt{2}}{4} - \frac{\epsilon}{2}$, we have $\text{area}(T \cap D(r)) \geq \frac{1}{16}$, but it is enough to consider $r \leq \sqrt{2}$, since otherwise the whole T is contained in $D(r)$. Hence, $\text{area}(T \cap D(r)) \geq \frac{1}{16} = \frac{2\pi}{32\pi} \geq \frac{\text{area}(D(r))}{32\pi}$. ◀

3 NP-completeness of βSP

The aim of this section is to prove Theorem 1. Let G, f, s and t be as in the statement of the problem βSP . First, we show that we can certify that a given path π in f is a β -stretch path in polynomial time, which follows by the next lemma.

► **Lemma 6.** *Let π be a non-self-intersecting path in f between s and t . There exists a quadratic time algorithm to check if π is a β -stretch path.*

Proof. Note that it is enough to compute the maximum of

$$\max_{s \in e, t \in f} \frac{d_\pi(s, t)}{\|f(s) - f(t)\|}, \tag{4}$$

over pairs of edges e and f on the path P in G such that $\pi = f(P)$. Due to a constant algebraic complexity of edges in f , (4) can be seen as a rational function of two variables whose

maximum can be computed in constant time by the standard calculus and approximated by solving a system of polynomial equations, and therefore the quadratic time complexity follows. \blacktriangleleft

Thus, the problem is in NP, and it remains to argue the NP-hardness. We proceed by a reduction from the graph vertex cover problem, which is one of the first known NP-complete problems from Karp's seminal paper [13], and which we state next. A *vertex cover* in a graph $G = (V, E)$ is a subset V' of its vertex set V such that every edge in E has at least one vertex in V' .

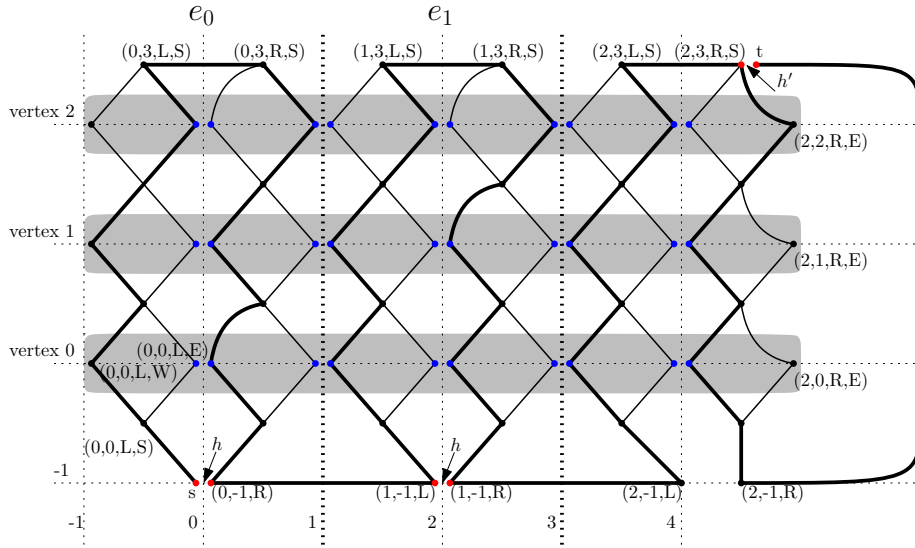
► **Problem 3.** *VERTEX COVER.* We are given a graph G , and a positive integer k . Decide whether there exists a vertex cover in G of size at most k . The instance of the problem is denoted by (G, k) .

For any instance (G, k) of vertex cover we construct an instance (H, f, β, s, t) of β SP that is positive if and only if (G, k) is positive. It will follow from the reduction that β SP is strongly NP-complete, since all of the numerical values in the constructed instance of β SP are bounded by a polynomial in the size of G . The construction follows.

Note that the problem β SP in trees is solvable in quadratic time, by Lemma 6, since in a tree there exists exactly one path between every pair of vertices. Our reduction shows that β SP becomes NP-hard even for graphs whose maximal 2-connected components are cycles.

We put $\beta = n^5$, where n is the number of vertices in G . Let m be the number of edges in G . We identify $V(G)$ with $[n] = \{0, \dots, n-1\}$ and label the edges e_0, \dots, e_{m-1} . The graph $H = (V(H), E(H))$ is constructed as follows; see Figure 3 for an illustration. Roughly, H is composed of chains of 4-cycles arranged in a serial fashion between the distinguished vertices s and t , and drawn as diamonds. Each 4-cycle in a chain (except the two rightmost chains) corresponds to an edge-vertex pair in G , and each pair of consecutive chains except the last one corresponds to an edge of G . Two consecutive chains are joined by an edge or a subdivided edge. The abstract graph H depends only on the number of vertices and edges in G , that is, n and m , and the structure of G is encoded in the drawing of H . Every vertex of H is either a triplet or a 4-tuple: the first element corresponds to an index of an edge of G or is equal to m , the second element corresponds to a vertex of G or is equal to -1 or n , the third element is “L” (for left) or “R” (right), and the fourth element is “E” (for east), “S” (for south) or “W” (for west). Formally, the vertex set is $V(H) = \{s = (0, -1, L), t\} \cup \{(v, e, \alpha, \beta) \mid v \in [n], e \in [m+1], \alpha \in \{L, R\}, \beta \in \{E, S, W\}\} \cup \{(e, n, \alpha, S), (-1, e, \alpha) \mid e \in [m+1], \alpha \in \{L, R\}\}$, and the edge set $E(H) = \{(e, v, \alpha, W)(e, v, \alpha, S), (e, v, \alpha, S)(e, v, \alpha, E), (e, v, \alpha, E)(e, v+1, \alpha, S), (e, v+1, \alpha, S)(e, v, \alpha, W) \mid v \in [n], \alpha \in \{L, R\}, e \in [m+1]\} \cup \{(e, -1, R)(e+1, -1, L), (e, n, L)(e, n, R) \mid e \in [m]\} \cup \{(e, -1, \alpha)(e, 0, \alpha, S) \mid e \in [m+1], \alpha \in \{L, R\}\} \cup \{(m, -1, R)t\}$.

The drawing f represents H in a zig-zag fashion, and has a grid-like structure reminiscent of the edge-vertex incidence matrix of G with rows corresponding to the vertices and columns corresponding to the edges of G . Thus, every chain of 4-cycles of H occupies its own column, and 4-cycles corresponding to the same vertex of G occupy their own row. First, we define $f(v)$ for each $v \in V(H)$. Let $\varepsilon = \beta^{-1} = n^{-5}$. Let $h > 0$ and $h' > 0$ be sufficiently small constants that we specify later. We put $f(t) = (2m + \frac{1}{2} + h', n - \frac{1}{2})$. We put $f((e, -1, L)) = (2e - h, -1)$ and $f((e, -1, R)) = (2e + h, -1)$. We put $f((m, -1, L)) = (2m, -1)$ and $f((m, -1, R)) = (2m + 1, -1)$. We put $f((e, v, L, E)) = (2e - \varepsilon, v)$, $f((e, v, R, E)) = (2e + 1 - \varepsilon, v)$, $f((e, v, L, W)) = (2e - 1 + \varepsilon, v)$, and $f((e, v, R, W)) = (2e + \varepsilon, v)$. We put $f((e, v, L, S)) = (2e - \frac{1}{2}, v - \frac{1}{2})$ and $f((e, v, R, S)) = (2e + \frac{1}{2}, v - \frac{1}{2})$, for $v \in [n]$ and $e \in [m+1]$.



■ **Figure 3** The drawing f of H in the NP-hardness reduction if G is a path on three vertices 0, 1 and 2, with edges $e_0 = 02$ and $e_1 = 21$. Letters in the 3rd and 4th component of a vector representing a vertex stand for Left, Right and East, South, West, respectively. A β -stretch path π between s and t is depicted bold, and corresponds to the minimum vertex cover $VC(\pi)$ of G consisting of the single vertex 2. (A vertex v is contained in $VC(\pi)$ if and only if π passes through $(2, v, R, E)$.)

In f , all of the edges are drawn as straight-line segments except in the following cases. For every $v \in V$ and e_i such that $v \in e_i$, we draw the edge $(i, v, R, W)(i, v + 1, R, S)$ in a close neighborhood of the straight-line segments connecting their end vertices as an xy -monotone curve (that is, a curve that intersects every vertical and horizontal line in at most 1 point) that is longer by more than $20n^{-4}$ in comparison with the straight-line segment $(i, v, R, W)(i, v + 1, R, S)$. We do not care about the shape of the curve and we can think of it as a slightly perturbed line segment. Note that the length of the curve is at most $\sqrt{2}\|f((i, v, R, W)) - f((i, v + 1, R, S))\|$. In the same way, we also draw all of the edges $(m, v, R, E)(m, v + 1, R, S)$, for all $v \in [n]$. Finally, we draw the edge $(m, -1, R)t$ as a concatenation of the horizontal line segment between $f(t)$ and the point $p = f((m, n, R, S)) - (20n^{-4}, 0) \in \mathbb{R}^2$ and a y -monotone curve (that is, every horizontal line intersects the curve at most once) of length $10n$ between $f(m, -1, R)$ and p such that its relative interior does not pass very close to the rest of the drawing.

To finish the drawing $f = f(h, h')$ it remains to choose the values of h and h' . We denote $f_{\text{aux}} = f(0, 0)$ an auxiliary drawing of H with $h = h' = 0$. Let $\pi_e = f_{\text{aux}}(P_e)$ be the 2nd shortest path in f_{aux} between the vertex $(e, -1, L)$ and $(e, -1, R)$, which is independent of the choice of $e \in [m]$. Note that π_e is a path all of whose edges but 1 are drawn as line segments, and its first and last vertex coincide in the drawing. We put $h = \frac{\|\pi_e\|}{2\beta} \leq \frac{20n}{2n^5} = 10n^{-4}$. Let $\pi' = f_{\text{aux}}(P')$ be the $(k + 1)$ -st shortest path in f_{aux} between (m, n, R, S) and t . We put $h' = \frac{\|\pi'\|}{\beta} \leq \frac{20n}{n^5} = 20n^{-4}$. Note that π' is a path with all but $k + 1$ of its edges drawn as line segments, and its first and last vertex t coincide in the drawing.

► **Observation 7.** *The path $f(P_e)$, for $e \in [m]$, and $f(P')$ is shorter than π_e and π' , respectively, and longer than $\|\pi_e\| - 20n^{-4}$ and $\|\pi'\| - 20n^{-4}$.*

For every $v \in [n]$, $e \in [m + 1]$ and $\alpha \in \{L, R\}$, every path in G between s and t must pass either through (e, v, α, W) or (e, v, α, E) . Furthermore, due to the very short distances between blue vertices in the figure we have the following.

7:10 Computing β -Stretch Paths in Drawings of Graphs

► **Lemma 8.** *Let π be a β -stretch path in f between s and t . If π passes through (e, v, L, E) then π passes through (e, v, R, E) and (e', v, α, E) , for all $e' > e$ and $\alpha \in \{L, R\}$. If π passes through (e, v, R, E) then π passes through (e', v, α, E) , for all $e' > e$ and $\alpha \in \{L, R\}$.*

Proof. Suppose that π passes through (e, v, L, E) , and, for the sake of contradiction, let $e' \geq e$ denote the smallest value such that π passes through $(e, v, \alpha, E) \neq (e, v, L, E)$ for some $\alpha \in \{L, R\}$. Suppose that $e = e'$. The other case is treated analogously. By the construction of the drawing f , $\|f((e, v, L, E)) - f((e, v, R, W))\| = 2\epsilon = \frac{2}{\beta}$, and $d_\pi((e, v, L, E), (e, v, R, W)) > 2\epsilon$. Hence, the stretch factor of π is strictly more than β (contradiction). ◀

Proof of Theorem 1. It is easy to verify that the construction of (H, f, β, s, t) can be carried out in polynomial time, and all of the numerical values appearing in the construction of f can be bounded from above by a polynomial function of n , the number of vertices in G . Thus, the strong NP-completeness of β SP follows once we show that (G, k) is a positive instance if and only if (H, f, β, s, t) is a positive instance.

First, if (G, k) is a positive instance, there exists a vertex cover $V' \subseteq V$ of G of size at most k . Let π_{\max} denote the longest path of H in f . Let π be the path in f between s and t passing through (e, v, α, w) if and only if $v \in V'$, for all $e \in [m+1]$ and $\alpha \in \{L, R\}$. We need to show that π is a β -stretch path. Note that π is uniquely determined, and that by the choice of β , the only possible pairs of points that could violate the property of π being a β -stretch path are $(e, -1, L)$ and $(e, -1, R)$, for some $e \in [m]$, and (m, n, R, S) and t . Indeed, it is easy to check that the union of two edges sharing a vertex is always a β -stretch path in f , which follows from the fact that an xy -monotone curve is at most $\sqrt{2}$ -stretch. Hence, in order to violate that π is a β -stretch path, we need to find a pair of points $p \in e_i \in E(H)$ and $q \in e_{i'} \in E(H)$, $e_i \cap e_{i'} = \emptyset$, such that $f(p) \in \pi$, $f(q) \in \pi$, and $\|f(p) - f(q)\| < \frac{\|\pi_{\max}\|}{\beta} < \frac{20n^3}{n^5} = 20n^{-2}$. We can assume that n is sufficiently large such that the pre-image in f of a disk neighborhood of $f(p) \in \mathbb{R}^2$, $p \in H$, with radius $20n^{-2}$ is a single component of H , that does not intersect a pair of edges not sharing a vertex, except when p is very close to $(e, -1, \alpha)$, for some $e \in [m+1]$, $\alpha \in \{L, R\}$, (m, n, R, S) or t , which are colored red in the figure.

Since V' is a vertex cover, we have $d_\pi((i, -1, L), (i, -1, R)) \leq \|\pi_i\|$, for all $i \in [m]$. Indeed, for each $i \in [m]$, the path π misses two non-linear edges incident to $(i, v, R, 0)$ for $v \in e_i$ such that $v \in V'$. Then by Observation 7, $\frac{d_\pi((i, -1, L), (i, -1, R))}{\|f(i, -1, L) - f(i, -1, R)\|} \leq \frac{\|\pi_i\|}{2h} = \beta$. Furthermore, since $|V'| \leq k$, we have $d_\pi((m, n, S, R), t) \leq \|\pi'\|$. Then by Observation 7, $\frac{d_\pi((m, n, S, R), t)}{\|f(m, n, S, R) - f(t)\|} \leq \frac{\|\pi'\|}{h} = \beta$.

Second, if π is a β -stretch path between s and t , let $\text{VC}(\pi) \subseteq V$ be defined as follows. A vertex v is contained in $\text{VC}(\pi)$ if and only if π passes through (m, v, R, E) . Since π is β -stretch, we have $d_\pi((m, n, R, S), t) \leq h'\beta = \frac{\|\pi'\|}{\beta}\beta = \|\pi'\|$. If $|\text{VC}(\pi)| > k$ then by Observation 7 and the length of non-geodesic edges $d_\pi((m, n, R, S), t) > \|\pi'\| - 20n^{-4} + 20n^{-4} = \|\pi'\|$, which is in contradiction with the previous claim. Hence, $|\text{VC}(\pi)| \leq k$. It remains to show that $\text{VC}(\pi)$ is a vertex cover of G .

For the sake of contradiction, suppose that there exists an uncovered edge, that is, an edge $uv = e_i \in E$ such that $e_i \cap \text{VC}(\pi) = \emptyset$. On the one hand, by Lemma 8 and the definition of $\text{VC}(\pi)$, π passes through (i, u, R, W) and (i, v, R, W) . Hence, by Observation 7 and the length of non-geodesic edges, $d_\pi((e, -1, L), (e, -1, R)) > \|\pi_e\| - 20n^{-4} + 20n^{-4} = \|\pi_e\|$. On the other hand, since π is β -stretch, $d_\pi((e, -1, L), (e, -1, R)) \leq 2h\beta = 2\frac{\|\pi_e\|}{2\beta}\beta = \|\pi_e\|$ (contradiction). ◀

Note that our NP-hardness proof involves large stretch values (here, $\beta = n^5$). It would be interesting to show NP-hardness for small stretch values.

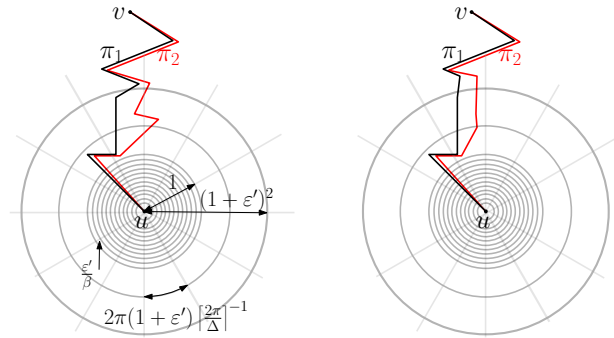
4 Approximation Algorithms

In Section 3, we proved that β SP is strongly NP-complete, which rules out that there exists a FPTAS [22, Section 8] for it, unless $P=NP$; see [22, Corollary 8.6]¹. Let (G, f, β, s, t) be an instance of β SP, and let $\beta^* = \operatorname{argmin}_{\beta}((G, f, \beta, s, t)$ is positive), which is well defined by compactness. In other words, it is highly unlikely that we can approximate β^* within a factor of $(1 + \varepsilon)$, for any $\varepsilon > 0$, in time that is polynomial in both $|V(G)|$ and $\frac{1}{\varepsilon}$.

To complement our hardness result, we show that there exists an algorithm with a quasi-polynomial, that is $O(n^{\operatorname{poly}(\log n)})$, running time that for a given $\varepsilon > 0$ and β , $1 \leq \beta \leq \log^c n$, for some fixed $c \geq 1$, returns a β -stretch path between s and t if a $\beta(1 - \varepsilon)$ -stretch path between s and t exists thereby proving Theorem 2. We assume that ε, c and β satisfy the above properties in the rest of the section. Unless specified otherwise, the base of \log is 2.

4.1 A Path Filtering Scheme

We give a path filtering scheme that we use in Section 4.2 to prove Theorem 2. The main idea behind our algorithm is the following. Since we are aiming only at $\varepsilon > 0$ approximation, we do not need to take into account all of the possible paths between s and t . From a set of paths that are very “similar“ to each other, in the sense that we specify later, we only keep one candidate and delete the rest. Our algorithm proceeds in $\lceil \log n \rceil$ rounds; in the i -th round we compute a set of at most quasi-polynomially many (in terms of n, ε and β) paths of G with at most 2^i edges that are $(1 - \varepsilon_i)\beta$ -stretch in f , for some small ε_i ’s, such that $\varepsilon_0 = \varepsilon, \varepsilon_i > \varepsilon_{i+1}$, and $\varepsilon_{\lceil \log n \rceil} = 0$. In the following, we rigorously define what we mean by “similar”, and how we cluster similar paths. In particular, we cluster paths connecting the same pair of vertices u and v according to their behaviour with respect to stretched radial grids centered at their end vertex u or v ; see Figure 4 for an illustration.



■ **Figure 4** A pair of paths π_1 and π_2 that are not equivalent (on the left) and that are equivalent (on the right) w.r.t. a radial grid centered at u .

Radial grid. Let $\varepsilon > 0$, $\varepsilon' = \varepsilon/\beta$, $r_i = (1 + \varepsilon')^i$ and $\Delta = \frac{\varepsilon'}{1 + \varepsilon'}$. The *radial grid* $F_u(\varepsilon, \beta)$ centered at a point (vertex) $u \in V(G)$ consists of $\lceil \frac{\beta}{\varepsilon'} \rceil$ circles centered at $f(u)$ of radius $i \frac{\varepsilon'}{\beta}$, for $i \in \left[\left\lceil \frac{\beta}{\varepsilon'} \right\rceil \right]$, and circles of radius r_i , for $i \in \lceil [c \log_{1 + \varepsilon'} n] + 1 \rceil$, and $D = \lceil \frac{2\pi}{\Delta} \rceil$ equiangular

¹ Indeed, we can place the vertices in the construction of the reduction on a grid of polynomial size in $n = |V(G)|$ with the unit corresponding to $n^{1/10}$.

7:12 Computing β -Stretch Paths in Drawings of Graphs

spaced rays emanating from $f(u)$. (Recall that we assumed that the shortest edge has length 1 and the largest simple path length is n^c for some constant $c > 0$.) The complement of the radial grid $F_u(\varepsilon, \beta)$ in \mathbb{R}^2 consists of at most $N = D \cdot (\lceil \frac{1}{\varepsilon'} \rceil + \log_{1+\varepsilon'} n^c) = O(\text{poly}(\log n))$ two-dimensional open path connected components, whose closures are *cells* of $F_u(\varepsilon, \beta)$. Note that, ε is treated as a constant and $\beta = O(\text{poly}(\log n))$ by the hypothesis of Theorem 2. In the following, we disregard unbounded cells since they do not intersect $f(G)$. Without loss of generality, we assume that $F_u(\varepsilon, \beta)$ is sufficiently generic with respect to f , that is, $F_u(\varepsilon, \beta) \cap f(G)$ consists of a finite set of points. To this end we might need to slightly perturb the value of ε .

Let $\pi = \pi(u, v)$ be a path in f . Let Σ_π^u denote the subset of cells of $F_u(\varepsilon, \beta)$ that π intersects. We group paths $\pi = \pi(u, v)$ between u and v according to Σ_π^u and approximate distances between u and cells σ in Σ_π^u , which we define next. Let $d_\pi(\sigma, u)$ be the minimum length of the sub-path of π between the point p on π such that $f(p) \in \sigma$ and u . Let r_σ denote the Euclidean distance from u to a furthest point in σ from u . Let $\Xi_\pi^u = \Xi_\pi^u(\varepsilon, \beta) = \left\{ \left(\sigma, \left\lfloor \log_{1+\varepsilon'} \frac{d_\pi(\sigma, u)}{r_\sigma} \right\rfloor \right) \mid \sigma \in \Sigma_\pi^u \right\}$. If π is a β -stretch path, then $\frac{d_\pi(\sigma, u)}{r_\sigma} \leq \beta$. Therefore the second component of each pair in Ξ_π^u is a natural number not bigger than $\lfloor \log_{1+\varepsilon'} \beta \rfloor$.

Path equivalence. Two paths $\pi = \pi(u, v)$ and $\pi' = \pi'(u, v)$ are *equivalent with respect to the radial grid $F_u(\varepsilon, \beta)$* if the first and last edge of π and π' are identical, $\Xi_\pi^u(\varepsilon, \beta) = \Xi_{\pi'}^u(\varepsilon, \beta)$, and the length of π differs from the length of π' by a multiplicative factor of at most $(1 + \varepsilon)$.

Intuitively, equivalent paths pass through the same cells with almost similar distances from u to each intersected cell. Let N be as above, the number of the cells, and $k = \lfloor \log_{1+\varepsilon'} \beta \rfloor + 1$. The crucial aspect of the grid $F_u(\varepsilon, \beta)$ is that there are at most k^N pairwise non-equivalent paths. We have $k^N = (\log_{1+\varepsilon'} \beta)^{cD(\lceil \frac{1}{\varepsilon'} \rceil + \log_{1+\varepsilon'} n)} = O(\text{poly}(\log n)^{\text{poly}(\log n)}) = O(n^{\text{poly}(\log \log n)})$, which is quasi-polynomial in n .

The following lemma (proved in Section 6.1) quantifies the approximation guarantee of our filtering scheme.

► **Lemma 9.** *Let $j \in \mathbb{N}$ such that $j \geq 2$. Let $\pi_1 = \pi_1(u = v_0, v_1), \pi_2 = \pi_2(v_1, v_2) \dots, \pi_j = \pi_j(v_{j-1}, w = v_j)$, and $\pi'_1 = \pi'_1(u = v_0, v_1), \pi'_2 = \pi'_2(v_1, v_2), \dots, \pi'_j = \pi'_j(v_{j-1}, w = v_j)$ be β -stretch paths such that π_i and π'_i , for every $1 \leq i < j$, are equivalent with respect to $F_{v_i}(\varepsilon, \beta_0)$ and $F_{v_{i-1}}(\varepsilon, \beta_0)$, for some $\beta_0 \geq \beta$. Then the following holds.*

If $\pi = \pi_1 \frown \pi_2 \frown \dots \frown \pi_j$ is not a β -stretch path, then $\pi' = \pi'_1 \frown \pi'_2 \frown \dots \frown \pi'_j$ is not a $(1 - 31\varepsilon)\beta$ -stretch path.

4.2 Approximation algorithm for paths

We give an algorithm proving Theorem 2. Refer to the pseudo-code of Algorithm 1. We initialize $\Psi_0 := E(G)$ and $\varepsilon' := \frac{\ln(1-\varepsilon)^{-1}}{32^{\lceil \log n \rceil}}$. The algorithm proceeds in $\lceil \log n \rceil$ many steps, and in the i -th step it computes a set of $\frac{1-\varepsilon}{(1-31\varepsilon')^i} \beta$ -stretch paths Ψ_i in G such that every path in Ψ_i has at most 2^i edges. The set Ψ_{i+1} is computed from $\Psi_{\leq i} = \bigcup_{j \leq i} \Psi_j$ as follows. We pick every pair of distinct paths $\pi_1(u, v) \in \Psi_{\leq i}$ and $\pi_2(v, w) \in \Psi_{\leq i}$ such that the concatenation $\pi = \pi(u, w) = \pi_1(u, v) \frown \pi_2(v, w)$ is a self-intersection free path with at least $2^i + 1$ edges. We put π into Ψ_{i+1} if π is a $\frac{1-\varepsilon}{(1-31\varepsilon')^{i+1}} \beta$ -stretch path. At the end of the $(i + 1)$ -st step, we recursively delete for every pair of vertices u and v of G in Ψ_{i+1} a path $\pi'(u, v)$ if an equivalent path $\pi'(u, v)$ with respect to $F_u(\varepsilon', \beta)$ and $F_v(\varepsilon', \beta)$ still exists in Ψ_{i+1} .

The algorithm outputs a β -stretch path between s and t if $\Psi_{\leq \lceil \log n \rceil}$ contains such a path.

Correctness. Suppose that there exists a $(1 - \varepsilon)\beta$ -stretch path π_0 in f connecting s and t with ℓ edges. We show that the algorithm outputs a β -stretch path connecting s and t . We show by induction on i that after the i -th step of the algorithm, in $\Psi_{\leq i}$ there exists a sequence S_i of $\lceil \frac{\ell}{2^i} \rceil$ paths, whose concatenation is a $\beta \frac{1-\varepsilon}{(1-31\varepsilon')^i}$ -stretch path π_i between s and t . If the claim holds, we are done, since, for a sufficiently large n , we have

$$(1 - 31\varepsilon')^{-\lceil \log n \rceil} (1 - \varepsilon)\beta = \left(1 - \frac{31 \ln(1 - \varepsilon)^{-1}}{32 \lceil \log n \rceil}\right)^{-\lceil \log n \rceil} (1 - \varepsilon)\beta < e^{\ln(1 - \varepsilon)^{-1}} (1 - \varepsilon)\beta = \beta.$$

In the base case the claim holds by the existence of π_0 . By the induction hypothesis, we suppose that the claim holds after the i -th round. We apply Lemma 9 with $\beta_0 := \beta, \varepsilon := \varepsilon'$, and $\beta := \beta \frac{1-\varepsilon}{(1-31\varepsilon')^i}$ to the paths in S_i , whose concatenation π_i in the given order plays the role of π' , and to the equivalent representatives of consecutive pairs of paths in S_i that were not deleted from $\Psi_{\leq i+1}$, whose concatenation plays the role of π . It follows that π is $\beta \frac{1-\varepsilon}{(1-31\varepsilon')^{i+1}}$ -stretch yielding S_{i+1} . Putting $\pi_{i+1} = \pi$ concludes the proof of the correctness of the algorithm.

Running time. The bottleneck of the algorithm is clearly the path filtering scheme that filters all but quasi-polynomially many paths, and therefore the claimed running time follows by the fact that the algorithm ends in $\lceil \log n \rceil$ steps and Lemma 6.

■ **Algorithm 1** Approximation algorithm.

Data: An instance of β SP (G, f, β, s, t) and $\varepsilon > 0$.

Result: A β -stretch path between s and t in f if a $(\beta(1 - \varepsilon))$ -stretch path between s and t exists. (The algorithm can possibly output a β -stretch path even if no $(\beta(1 - \varepsilon))$ -stretch path exists.)

$\varepsilon' := \frac{\ln(1-\varepsilon)^{-1}}{32 \lceil \log n \rceil}$;

$\Psi_0 := E(G), i := 0$; (Ψ_i : the set of candidate β -stretch paths with at most 2^i edges.)

while $\Psi_i \neq \emptyset$ **do**

$\Psi_{i+1} := \emptyset$;

for $\pi_1(u, v), \pi_2(v, w) \in \bigcup_{j \leq i} \Psi_j$ **do**

if $\pi = \pi(u, w) = \pi_1(u, v) \frown \pi_2(v, w)$ has at least $2^i + 1$ edges, and is a $\beta \frac{1-\varepsilon}{(1-31\varepsilon')^{i+1}}$ -stretch path. **then**

 add π to Ψ_{i+1}

while there exists two equivalent paths $\pi(u, v)$ and $\pi'(u, v)$ with respect to

$F_u(\varepsilon', \beta)$ and $F_v(\varepsilon', \beta)$ in Ψ_{i+1} . **do**

 remove π from Ψ_{i+1}

$i \leftarrow i + 1$;

return A β -stretch path between s and t if $\bigcup_i \Psi_i$ contains such path.

4.3 Approximation Algorithm for Cycles

We discuss an extension of the algorithm from Section 4.2 from paths to cycles thereby establishing Theorem 3. Let (G, f, β, s, t) be the input instance for β CP. Let $G_0 = G \setminus \{s, t\}$. We subdivide the edges of G_0 such that every edge has the length at least 1 and at most 2 in f . Let f_0 denote the drawing of G_0 inherited from f . The graph G_0 has polynomially many vertices in terms of the number of vertices of G . We will work with the input instance $(G_0, f_0, \beta, s_0, t_0)$ of β SP, where $s_0, t_0 \in V(G_0)$ and $\varepsilon_0 = 1 - \sqrt{1 - \varepsilon}$. The reason for the

7:14 Computing β -Stretch Paths in Drawings of Graphs

choice of smaller ε_0 is that we will need to work with ε_0 such that $(1 - \varepsilon_0)^2 = (1 - \varepsilon)$. Intuitively, we try to combine all pairs of paths joining the same pair of vertices in $\Psi_{\leq \lceil \log n \rceil}$ constructed by the algorithm from Section 4.2.

A self-intersection free cycle in f_0 separates $f_0(s)$ from $f_0(t)$ if and only if it crosses the line segment between $f_0(s)$ and $f_0(t)$ an odd number of times. In order to keep track of the parity of crossings of paths with the line segment between s and t , we extend the path filtering scheme from Section 4.1 as follows.

Path equivalence. Two paths $\pi = \pi(u, v)$ and $\pi' = \pi'(u, v)$ are *equivalent with respect to the radial grid* $F_u(\varepsilon, \beta)$ in f_0 if the first and last edge of π and π' are identical, $\Xi_\pi^u(\varepsilon, \beta) = \Xi_{\pi'}^u(\varepsilon, \beta)$, the length of π differs from the length of π' by a multiplicative factor of at most $(1 + \varepsilon)$, and additionally the parities of the number of crossings of π' and π with the line segment connecting $f_0(s)$ and $f_0(t)$ are the same.

Algorithm. First, we run a brute-force algorithm to find a β -stretch separating cycle C such that the length of $\gamma = f(C)$ is at least $\frac{4}{\varepsilon_0} + 2$. If we fail to find a β -stretch cycle C , we run the algorithm from Section 4.2 with the input instance $(G_0, f_0, \beta, s_0, t_0)$, for $\varepsilon_0 > 0$, using the previously modified notion of path equivalence with radial grids parametrized by $\varepsilon'(\varepsilon_0) = \frac{\ln(1-\varepsilon_0)^{-1}}{3200 \lceil \log n \rceil}$ and β , that is, $F_u(\varepsilon'/100, \beta)$ rather than $F_u(\varepsilon', \beta)$ in comparison with the original algorithm. The algorithm returns $\Psi_{\leq \lceil \log n \rceil}$. We check if there exists a pair of paths in $\Psi_{\leq \lceil \log n \rceil}$, whose concatenation is a β -stretch cycle C separating s from t . If this is the case we output C .

Correctness. Suppose that there exists a $(1 - \varepsilon)\beta$ -stretch cycle $\gamma = f(C)$ in G_0 separating s from t . Let P_1 and P_2 denote a pair of paths in G between $u \in V(G_0)$ and $v \in V(G_0)$, whose union is C . We choose P_1 and P_2 so that the difference of the length of $\pi_1 = f(P_1)$ and $\pi_2 = f(P_2)$ is minimized. Note that this difference is at most 2. Suppose that π_1 is not shorter than π_2 . We claim that π_1 and π_2 are $\frac{1-\varepsilon}{1-\varepsilon_0}\beta$ -stretch paths. Indeed, for any $p_1, p_2 \in P_1$ $d_\gamma(p_1, p_2) \geq d_{\pi_1}(p_1, p_2) - 2 \geq (1 - \varepsilon_0)d_{\pi_1}(p_1, p_2)$. The first inequality is by the choice of P_1 and P_2 , and the second one by the fact that the length of π_1 is at least $\frac{2}{\varepsilon_0}$, since the length of γ is at least $\frac{4}{\varepsilon_0} + 2$.

Note that $\frac{1-\varepsilon}{1-\varepsilon_0}\beta = (1 - \varepsilon_0)\beta$. Mimicking the proof of the correctness of the algorithm from Section 4.2, we derive that $\Psi_{\leq \lceil \log n \rceil}$ contains a pair of $(1 - \varepsilon_0)\beta$ -stretch paths P'_1 and P'_2 joining the same pair of vertices at P_1 and P_2 such that the concatenation of $\pi'_1 = f_0(P'_1)$ and $\pi'_2 = f_0(P'_2)$ is a β -stretch cycle γ' . To this end we need to adapt Lemma 9 to the case when $u = w$.

► **Lemma 10.** *Let $\varepsilon > 0$ be sufficiently small. Let $j \in \mathbb{N}$ such that $j \geq 2$. Let $\pi_1 = \pi_1(u = v_0, v_1), \pi_2 = \pi_2(v_1, v_2) \dots, \pi_j = \pi_j(v_{j-1}, u = v_j)$, and $\pi'_1 = \pi'_1(u = v_0, v_1), \pi'_2 = \pi'_2(v_1, v_2), \dots, \pi'_j = \pi'_j(v_{j-1}, u = v_j)$ be β -stretch paths such that π_i and π'_i , for every $0 \leq i \leq j$, are equivalent with respect to $F_{v_i}(\varepsilon/100, \beta_0)$ and $F_{v_{i-1}}(\varepsilon/100, \beta_0)$, for some $\beta_0 \geq \beta$. Then the following holds. If $\gamma = \pi_1 \widehat{\cap} \pi_2 \widehat{\cap} \dots \widehat{\cap} \pi_j$ has length at least 20 , and is not a β -stretch cycle, then $\gamma' = \pi'_1 \widehat{\cap} \pi'_2 \widehat{\cap} \dots \widehat{\cap} \pi'_j$ is not a $(1 - 31\varepsilon)\beta$ -stretch cycle. Furthermore, γ separates s from t if and only if γ' separates s from t .*

5 Conclusion and Future Work

We proved that β SP is strongly NP-complete, but our reduction seems to work only with large β that is polynomial in the number of vertices n of the input graph. A natural open problem is to determine the complexity of β SP for β constant or logarithmic in n . We proposed a

quasi-polynomial algorithm for β SP that works only for β that is at most logarithmic in n , and that has a quasi-polynomial running already for constant values of β . Therefore we find the problem of devising a PTAS for β SP interesting even when β is a fixed constant.

This leads us to suspect that devising an approximation algorithm for β SP becomes easier if we restrict ourselves to drawings of graphs in which the vertex set is supported by an integer grid of a polynomial size and edges are straight-line segments.

In the future, we intend to extend our work in the following direction, motivated by the computation of districts that avoid gerrymandering. We mark some vertices in a plane graph as “important” and we wish to cut the graph into regions, whose boundaries are β -stretch cycles, such that each region contains exactly one important vertex. A related work by Eppstein et al. [10] describes a method for defining geographic districts in road networks using stable matching. However, their resulting regions might even be disconnected. As we discussed in Section 2, the β -stretch condition is more constraining than local fatness; a locally fat region, whose boundary has a large stretch factor, might look like the shape in Figure 2, which is indicative of a gerrymandered district, with a selective slit removed. We propose that partitioning of geographic regions using β -stretch paths/cycles can lead to districting solutions that may better avoid gerrymandering. We leave this work for future study.

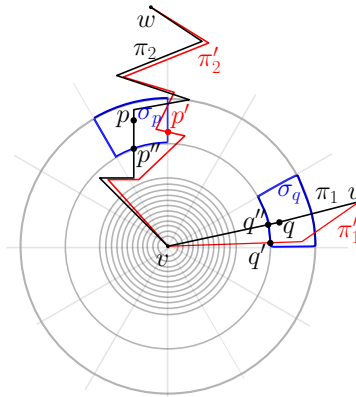
References

- 1 Soroush Alamdari, Timothy M Chan, Elyot Grant, Anna Lubiw, and Vinayak Pathak. Self-approaching graphs. In *International Symposium on Graph Drawing*, pages 260–271. Springer, 2012.
- 2 Boris Aronov, Mark De Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM Journal on Computing*, 43(2):543–572, 2014.
- 3 Jonas Azzam and Raanan Schul. How to take shortcuts in Euclidean space: making a given set into a short quasi-convex set. *Proceedings of the London Mathematical Society*, 105(2):367–392, 2012.
- 4 Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry*, 46(7):818–830, 2013.
- 5 Ke Chen, Adrian Dumitrescu, Wolfgang Mulzer, and Csaba D Tóth. On the stretch factor of polygonal chains. *arXiv preprint arXiv:1906.10217*, 2019.
- 6 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998.
- 7 Mark de Berg. Improved bounds on the union complexity of fat objects. *Discrete & Computational Geometry*, 40(1):127–140, 2008.
- 8 Annette Ebbers-Baumann, Rolf Klein, Elmar Langetepe, and Andrzej Lingas. A fast algorithm for approximating the detour of a polygonal chain. *Computational Geometry*, 27(2):123–134, 2004.
- 9 David Eppstein. Spanning trees and spanners. *Handbook of computational geometry*, pages 425–461, 1999.
- 10 David Eppstein, Michael T. Goodrich, Doruk Korkmaz, and Nil Mamano. Defining equitable geographic districts in road networks via stable matching. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, pages 52:1–52:4, 2017.
- 11 Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2008.
- 12 Christian Icking, Rolf Klein, and Elmar Langetepe. Self-approaching curves. *Mathematical Proceedings of the Cambridge Philosophical Society*, 125(3):441–453, 1999.

- 13 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 14 Rolf Klein and Martin Kutz. Computing geometric minimum-dilation graphs is np-hard. In *International Symposium on Graph Drawing*, pages 196–207. Springer, 2006.
- 15 Giri Narasimhan and Michiel Smid. Approximating the stretch factor of euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.
- 16 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007.
- 17 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.
- 18 Daniel D. Polsby and Robert D. Popper. The third criterion: Compactness as a procedural safeguard against partisan gerrymandering. *Yale Law and Policy Review*, 9(2):301–353, 1991.
- 19 Kristopher Tapp. Measuring political gerrymandering. *The American Mathematical Monthly*, 126(7):593–609, 2019.
- 20 A Frank van der Stappen, Dan Halperin, and Mark H Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Computational Geometry*, 3(6):353–373, 1993.
- 21 A Frank van der Stappen and Mark H Overmars. Motion planning amidst fat obstacles. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 31–40. ACM, 1994.
- 22 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

6 Appendix

6.1 Proof of Lemma 9



■ **Figure 5** An illustration of Lemma 9 when $j = 2$. A radial grid centered at v_1 , and a pair of paths $\pi = \pi_1 \frown \pi_2$ and $\pi' = \pi'_1 \frown \pi'_2$ that are equivalent with respect to the radial grid centered at v_1 .

► **Lemma 9.** *Let $j \in \mathbb{N}$ such that $j \geq 2$. Let $\pi_1 = \pi_1(u = v_0, v_1), \pi_2 = \pi_2(v_1, v_2) \dots, \pi_j = \pi_j(v_{j-1}, w = v_j)$, and $\pi'_1 = \pi'_1(u = v_0, v_1), \pi'_2 = \pi'_2(v_1, v_2), \dots, \pi'_j = \pi'_j(v_{j-1}, w = v_j)$ be β -stretch paths such that π_i and π'_i , for every $1 \leq i < j$, are equivalent with respect to $F_{v_i}(\varepsilon, \beta_0)$ and $F_{v_{i-1}}(\varepsilon, \beta_0)$, for some $\beta_0 \geq \beta$. Then the following holds.*

If $\pi = \pi_1 \frown \pi_2 \frown \dots \frown \pi_j$ is not a β -stretch path, then $\pi' = \pi'_1 \frown \pi'_2 \frown \dots \frown \pi'_j$ is not a $(1 - 31\varepsilon)\beta$ -stretch path.

Proof. Refer to Figure 5. Assume that π is not a β -stretch path. It follows that either π contains a self-intersection, or there exists two points q and p on π , whose stretch factor is bigger than β . Formally, in either case, there exists a pair of points p and q in G such that

$$\frac{d_\pi(p, q)}{\|f(p) - f(q)\|} > \beta. \quad (5)$$

It is enough to consider the case, in which p is on π_1 and q is on π_j , and p and q are not contained in the union of 2 consecutive edges of π . Indeed, these 2 consecutive edges would be also both on π' by the definition of the equivalent paths.

We show that π' is not a $\beta(1 - 31\varepsilon)$ -stretch path. Consider the cell σ_q and σ_p in the radial grid $F_{v_1}(\varepsilon, \beta_0)$ and $F_{v_{j-1}}(\varepsilon, \beta_0)$, respectively, that contains p and q . Let $q' \in G$ and $q'' \in G$, and $p' \in G$ and $p'' \in G$, respectively, be the points such that $f(q') \in \sigma_q$ and $f(q'') \in \sigma_q$, and $f(p') \in \sigma_p$ and $f(p'') \in \sigma_p$, respectively, minimizing $d_{\pi'}(q', v)$ and $d_{\pi'}(q'', v)$, and $d_{\pi'}(p', v)$ and $d_{\pi'}(p'', v)$. We show that the stretch factor of p' and q' along π' is bigger than $\beta(1 - 16\varepsilon)$, which will conclude the proof. To this end we first derive several simple inequalities.

Since π_1 and π'_1 , and π_j and π'_j are equivalent with respect to $F_{v_1}(\varepsilon, \beta_0)$ and $F_{v_{j-1}}(\varepsilon, \beta_0)$, respectively, the values of $d_{\pi'}(q', v_1)$ and $d_{\pi'}(q'', v_1)$, and $d_{\pi'}(p', v_{j-1})$ and $d_{\pi'}(p'', v_{j-1})$ are within the factor of $(1 + \varepsilon')$ of each other, where $\varepsilon' = \varepsilon/\beta_0$. Since π_1 is a β -stretch paths, $d_\pi(q, q'') \leq \beta L_{\sigma_q}$, where L_{σ_q} is the diameter of σ_q . Therefore

$$d_\pi(q, v_1) = d_\pi(q, q'') + d_\pi(q'', v_1) \leq \beta L_{\sigma_q} + (1 + \varepsilon')d_{\pi'}(q', v_1). \quad (6)$$

The same holds for p, p' and p'' . By the construction of $F_{v_1}(\varepsilon, \beta)$ and $F_{v_{j-1}}(\varepsilon, \beta)$, the diameter of $\sigma \in \{\sigma_p, \sigma_q\}$ such that $r_\sigma = (1 + \varepsilon')^{i+1}$ can be bounded from the above as follows

$$L_\sigma < (1 + \varepsilon')^{i+1} - (1 + \varepsilon')^i + \frac{2\pi\varepsilon'}{1 + \varepsilon'}(1 + \varepsilon')^i \leq (1 + 2\pi)\frac{\varepsilon'}{1 + \varepsilon'}r_\sigma. \quad (7)$$

The upper bound on the diameter of all of the other cells σ contained in the unit disk centered at v_1 and v_{j-1} , respectively, follows if p and q is contained in the annulus between the unit circle and the circle of radius $\frac{1}{\beta_0}$ centered at v_1 and v_{j-1} .

$$L_\sigma < \frac{\varepsilon'}{\beta_0} + \frac{2\pi\varepsilon' \left(r_\sigma - \frac{\varepsilon'}{\beta_0}\right)}{\varepsilon' + 1} < \varepsilon' \left(r_\sigma - \frac{\varepsilon'}{\beta_0}\right) + 2\pi \left(r_\sigma - \frac{\varepsilon'}{\beta_0}\right) \varepsilon' = (1 + 2\pi)\varepsilon' \left(r_\sigma - \frac{\varepsilon'}{\beta_0}\right) \quad (8)$$

By the triangle inequality, $\|f(q) - f(p)\| \geq \|f(q') - f(p')\| - \|f(q) - f(q')\| - \|f(p) - f(p')\| \geq \|f(p') - f(q')\| - L_{\sigma_q} - L_{\sigma_p}$. Therefore

$$\begin{aligned} \beta &\stackrel{(5)}{<} \frac{d_\pi(q, v_1) + d_\pi(v_1, v_2) + \dots + d_\pi(v_{j-1}, p)}{\|f(q) - f(p)\|} \\ &\stackrel{(6)}{\leq} \frac{(1 + \varepsilon')(d_{\pi'}(q', v_1) + \dots + d_{\pi'}(v_{j-1}, p')) + \beta(L_{\sigma_q} + L_{\sigma_p})}{\|f(q') - f(p')\| - L_{\sigma_q} - L_{\sigma_p}} \\ &\leq \frac{d_{\pi'}(q', v_1) + \dots + d_{\pi'}(v_{j-1}, p')}{\|f(q') - f(p')\|} \frac{1 + \varepsilon'}{1 - \frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|}} + \beta \frac{\frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|}}{1 - \frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|}}. \end{aligned} \quad (9)$$

We consider two cases depending on whether π' is a β -stretch path. If π' is not a β -stretch path, then it is also not a $\beta(1 - 16\varepsilon')$ -stretch path and we are done. If π' is a β -stretch path

7:18 Computing β -Stretch Paths in Drawings of Graphs

and both σ_q and σ_p are not contained in the unit disk centered at v_1 and v_{j-1} , respectively, then we must have

$$\|f(p') - f(q')\| \geq \frac{d_{\pi'}(p', q')}{\beta} > \frac{\|f(q') - f(v_1)\| + \|f(v_{j-1}) - f(p')\|}{\beta} \geq \frac{r_{\sigma_q} + r_{\sigma_p}}{(1 + \varepsilon')\beta}. \quad (10)$$

Combining (10) with the upper bound (7) on L_σ from the above yields

$$\frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|} < \frac{(1 + 2\pi)\varepsilon'(r_{\sigma_q} + r_{\sigma_p})}{(r_{\sigma_q} + r_{\sigma_p})/\beta} = (1 + 2\pi)\varepsilon \frac{\beta}{\beta_0} \leq (1 + 2\pi)\varepsilon. \quad (11)$$

If σ_q and σ_p is contained in the annulus between the unit circle and the circle of radius $\frac{1}{\beta_0}$ centered at v_1 and v_{j-1} , respectively, then (10) becomes

$$\|f(p') - f(q')\| > \frac{\|f(q') - f(v_1)\| + \|f(v_{j-1}) - f(p')\|}{\beta} \geq \frac{r_{\sigma_q} - \varepsilon'/\beta_0 + r_{\sigma_p} - \varepsilon'/\beta_0}{\beta}. \quad (12)$$

Then using (8) and (12), we recover the upper bound from (11).

$$\frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|} < \frac{(1 + 2\pi)(r_{\sigma_q} - \varepsilon'/\beta_0 + r_{\sigma_p} - \varepsilon'/\beta_0)\varepsilon'}{\frac{r_{\sigma_q} - \varepsilon'/\beta_0 + r_{\sigma_p} - \varepsilon'/\beta_0}{\beta}} = (1 + 2\pi)\varepsilon \frac{\beta}{\beta_0} \leq (1 + 2\pi)\varepsilon \quad (13)$$

If σ_q is contained in the annulus between the unit circle and the circle of radius $\frac{1}{\beta_0}$ centered at v_1 , and σ_p is not contained in the unit disk centered at v_{j-1} then (10) becomes.

$$\|f(p') - f(q')\| > \frac{\|f(q') - f(v_1)\| + \|f(v_{j-1}) - f(p')\|}{\beta} \geq \frac{\frac{r_{\sigma_p}}{(1 + \varepsilon')} + (r_{\sigma_q} - \frac{\varepsilon'}{\beta_0})}{\beta}. \quad (14)$$

Then using (7),(8) and (10), we again recover the upper bound from (11).

$$\frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|} < \frac{(1 + 2\pi) \left(r_{\sigma_q} - \varepsilon'/\beta_0 + \frac{r_{\sigma_p}}{(1 + \varepsilon')} \right) \varepsilon'}{\frac{\frac{r_{\sigma_p}}{(1 + \varepsilon')} + (r_{\sigma_q} - \varepsilon'/\beta_0)}{\beta}} = (1 + 2\pi)\varepsilon \frac{\beta}{\beta_0} \leq (1 + 2\pi)\varepsilon \quad (15)$$

Finally, if σ_q is contained in the disk of radius $\frac{1}{\beta_0}$ centered at v_1 we distinguish two cases depending on whether σ_p is contained in the unit disk centered at v_{j-1} . If this is the case, q is contained on an edge of π_1 incident to v_j , since π_1 is a β -stretch path, and $\beta_0 \geq \beta$. Hence, as every edge has length at least 1 in f , we have that σ_p is not contained in the unit disk centered at v_{j-1} with diameter $\frac{1}{\beta_0}$. Indeed, q and p are not contained in two consecutive edges of π and therefore they are at distance more than 1 along π , and thus, σ_p is not in the disk of radius $\frac{1}{\beta}$, but $\beta_0 \geq \beta$. Depending on whether σ_p is contained in the unit disk centered at v_{j-1} , we obtain one of the following bounds.

$$\|f(p') - f(q')\| \geq \frac{d_{\pi'}(p', q')}{\beta} > \frac{\|f(v_{j-1}) - f(p')\|}{\beta} \geq \frac{\frac{r_{\sigma_p}}{(1 + \varepsilon')}}{\beta} \quad (16)$$

$$\|f(p') - f(q')\| \geq \frac{d_{\pi'}(p', q')}{\beta} > \frac{\|f(v_{j-1}) - f(p')\|}{\beta} \geq \frac{r_{\sigma_p} - \varepsilon'/\beta_0}{\beta} \quad (17)$$

Then using (7),(8) and (16) and (17), we again recover an upper bound analogous to (11), but worse by a multiplicative factor of 2.

$$\frac{L_{\sigma_q} + L_{\sigma_p}}{\|f(q') - f(p')\|} \leq \frac{2L_{\sigma_p}}{\|f(q') - f(p')\|} \leq 2(1 + 2\pi)\varepsilon \quad (18)$$

Using (11), (13), (15), and (18), (9) can be in every possible case rewritten as follows, which concludes the proof.

$$\begin{aligned} \frac{d_{\pi'}(q', p')}{\|f(q') - f(p')\|} &= \frac{d_{\pi'}(q', v_1) + \dots + d_{\pi'}(v_{j-1}, p')}{\|f(q') - f(p')\|} > \beta \frac{1 - 4(1 + 2\pi)\varepsilon}{1 + \varepsilon/\beta} \\ &> \beta \frac{1 - 4(1 + 2\pi)\varepsilon}{1 + \varepsilon} > \frac{1 - 31\varepsilon}{1 + \varepsilon} \beta > (1 - 31\varepsilon)\beta \end{aligned} \quad \blacktriangleleft$$

6.2 Proof of Lemma 10

► **Lemma 10.** *Let $\varepsilon > 0$ be sufficiently small. Let $j \in \mathbb{N}$ such that $j \geq 2$. Let $\pi_1 = \pi_1(u = v_0, v_1), \pi_2 = \pi_2(v_1, v_2) \dots, \pi_j = \pi_2(v_{j-1}, u = v_j)$, and $\pi'_1 = \pi'_1(u = v_0, v_1), \pi'_2 = \pi'_2(v_1, v_2), \dots, \pi'_j = \pi'_j(v_{j-1}, u = v_j)$ be β -stretch paths such that π_i and π'_i , for every $0 \leq i \leq j$, are equivalent with respect to $F_{v_i}(\varepsilon/100, \beta_0)$ and $F_{v_{i-1}}(\varepsilon/100, \beta_0)$, for some $\beta_0 \geq \beta$. Then the following holds. If $\gamma = \pi_1 \widehat{\cap} \pi_2 \widehat{\cap} \dots \widehat{\cap} \pi_j$ has length at least 20, and is not a β -stretch cycle, then $\gamma' = \pi'_1 \widehat{\cap} \pi'_2 \widehat{\cap} \dots \widehat{\cap} \pi'_j$ is not a $(1 - 31\varepsilon)\beta$ -stretch cycle. Furthermore, γ separates s from t if and only if γ' separates s from t .*

Proof. The proof is analogous to the proof of Lemma 9 except that we consider distances along γ and γ' , which are cycles rather than paths. Due to this reason we slightly weaken some inequalities. The second claim of the lemma is immediate from the definition of the path equivalence. In the following we derive the first claim.

Assume that γ is not a β -stretch cycle. It follows that either γ contains a self-intersection, or there exists two points q and p on π , whose stretch factor is bigger than β . Formally, in either case, there exists a pair of points p and q in G_0 such that

$$\frac{d_\gamma(p, q)}{\|f_0(p) - f_0(q)\|} > \beta. \quad (19)$$

It is enough to consider the case, in which p is on $\pi_{i'}$ and q is on $\pi_{j'}$, and p and q are not contained in the union of 2 consecutive edges of γ . Indeed, these 2 consecutive edges would be also both on γ' by the definition of the equivalent paths, and the edges have length at most 2. Therefore the minimum length curve between p and q in γ is contained in these 2 consecutive edges.

We show that π' is not a $\beta(1 - 31\varepsilon)$ -stretch path. Consider the cell σ_q and σ_p in the radial grid $F_{v_1}(\varepsilon/100, \beta_0)$ and $F_{v_{j-1}}(\varepsilon/100, \beta_0)$, respectively, that contains p and q . We have $\varepsilon' = \frac{\varepsilon}{100\beta_0}$. The rest of the proof differs from the proof of Lemma 9 in the following weaker consequence of a variant of (6), and other inequalities with $d_{\pi'}(q', p')$ that needs to be replaced with $d_{\gamma'}(q', p')$.

$$d_\gamma(q, p) = \beta(L_{\sigma_q} + L_{\sigma_p}) + (1 + 100\varepsilon')d_{\gamma'}(q', p'), \quad (20)$$

where $f_0(q') \in \pi_{i'} \cap \sigma_q$ and $f_0(p') \in \pi_{j'} \cap \sigma_p$.

In the following we derive (20). Let $\pi = \pi(q, p) \subset \gamma$ such that $d_\pi(q, p) = d_\gamma(q, p)$. Let $\pi' = \pi'(q', p') \subset \gamma'$ such that $\pi' \cap \pi'_i \neq \emptyset$ if and only if $\pi \cap \pi_i \neq \emptyset$. Thus, π' is equivalent to π .

Let $\ell(\gamma)$ and $\ell(\gamma')$ denote the length of γ and γ' , respectively. If $d_{\pi'}(q', p') = d_{\gamma'}(q', p')$ then (20) holds by the same argument as in the proof of Lemma 9.

Otherwise, $d_{\gamma'}(q', p') = \ell(\gamma') - d_{\pi'}(q', p')$. Furthermore, $d_{\pi'}(q', p') = \beta(L_{\sigma_q} + L_{\sigma_p}) + (1 + \varepsilon')d_\gamma(q, p) \leq \beta(L_{\sigma_q} + L_{\sigma_p}) + \frac{1}{2}\ell(\gamma) \leq \beta(L_{\sigma_q} + L_{\sigma_p}) + \frac{1}{2}\ell(\gamma')(1 + \varepsilon')$. Combining the previous two (in)equalities we get that $d_{\gamma'}(q', p') \geq \ell(\gamma') - \beta(L_{\sigma_q} + L_{\sigma_p}) - \frac{1}{2}\ell(\gamma')(1 + \varepsilon') = \frac{1}{2}\ell(\gamma')(1 - \varepsilon') - \beta(L_{\sigma_q} + L_{\sigma_p})$.

7:20 Computing β -Stretch Paths in Drawings of Graphs

By the previous paragraph, and (7) and (8),

$$\frac{d_{\pi'}(q', p')}{d_{\gamma'}(q', p')} \leq \frac{\frac{1}{2}\ell(\gamma')(1 + \varepsilon') + \beta(L_{\sigma_q} + L_{\sigma_p})}{\frac{1}{2}\ell(\gamma')(1 - \varepsilon') - \beta(L_{\sigma_q} + L_{\sigma_p})} \leq \frac{\frac{1}{2}\ell(\gamma')(1 + \varepsilon') + 16\varepsilon'\ell(\gamma')}{\frac{1}{2}\ell(\gamma')(1 - \varepsilon') - 16\varepsilon'\ell(\gamma')} \leq \frac{1 + 33\varepsilon'}{1 - 33\varepsilon'} \quad (21)$$

Now, (20) follows from (6) using (21) for sufficiently small ε' . ◀

Submodular Clustering in Low Dimensions

Arturs Backurs

Toyota Technological Institute at Chicago, IL, USA

Sariel Har-Peled

University of Illinois at Urbana-Champaign, IL, USA

Abstract

We study a clustering problem where the goal is to maximize the coverage of the input points by k chosen centers. Specifically, given a set of n points $P \subseteq \mathbb{R}^d$, the goal is to pick k centers $C \subseteq \mathbb{R}^d$ that maximize the service $\sum_{p \in P} \varphi(\mathbf{d}(p, C))$ to the points P , where $\mathbf{d}(p, C)$ is the distance of p to its nearest center in C , and φ is a non-increasing service function $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. This includes problems of placing k base stations as to maximize the total bandwidth to the clients – indeed, the closer the client is to its nearest base station, the more data it can send/receive, and the target is to place k base stations so that the total bandwidth is maximized. We provide an $n^{\varepsilon^{-O(d)}}$ time algorithm for this problem that achieves a $(1 - \varepsilon)$ -approximation. Notably, the runtime does not depend on the parameter k and it works for an arbitrary non-increasing service function $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering; Theory of computation \rightarrow Computational geometry

Keywords and phrases clustering, covering, PTAS

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.8

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF awards CCF-1907400 and CCF-1421231.

1 Introduction

Clustering is a fundamental problem, used almost everywhere in computing. It involves partitioning the data into groups of similar objects – and, under various disguises, it is the fundamental problem underlying most machine learning applications. The (theoretically) well studied variants include k -median, k -means and k -center clustering. But many other variants of the clustering problem have been subject of a long line of research [10].

A clustering problem is often formalized as a constrained minimization problem of a cost function. The cost function captures the similarity of the objects in the same cluster. By minimizing the cost function we obtain a clustering of the data such that objects in the same cluster are more similar (in some sense) to each other than to those in other clusters. Many of this type of formalizations of clustering are both computationally hard, and sensitive to noise – often because the number of clusters is a hard constraint.

Clustering as a quality of service maximization

An alternative formalization of the clustering problem is as a *maximization* problem where the goal is to maximize the quality of “service” the data gets from the facilities chosen. As a concrete example, consider a set of n clients, and the problem is building k facilities. The quality of service a client gets is some monotonically decreasing non-negative function of its distance to the closest facility. As a concrete example, for a mobile client, this quantity might be the bandwidth available to the client. We refer to this problem as the *k-service* problem.

Such a formalization of clustering has several advantages. The first is diminishing returns (a.k.a. submodularity) – that is, the marginal value of a facility decreases as one adds more facilities. This readily leads to an easy constant approximation algorithm. A second



© Arturs Backurs and Sariel Har-Peled;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 8; pp. 8:1–8:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

significant advantage is the insensitivity to outliers – a few points being far away from the chosen facilities are going to change the target function by an insignificant amount (naturally, these outliers would get little to no service).

Formal problem statement: k -service

Given a set $P \subseteq \mathbb{R}^d$ of n points, a monotonically decreasing function $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, the goal is to choose a set C of k centers (not necessarily among the n given points), that maximize $\sum_{p \in P} \varphi(\mathbf{d}(p, C))$, where $\mathbf{d}(p, C) = \min_{c \in C} \|p - c\|$.

Our result

We obtain an $n^{\varepsilon^{-O(d)}}$ time algorithm for this problem that achieves $(1 - \varepsilon)$ -approximation for points in \mathbb{R}^d .

Related work

Maximum coverage problems, such as partial coverage by disks, were studied in the past [15]. These problems can be interpreted as a k -service problem, where the function is 1 within distance r from a facility, and zero otherwise. In particular, Chaplick *et al.* [3] showed a PTAS for covering a maximum number of points, out of a given set of disks in the plane. Our result implies a similar result in higher dimensions, except that we consider the continuous case (i.e., our results yields a PTAS for covering the maximum number of points using k unit disks). Cohen-Addad *et al.* [7] showed that local search leads to a PTAS for k -median and k -means clustering in low dimensions (and also in minor-free graphs). In [5] it was shown that the local search for k -means can be made faster achieving the runtime of $n \cdot k \cdot (\log n)^{(d/\varepsilon)^{O(d)}}$. In [6] near-linear time approximation schemes were obtained for several clustering problems improving on an earlier work (in particular, [12]). The authors achieve the runtime of $2^{(1/\varepsilon)^{O(d^2)}} n(\log n)^{O(1)}$. The techniques from the above works do not seem to be able to give near-linear time solution for the k -service problem, unfortunately. For instance, consider the special case of the k -service problem with $k = 1$ and where the service function is 1 within distance r from a facility, and zero otherwise (the maximum coverage problem as above). Even for this very special case of the problem there is no algorithm known running in time $n^{o(d)}$ where d is the dimension of the underlying space. The special case of $k = 1$ is a significant obstacle towards obtaining near-linear time algorithms for the k -service problem.

Another related line of work is on the kernel density estimation problem where a set P of n points is given and the goal is to preprocess P such that for an arbitrary query point c one can efficiently approximate $\sum_{p \in P} \varphi(\mathbf{d}(p, c))$. The goal is to answer such queries much faster than in $O(nd)$ time, which is just the linear scan. For various service functions φ and distance functions \mathbf{d} significantly faster algorithms are known [14, 4, 1]. Despite the similarity, however, finding a point (center) c that (approximately) maximizes the sum $\sum_{p \in P} \varphi(\mathbf{d}(p, c))$ seems to be a much harder problem [13].¹ Our work can be seen as a generalization of the latter problem where our goal is to pick k centers instead of one center.

In another work, Friggstad *et al.* [11] addressed the clustering problem in the setting with outliers.

¹ In particular, to find such a point, [13] use a heuristic that iteratively computes the gradient (mean shift vector) to obtain a sequence of points that converge to a local maxima (mode) of the density.

Balanced divisions

One of the building blocks we need is balanced divisions for Voronoi diagrams. This is present in the recent paper of Cohen-Addad *et al.* [7]. The idea of balanced divisions seems to go back to the work of Cohen-Addad and Mathieu [8]. Chaplick *et al.* [3] also prove a similar statement for planar graphs.

For the sake of completeness, we include the proof of the desired balanced divisions we need in Appendix A. Both here and [7] uses the Voronoi separator of Bhattiprolu and Har-Peled [2] as the starting point to construct the desired divisions. The Voronoi divisions we construct here are slightly stronger than the one present in [7] – all the batches in the division are approximately of the same size, and each one has a small separator from the rest of the point set.

Clustering and submodularity

Work using submodularity in clustering includes Nagano *et al.* [16] and Wei *et al.* [17]. Nagano *et al.* [16] considers the problem of computing the multi-way cut, that minimizes the average cost (i.e., number of edges in the cut divided by the number of clusters in the cut). Wei *et al.* [17] also studies such partitions with average cost target function. These works do not have any direct connection to what is presented here, beyond the usage of submodularity.

Paper organization

We define the problem formally in Section 2, and review some necessary tools including submodularity and balanced subdivisions. Section 3 describes how to find a good exchange for the current solution. We describe the local search algorithm in Section 4. The main challenge is to prove that if the local search did not reach a good approximation, then there must be a good exchange that improves the solution – this is proved in Section 5.

2 Preliminaries

Notations

In the following, we use $X + x$ and $X - x$ as a shorthand for $X \cup \{x\}$ and $X \setminus \{x\}$, respectively.

Given a point $p \in \mathbb{R}^d$, and a set $D \subseteq \mathbb{R}^d$, we denote by $d(p, D) = \min_{f \in D} \|p - f\|$ the distance of p from D . A point in D realizing this distance is the *nearest-neighbor* to p in D , and is denoted by $\text{nn}(p, D) = \arg \min_{c \in D} \|c - p\|$.

2.1 Service function and problem statement

A service function is a monotonically non-increasing function $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. In the following, given $x \geq 0$, assume that one can compute, in constant time, both $\varphi(x)$ and $\varphi^{-1}(x)$. Given a point $p \in \mathbb{R}^d$, and a center $c \in \mathbb{R}^d$, the quality of service that c provides p is $\rho(c, p) = \varphi(\|p - c\|)$. For a set of centers C , the quality of service it provides to p is

$$\rho(C, p) = \max_{c \in C} \rho(c, p) = \rho(\text{nn}(p, C), p).$$

The service to P provided by the set of centers C , or just profit, is

$$\rho(C) = \rho(C, P) = \sum_{p \in P} \rho(C, p).$$

In the k -service problem, the task is to compute the set C^* of k points that realizes

$$\text{opt}_k(P) = \max_{C \subseteq \mathbb{R}^d, |C|=k} \rho(C, P).$$

2.2 Submodularity

The above is a submodular optimization problem. Indeed, consider a center point c , and a set of centers C . The cell of c , in the Voronoi partition induced by C , is

$$\text{cl}(c, C) = \{p \in P \mid \|c - p\| < \text{d}(p, C - c)\}.$$

► **Definition 1.** *The marginal value of c is*

$$\nabla(c, C) = \rho(C + c, P) - \rho(C - c, P) = \sum_{p \in \text{cl}(c, C+c)} (\rho(C + c, p) - \rho(C - c, p)).$$

In words, this is the increase in the service that one gets from adding the center c .

For two sets of centers $D \subseteq C$, and a center c , observe that $\text{cl}(c, C + c) \subseteq \text{cl}(c, D + c)$. In particular, we have

$$\begin{aligned} \nabla(c, D) &= \sum_{p \in \text{cl}(c, D+c)} (\rho(c, p) - \rho(D, p)) \geq \sum_{p \in \text{cl}(c, C+c)} (\rho(c, p) - \rho(D, p)) \\ &\geq \sum_{p \in \text{cl}(c, C+c)} (\rho(c, p) - \rho(C, p)) = \nabla(c, C). \end{aligned}$$

This property is known as submodularity.

2.3 Balanced divisions

For a point set $P \subseteq \mathbb{R}^d$, the Voronoi diagram of P , denoted by $\mathcal{V}(P)$ is the partition of space into convex cells, where the Voronoi cell of $p \in P$ is

$$\mathcal{C}_P(p) = \{q \in \mathbb{R}^d \mid \|q - p\| \leq \text{d}(q, P - p)\},$$

where $\text{d}(q, P) = \min_{t \in P} \|q - t\|$ is the distance of q to the set P , see [9] for more details on Voronoi diagrams.

► **Definition 2.** *Let P be a set of points in \mathbb{R}^d , and P_1 and P_2 be two disjoint subsets of P . The sets P_1 and P_2 are Voronoi separated in P if for all $p_1 \in P_1$ and $p_2 \in P_2$, we have that their Voronoi cells are disjoint – that is, $\mathcal{C}_P(p_1) \cap \mathcal{C}_P(p_2) = \emptyset$. That is, the Voronoi cells of the pointsets are non-adjacent.*

► **Definition 3.** *Given a set P of n points in \mathbb{R}^d , a set of pairs $\{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$ is a Voronoi α -division of P , if for all i , we have*

- (i) B_1, \dots, B_m are disjoint,
- (ii) $\bigcup_j B_j = P$,
- (iii) pointset ∂_i Voronoi separates B_i from $P \setminus B_i$ in the Voronoi diagram of $P \cup \partial_i$ in the sense of Definition 2, and
- (iv) $|B_i| \leq \alpha$.

The set B_i is the i th batch, and ∂_i is its boundary.

A balanced coloring is a coloring $\chi : P \rightarrow \{-1, +1\}$ of P by ± 1 , such that $\chi(P) = \sum_{p \in P} \chi(p) = 0$. For a set $X \subseteq P$, its discrepancy is $|\chi(X)|$. We need the following balanced α -division result. Since this result is slightly stronger than what is available in the literature, and is not stated explicitly in the form we need it, we provide a proof for the sake of completeness in Appendix A.

► **Theorem 4.** Given a set P of n points in \mathbb{R}^d , parameters $\delta \in (0, 1)$ and $\alpha = \Omega(1/\delta^{d+1})$, and a balanced coloring χ of P , one can compute in polynomial time, a Voronoi α -division $\mathcal{D} = \{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$, such that the following holds:

(A) $\bigcup B_i = P$, and the batches B_1, \dots, B_m are disjoint.

(B) $m = O(n/\alpha)$.

(C) For all i , we have the following properties:

(C.i) the set ∂_i Voronoi separates B_i from $P \setminus B_i$.

(C.ii) $(1 - \delta)\alpha \leq |B_i| \leq \alpha$ (except for the last batch, which might be of size at least $(1 - \delta)\alpha$, and at most size 2α).

(C.iii) $|\partial_i| \leq \delta |B_i|$.

(C.iv) $|\chi(B_i)| \leq \delta |B_i|$.

3 Computing a good local exchange (if it exists)

► **Lemma 5** (Computing a good single center). Let P be a set of n points in \mathbb{R}^d , and let C be a set of k centers. Given a parameter $\varepsilon \in (0, 1)$, one can $(1 - \varepsilon)$ -approximate the center $c \in \mathbb{R}^d$ that maximizes the marginal value in $(n/\varepsilon)^{O(d)}$ time. Formally, we have $\nabla(c, C) \geq (1 - \varepsilon) \max_{f \in \mathbb{R}^d \setminus C} \nabla(f, C)$.

Proof. Let $\rho = \rho(C, P)$, $g = \arg \max_{f \in P} \nabla(f, C)$, $\Delta = \nabla(g, C)$, and $\mathbf{u} = \varphi(0)$. Clearly, the profit of the optimal solution, after adding any number of centers to C (but at least one), is somewhere in the interval $[\rho + \Delta, n\mathbf{u}] \subseteq [\rho + \Delta, \rho + n\Delta]$, which follows from $\mathbf{u} \leq (\rho/n) + \Delta$. For a point $p \in P$, let

$$v(p, i) = \min\left(\rho(C, p) + \ell(i), \mathbf{u}\right) \quad \text{where } \ell(i) = (1 + \varepsilon/4)^i \frac{\varepsilon \Delta}{4n},$$

for $i = 0, \dots, N$, where $N = \lceil 16(\ln n)/\varepsilon^2 \rceil$. Let $r(p, i) = \varphi^{-1}(v(p, i))$ (this is the radius from p where a center provides service $v(p, i)$).

Place a sphere of radius $r(p, i)$ around each point $p \in P$, for $i = 0, \dots, N$. Let \mathcal{F} be the resulting set of spheres. Compute the arrangement $\mathcal{A}(\mathcal{F})$, and place a point inside each face of this arrangement. Let Q be the resulting set of points. Compute the point $c \in Q$ realizing $\max_{f \in Q} \nabla(f, C)$, and return it as the desired new center.

To show the correctness, consider the (open) face F of $\mathcal{A}(C)$, that contains c^* , where c^* is the optimal center to be added. Let f be any point of Q in F . Let

$$\nabla(f, p) = \rho(f \cup C, p) - \rho(C, p).$$

Define $\nabla(c^*, p)$ similarly. Clearly, we have $\nabla(f, C) = \sum_{p \in P} \nabla(f, p)$. Let P_1 be all the points p of P such that $\nabla(c^*, p) \leq \nabla(f, p) + \varepsilon \Delta / (4n)$. Similarly, let P_2 be all the points p of P , such that

$$\nabla(c^*, p) > \nabla(f, p) + \varepsilon \Delta / (4n).$$

For any point $p \in P_2$, by the choice of N , there exists an index i , such that $\ell(i) \leq \nabla(c^*, p) < \ell(i + 1)$. By the choice of f from the arrangement, we have that $\nabla(f, p) \geq \ell(i)$, which in turn implies that

$$\nabla(f, p) \leq \nabla(c^*, p) < (1 + \varepsilon/4)\nabla(f, p) \implies \nabla(f, p) \geq (1 - \varepsilon/2)\nabla(c^*, p).$$

We thus have the following

$$\nabla(f, C) = \sum_{p \in P} \nabla(f, p) = \sum_{p \in P_1} \nabla(f, p) + \sum_{p \in P_2} \nabla(f, p)$$

8:6 Submodular Clustering in Low Dimensions

$$\begin{aligned}
&\geq \sum_{p \in P_1} (\nabla(c^*, p) - \frac{\varepsilon \Delta}{4n}) + \sum_{p \in P_2} (1 - \varepsilon/2) \nabla(c^*, p) \\
&\geq (1 - \varepsilon/2) \sum_{p \in P} \nabla(c^*, p) - \frac{\varepsilon \Delta}{4} \geq (1 - \varepsilon) \nabla(c^*, C).
\end{aligned}$$

The runtime follows from the observation that the number of faces in the arrangement is $(n/\varepsilon)^{O(d)}$. ◀

► **Lemma 6.** *Given a set P of n points in the plane, and a parameter k , one can compute in polynomial time (i.e., $n^{O(d)}$) a constant approximation to $\rho_{\mathcal{O}} = \text{opt}_k(P)$.*

Proof. Follows by using Lemma 5 in a greedy fashion k times, with $\varepsilon = 0.1$, to get a set of k centers. The quality of approximation readily follows from known results about submodularity [18]. Indeed, let $v_i = \rho(C_i, P)$ be the service provided by the first i centers computed. By submodularity, and the quality guarantee of Lemma 5, we have that $\nabla(c_i, C_{i-1}) \geq (1 - \varepsilon)(\rho_{\mathcal{O}} - v_{i-1})/k$. In particular, setting $\Delta_0 = \rho_{\mathcal{O}}$, and $\Delta_i = \rho_{\mathcal{O}} - v_{i-1}$, we have that $\Delta_i \leq (1 - (1 - \varepsilon)/k)\Delta_{i-1}$. As such, $\Delta_k \leq \exp(-k(1 - \varepsilon)/k)\Delta_0 = \rho_{\mathcal{O}}/e^{\varepsilon-1} \leq \rho_{\mathcal{O}}/2$. Namely, we have $v_k \geq \rho_{\mathcal{O}}/2$, as desired. ◀

For two sets of points S and C we define $\nabla(S, C) = \rho(C + S, P) - \rho(C - S, P)$.

► **Lemma 7.** *Let P be a set of n points in \mathbb{R}^d , and let C be a set of k centers. Given an integer $t \geq 1$, a parameter $\varepsilon \in (0, 1)$, in $(n/\varepsilon)^{O(dt)}$ time, one can $(1 - \varepsilon)$ -approximate the set $S \subset \mathbb{R}^d$ with $|S| = t$ that maximizes the marginal value in $(n/\varepsilon)^{O(td)}$ time. Formally, we have $\nabla(S, C) \geq (1 - \varepsilon) \max_{F \subset \mathbb{R}^d, |F|=t} \nabla(F, C)$.*

Proof. Consider the optimal set F of size t . Denote it by S^* . Compute the same arrangement as in the proof of Lemma 5. Let F_1, \dots, F_t be the faces of $\mathcal{A}(C)$ that contain the t points of S^* . Pick an arbitrary point from each F_i and let S be the resulting point set of size t . Define

$$\nabla(S, p) = \rho(S \cup C, p) - \rho(C, p).$$

and similarly $\nabla(S^*, p)$.

As in the proof of Lemma 5, let P_1 be all the points of P such that $\nabla(S^*, p) \leq \nabla(S, p) + \varepsilon \Delta/4n$ and P_2 be all the points of P such that

$$\nabla(S^*, p) > \nabla(S, p) + \varepsilon \Delta/4n.$$

We also conclude that $\nabla(S, p) \geq (1 - \varepsilon/2)\nabla(S^*, p)$ for all $p \in P_2$. We get

$$\begin{aligned}
\nabla(S, C) &= \sum_{p \in P} \nabla(S, p) = \sum_{p \in P_1} \nabla(S, p) + \sum_{p \in P_2} \nabla(S, p) \\
&\geq \sum_{p \in P_1} (\nabla(S^*, p) - \frac{\varepsilon \Delta}{4n}) + \sum_{p \in P_2} (1 - \varepsilon/2) \nabla(S^*, p) \\
&\geq (1 - \varepsilon/2) \sum_{p \in P} \nabla(S^*, p) - \frac{\varepsilon \Delta}{4} \geq (1 - \varepsilon) \nabla(S^*, C).
\end{aligned}$$

The runtime follows from the observation that the number of faces in the arrangement is $(n/\varepsilon)^{O(d)}$ and that it is sufficient to consider subsets of size t of the faces. ◀

4 The local search algorithm

The algorithm starts with a constant approximation, using the algorithm of Lemma 6. Next, the algorithm performs local exchanges, as long as it can find a local exchange that is sufficiently profitable.

Specifically, let $\alpha = O(1/\varepsilon^d)$, and let $\xi = O(\alpha/\varepsilon) = O(1/\varepsilon^{d+1})$. Assume that one can “quickly” check given a set of k centers C , whether there is a local exchange of size ξ , such that the resulting set of centers provides service $(1 + \varepsilon^2/(16k))\rho_{\text{curr}}$, where ρ_{curr} is the service of the current solution. To this end, the algorithm considers at most k^ξ possible subsets of the current set of centers that might be dropped, and for each such subset, one can apply Lemma 7, to compute (approximately) the best possible centers to add. If all such subsets do not provide an improvement, the algorithm stops.

Running time analysis

The algorithm starts with a constant approximation. As such, there could be at most $O(k/\varepsilon^2)$ local exchanges before the algorithm must terminate. Finding a single such exchange requires applying Lemma 7 k^ξ times. Lemma 7 is invoked with $t = \xi$. The resulting running time is $k^\xi(n/\varepsilon)^{O(d\xi)} = (n/\varepsilon)^{O(d/\varepsilon^{d+1})}$, where we remember that $k \leq n$.

5 Correctness of the local search algorithm

Here we show that if the local algorithm has reached a local optimum, then it reached a solution that is a good approximation to the optimal solution.

► **Remark 8.** In the following, we simplify the analysis at some points, by assuming that the local solution takes an exchange if it provides any improvement (the algorithm, however, takes an exchange only if it is a significant improvement). Getting rid of the assumption and modifying the analysis is straightforward, but tedious.

5.1 Notations

Let L and O be the local and optimal set of k centers. Let $\mathcal{U} = L \cup O$. Assign a point of \mathcal{U} color $+1$ if it is in O and -1 if it is in L (for the sake of simplicity of exposition assume no point belong to both sets). Let γ be a sufficiently large constant. For $\delta = \varepsilon/\gamma$ and $\alpha = O(1/\delta^{d+1})$, compute a α -division $\mathcal{D} = \{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$ of $L \cup O$, using Theorem 4.

Let $L_i = B_i \cap L$, $\bar{L}_i = L_i \cup \partial_i$, $O_i = B_i \cap O$, $\bar{O}_i = O_i \cup \partial_i$, $\ell_i = |L_i|$, and $\phi_i = |O_i|$, for all i . Let $\bar{L} = \bigcup_i \bar{L}_i$, and $\bar{O} = \bigcup_i \bar{O}_i$. Let $\partial = \bigcup_i \partial_i$. By construction, we have that $\sum_i |\partial_i| \leq \delta 2k \leq \varepsilon k/4$ if γ is a sufficiently large constant.

5.2 Submodularity implies slow degradation

The following is a well known implication of submodularity. We include the proof for the sake of completeness.

► **Lemma 9.** *Let C be a set of k centers. Then, for any $t \leq k$, there exists a subset $C' \subseteq C$ of size t , such that $\rho(C') \geq \frac{t}{k}\rho(C)$, where $\rho(C) = \rho(C, P)$.*

Proof. Let $C_0 = C$. In the i th iteration, we greedily remove the point of C_{i-1} that is minimizing the marginal value. Formally,

$$f_i = \arg \min_{c \in C_{i-1}} \nabla(c, C_{i-1} - c),$$

8:8 Submodular Clustering in Low Dimensions

and $C_i = C_{i-1} - f_i$. By submodularity, we have that $\nabla(f_i, C_{i-1} - f_i) \leq \rho(C_{i-1})/|C_{i-1}|$. As such, we have

$$\begin{aligned} \rho(C_i) &= \rho(C_{i-1}) - \nabla(f_i, C_{i-1} - f_i) \geq \left(1 - \frac{1}{k-i+1}\right) \rho(C_{i-1}) = \frac{k-i}{k-i+1} \rho(C_{i-1}) \\ &\geq \frac{k-i}{k-i+1} \cdot \frac{k-i+1}{k-i+1+1} \cdots \frac{k-1}{k} \rho(C_0) = \frac{k-i}{k} \rho(C). \end{aligned}$$

The claim now readily follows by taking the set C_{k-t} . ◀

5.3 Boundary vertices are not profitable

First, we argue that adding the boundary points, does not increase the profit/service significantly, for either the local or optimal solutions.

► **Lemma 10.** $\rho(\bar{L}) \leq (1 + \varepsilon/4)\rho(L)$ and $\rho(\bar{O}) \leq (1 + \varepsilon/4)\rho(O)$.

Proof. Let $p = |\partial|$. Consider a point $c \in \partial$, and observe that $\nabla(c, L) \leq \rho(L)/k$. This is a standard consequence of submodularity and greediness/local optimality. To see that, order the centers of $L = \{c_1, \dots, c_k\}$ in an arbitrary order. Let $\nabla_i = \nabla(c_i, \{c_1, \dots, c_{i-1}\}) \geq 0$, for $i = 1, \dots, k$, and observe that $\rho(L) = \sum_{i=1}^k \nabla_i$. As such, there exists an index i , such that $\nabla_i \leq \rho(L)/k$. By submodularity, we have that $\nabla(c_i, L - c_i) \leq \nabla(c_i, \{c_1, \dots, c_{i-1}\}) = \nabla_i \leq \rho(L)/k$, and $\nabla(c, L - c_i) \geq \nabla(c, L)$. Assume, for the sake of contradiction, that $\nabla(c, L) > \rho(L)/k$. We have that

$$\begin{aligned} \rho(L - c_i + c) &= \rho(L) - \nabla(c_i, L - c_i) + \nabla(c, L - c_i) \geq \rho(L) - \frac{\rho(L)}{k} + \nabla(c, L) \\ &> \rho(L) - \frac{\rho(L)}{k} + \frac{\rho(L)}{k} = \rho(L). \end{aligned}$$

But the local search algorithm considered this swap, which means that $L - c_i + c$ can not be more profitable than the local solution. A contradiction (see Remark 8).

Setting $\partial = \{f_1, \dots, f_p\}$, we have

$$\begin{aligned} \rho(\bar{L}) &= \rho(L) + \sum_{i=1}^p \nabla(f_i, L + f_1 + \dots + f_{i-1}) \leq \rho(L) + \sum_{i=1}^p \nabla(f_i, L) \leq \\ &\rho(L) + p\rho(L)/k \leq (1 + \varepsilon/4)\rho(L), \end{aligned}$$

since $p = |\partial| \leq \varepsilon k/4$.

The second claim follows by a similar argument. ◀

5.4 If there is a gap, then there is a swap

The contribution of the clusters L_i and O_i is

$$\nabla L_i = \nabla(L_i, \bar{L} \setminus L_i). \quad \text{and} \quad \nabla O_i = \nabla(O_i, \bar{O} \setminus O_i), \quad (5.1)$$

respectively. Notice that, because of the separation property, the points in P that their coverage change when we move from $\bar{L} \setminus L_i$ to \bar{L} , are points that are served by $\partial_i \subseteq \bar{L} \setminus L_i$ (same holds for $\bar{O} \setminus O_i$ and \bar{O}).

This implies that

$$\nabla(L, \partial) = \sum_i \nabla L_i \quad \text{and} \quad \nabla(O, \partial) = \sum_i \nabla O_i.$$

In the following, we assume that $\rho(\mathbf{L}) < (1 - \varepsilon)\rho(\mathbf{O})$. By Lemma 10 this implies that $\rho(\bar{\mathbf{L}}) \leq (1 + \varepsilon/4)\rho(\mathbf{L}) < (1 + \varepsilon/4)(1 - \varepsilon)\rho(\mathbf{O}) \leq (1 - \varepsilon/2)\rho(\mathbf{O}) \leq (1 - \varepsilon/2)\rho(\bar{\mathbf{O}})$. As such, we have

$$\begin{aligned} \rho(\bar{\mathbf{L}}) < (1 - \varepsilon/2)\rho(\bar{\mathbf{O}}) &\implies \rho(\partial) + \nabla(\mathbf{L}, \partial) < (1 - \varepsilon/2)(\rho(\partial) + \nabla(\mathbf{O}, \partial)) \\ &\implies \nabla(\mathbf{L}, \partial) < (1 - \varepsilon/2)\nabla(\mathbf{O}, \partial) - (\varepsilon/2)\rho(\partial) \\ &\implies \nabla(\mathbf{L}, \partial) < (1 - \varepsilon/2)\nabla(\mathbf{O}, \partial) \\ &\implies (\varepsilon/2)\nabla(\mathbf{O}, \partial) < \sum_i (\nabla\mathbf{O}_i - \nabla\mathbf{L}_i). \end{aligned}$$

By averaging, this implies that there exists an index t , such that

$$\nabla\mathbf{O}_t - \nabla\mathbf{L}_t > \frac{\varepsilon}{2k} \nabla(\mathbf{O}, \partial) > \frac{\varepsilon}{2k} (\nabla(\mathbf{O}, \partial) - \nabla(\mathbf{L}, \partial)) \quad (5.2)$$

$$= \frac{\varepsilon}{2k} (\rho(\bar{\mathbf{O}}) - \rho(\bar{\mathbf{L}})) \geq \frac{\varepsilon}{2k} \cdot \frac{\varepsilon}{2} \rho(\bar{\mathbf{O}}) \geq \frac{\varepsilon^2}{4k} \rho(\bar{\mathbf{O}}), \quad (5.3)$$

where in the second to last inequality we use that $\rho(\bar{\mathbf{L}}) < (1 - \varepsilon/2)\rho(\bar{\mathbf{O}})$. Namely, there is a batch where the local and optimal solution differ significantly.

5.4.1 An unlikely scenario

Assume that $|\mathbf{L}_t| \geq |\mathbf{O}_t| + |\partial_t|$. We then have that

$$\rho(\mathbf{L} + \partial_t - \mathbf{L}_t + \mathbf{O}_t) = \rho(\mathbf{L} + \partial_t) - \nabla\mathbf{L}_t + \nabla\mathbf{O}_t \geq \rho(\mathbf{L}) + \frac{\varepsilon^2}{4k} \rho(\bar{\mathbf{O}}).$$

But this is impossible, since the local search algorithm would have performed the exchange $\mathbf{L} + \partial_t - \mathbf{L}_t + \mathbf{O}_t$, since $|\partial_t| + |\mathbf{L}_t| + |\mathbf{O}_t|$ is smaller than the size of exchanges considered by the algorithm.

5.4.2 The general scenario

► **Lemma 11.** *There exists a subset $Y \subseteq \mathbf{O}_t$, such that $|\mathbf{L}_t| \geq |Y| + |\partial_t|$, and*

$$\nabla(Y, \bar{\mathbf{O}} \setminus \mathbf{O}_t) \geq \nabla\mathbf{L}_t + \frac{\varepsilon^2}{8k} \rho(\bar{\mathbf{O}}),$$

see Eq. (5.1).

Proof. We have that $(\varepsilon/2)\nabla(\mathbf{O}, \partial) < \sum_i (\nabla\mathbf{O}_i - \nabla\mathbf{L}_i)$. Subtracting $(\varepsilon/8)\nabla(\mathbf{O}, \partial)$ from both sides implies that $(\varepsilon/4)\nabla(\mathbf{O}, \partial) < \sum_i ((1 - \varepsilon/8)\nabla\mathbf{O}_i - \nabla\mathbf{L}_i)$. This in turn implies that there exists t such that $(1 - \varepsilon/8)\nabla\mathbf{O}_t - \nabla\mathbf{L}_t > (\varepsilon/(4k))\nabla(\mathbf{O}, \partial)$. Arguing, as above, we have that $(1 - \varepsilon/8)\nabla\mathbf{O}_t - \nabla\mathbf{L}_t > (\varepsilon^2/(8k))\rho(\bar{\mathbf{O}})$.

Consider the following (submodular) function

$$f(X) = \nabla(X, \bar{\mathbf{O}} \setminus \mathbf{O}_t),$$

By Lemma 9 (or more precisely arguing as in this lemma), we have that there exists a set $Y \subset \mathbf{O}_t$, such that $|Y| = (1 - \varepsilon/8)|\mathbf{O}_t|$ and $f(Y) \geq (1 - \varepsilon/8)f(\mathbf{O}_t) = (1 - \varepsilon/8)\nabla\mathbf{O}_t$. As such, we have that

$$\nabla(Y, \bar{\mathbf{O}} \setminus \mathbf{O}_t) \geq (1 - \varepsilon/8)\nabla\mathbf{O}_t \geq \nabla\mathbf{L}_t + (\varepsilon^2/8k)\rho(\bar{\mathbf{O}}).$$

8:10 Submodular Clustering in Low Dimensions

As for the size of Y . Observe that by Theorem 4, we have $|\partial_i| \leq \frac{\varepsilon}{\gamma}(|\mathbf{O}_i| + |\mathbf{L}_i|)$ and $||\mathbf{O}_i| - |\mathbf{L}_i|| \leq \frac{\varepsilon}{\gamma}(|\mathbf{O}_i| + |\mathbf{L}_i|)$. This readily implies that $|\mathbf{O}_i| \leq (1 + 4\varepsilon/\gamma)|\mathbf{L}_i|$, and $|\mathbf{L}_i| \leq (1 + 4\varepsilon/\gamma)|\mathbf{O}_i|$, if γ is sufficiently large. As such, we have that

$$\begin{aligned} |\mathbf{L}_t| &\geq \frac{|\mathbf{O}_t|}{1 + 4\varepsilon/\gamma} \geq (1 - 4\varepsilon/\gamma)|\mathbf{O}_t| = (1 - \varepsilon/8)|\mathbf{O}_t| + (\varepsilon/8 - 4\varepsilon/\gamma)|\mathbf{O}_t| \\ &\geq |Y| + \frac{\varepsilon}{16}|\mathbf{O}_t| \geq |Y| + |\partial_i|, \end{aligned}$$

if $\gamma \geq 64$. ◀

► **Lemma 12.** *The local search algorithm computes a $(1 - \varepsilon)$ -approximation to the optimal solution.*

Proof. If not, then, arguing as above, there must be a batch for which there is an exchange with profit at least $(\varepsilon^2/4k)\rho(\bar{\mathbf{O}})$ (see Eq. (5.3)). By Lemma 11, we can shrink the optimal batch \mathbf{O}_t , such that the exchange becomes feasible, and is still profitable (the profit becomes $(\varepsilon^2/8k)\rho(\bar{\mathbf{O}})$). But that is impossible, since by arguing as above (i.e., the unlikely scenario), we have that this swap would result in a better local solution, and the exchange is sufficiently small to have been considered. Specifically, the local search algorithm uses Lemma 7, say with $\varepsilon = 1/2$, ensures that the local search algorithm would find an exchange with half this value, and would take it. A contradiction. ◀

5.5 The result

► **Theorem 13.** *Let P be a set of n points in \mathbb{R}^d , let $\varepsilon \in (0, 1)$ be a parameter, let $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ be a service function, and let $k \leq n$ be an integer parameter. One can compute, in $(n/\varepsilon)^{O(d/\varepsilon^{d+1})}$ time, a set of k centers C , such that $\rho(C, P) \geq (1 - \varepsilon)\text{opt}_k(P)$, where $\text{opt}_k(P)$ denotes the optimal solution using k centers.*

6 Discussion

We presented an algorithm that runs in polynomial time for any constant $\varepsilon > 0$ and any constant dimension d and achieves a $(1 - \varepsilon)$ -approximation. The dependency on the dimension d is doubly exponential, however. A natural question is whether the dependency on the dimension d in the runtime can be improved. Perhaps by considering some special cases of the problem, for more specific service function, such as $\varphi(\ell) = \frac{1}{1+\ell}$ or $\varphi(\ell) = \frac{1}{1+\ell^2}$ (i.e., the service quality drops roughly linearly or quadratically with the distance).

Our algorithm finds a subset $C \subseteq \mathbb{R}^d$ of size k that approximately maximizes the objective function. A close variant of the problem asks to find a subset $C \subseteq \mathbb{R}^d$ of size k that has an additional constraint that $C \subseteq P$. Can we obtain an algorithm for this problem with the same asymptotic runtime for $d > 2$? The main difficulty is that we would need a variant of Theorem 4 with an additional property $\partial_i \subseteq P$ for all i but such a division does not exist even for $d = 3$. (For $d = 2$ using planar graph divisions on the Voronoi diagram directly implies the desired result.)

Finally, one can ask a similar question about clustering in graphs. Specifically, given a graph on n vertices P , we would like to select k vertices C that approximately maximizes $\sum_{p \in P} \min_{c \in C} \varphi(\mathbf{d}(p, c))$, where $\mathbf{d}(p, c)$ is the shortest-path distance from p to c . Can we achieve a polynomial time algorithm for arbitrary small constant $\varepsilon > 0$ if the graph is planar or come from some other class of graphs?

References

- 1 Arturs Backurs, Moses Charikar, Piotr Indyk, and Paris Siminelakis. Efficient density evaluation for smooth kernels. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 615–626. IEEE, 2018.
- 2 Vijay V. S. P. Bhattiprolu and Sariel Har-Peled. Separating a Voronoi diagram via local search. In Sándor P. Fekete and Anna Lubiw, editors, *Proc. 32nd Int. Annu. Sympos. Comput. Geom. (SoCG)*, volume 51 of *LIPICs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.SoCG.2016.18.
- 3 Steven Chaplick, Minati De, Alexander Ravsky, and Joachim Spoerhase. Approximation schemes for geometric coverage problems. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proc. 27th Annu. Euro. Sympos. Alg. (ESA)*, volume 112 of *LIPICs*, pages 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.17.
- 4 Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1032–1043. IEEE, 2017.
- 5 Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–440. Society for Industrial and Applied Mathematics, 2018.
- 6 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-Linear Time Approximation Schemes for Clustering in Doubling Metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019.
- 7 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. *SIAM J. Comput.*, 48(2):644–667, 2019. doi:10.1137/17M112717X.
- 8 Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPICs*, pages 329–343. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPICs.SoCG.2015.329.
- 9 M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Santa Clara, CA, USA, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.
- 10 R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, New York, 2nd edition, 2001.
- 11 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R Salavatipour. Approximation schemes for clustering with outliers. *ACM Transactions on Algorithms (TALG)*, 15(2):26, 2019.
- 12 Zachary Friggstad, Mohsen Rezapour, and Mohammad R Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM Journal on Computing*, 48(2):452–480, 2019.
- 13 Bogdan Georgescu, Ilan Shimshoni, and Peter Meer. Mean shift based clustering in high dimensions: A texture classification example. In *ICCV*, volume 3, page 456, 2003.
- 14 Leslie Greengard and John Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.
- 15 Kai Jin, Jian Li, Haitao Wang, Bowei Zhang, and Ningye Zhang. Near-linear time approximation schemes for geometric maximum coverage. *Theoretical Computer Science*, 725:64–78, 2018. doi:10.1016/j.tcs.2017.11.026.
- 16 Kiyohito Nagano, Yoshinobu Kawahara, and Satoru Iwata. Minimum average cost clustering. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Neural Info. Proc. Sys. (NIPS)*, pages 1759–1767. Curran Associates, Inc., 2010. URL: <http://papers.nips.cc/paper/4106-minimum-average-cost-clustering>.

- 17 Kai Wei, Rishabh K. Iyer, Shengjie Wang, Wenruo Bai, and Jeff A. Bilmes. Mixed robust/average submodular partitioning: Fast algorithms, guarantees, and applications. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Neural Info. Proc. Sys. (NIPS)*, pages 2233–2241, 2015. URL: <http://papers.nips.cc/paper/5706-mixed-robustaverage-submodular-partitioning-fast-algorithms-guarantees-and-applications>.
- 18 L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982. doi:10.1007/BF02579435.

A Balanced Voronoi division

We need the following variant of a result of Bhattiprolu and Har-Peled [2].

► **Theorem 14.** *Let P be a set of n' points in \mathbb{R}^d , where every point has a positive integer weight, such that the total weight of the points is n , and let α be parameter. Furthermore, assume that no point has weight that exceeds α . Then, one can compute, in expected $O(n)$ time, a ball B , and a set Z that lies on the boundary of B , such that*

- (i) $|Z| \leq c_1 \alpha^{1-1/d}$,
- (ii) the total weight of the points of P inside B is at least α and at most $c_2 \alpha$,
- (iii) Z is a Voronoi separator of the points of P inside B from the points of P outside B .

Here $c_1, c_2 > 0$ are constants that depends only on the dimension d .

The points of the separator Z are guards.

A.1 A division using the above separator

A.1.1 Algorithm

We start with a set P of n points, and a parameter α . The idea is to repeatedly extract a set of weight (roughly) α from the point set, separate it, remove it, and put the set of guards associated with it back into the set.

To this end, let α be a parameter, such that

$$c_1 \alpha^{1-1/d} < \alpha/8 \iff 8c_1 < \alpha^{1/d} \iff \alpha > (8c_1)^d,$$

where c_1 is the constant from Theorem 14.

For an unweighted set of points X and a real number $\tau > 0$, let $\tau * X$ denote the set of points, where every points has weight τ .

The algorithm for constructing the division is the following:

1. $P_0 \leftarrow P$. Initially all the points in P_0 have weight 1.
2. $i \leftarrow 1$.
3. While P_{i-1} has total weight larger than α do:
 - 3.1 $(B_i, Z_i) \leftarrow$ ball and separator computed by Theorem 14 for P_{i-1} with parameter α .
 - 3.2 $I_i \leftarrow P_{i-1} \cap B_i$. // All points inside ball to be removed
 - 3.3 $G_i \leftarrow I_i \setminus P$. // The old guards in the ball
 - 3.4 $B_i = P \cap I_i$ // The batch of original points
 - 3.5 $P_i = (P_{i-1} \setminus I_i) \cup (\tau_i * Z_i)$, where $\tau_i = \lceil (\alpha/4)/|Z_i| \rceil$.
 - 3.6 $\partial_i = Z_i \cup G_i$ // The set of guards for the batch B_i
 - 3.7 $i \leftarrow i + 1$.
4. $m \leftarrow i$
5. $B_m = P_{m-1} \cap P$, and $\partial_m = P_{m-1} \setminus B_m$.
6. Return $\mathcal{D} = \{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$.

A.1.2 Analysis

► **Lemma 15.** *Consider the Voronoi diagram of $\mathcal{V}(P \cup \partial_i)$. There is no common boundary in this Voronoi diagram between a cell of a point of B_i and a cell of a point of $P \setminus B_i$.*

Proof. Consider a point p that is in equal distance to a point $f \in B_i$, and a point $g \in P \setminus B_i$, and furthermore, all other points of $P \setminus \{f, g\}$ are strictly further away from p .

The claim is that $d(p, \partial_i) < \|p - f\| = \|p - g\|$. Namely, the region of common boundary between f and g in $\mathcal{V}(P)$ is completely covered by cells of ∂_i in $\mathcal{V}(P \cup \partial_i)$.

If $g \in P_i$, then Z_i separates (in the Voronoi interpretation) $f \in B_i \subseteq I_i$ from all the points of $P_i \cap P \ni g$, which implies the claim.

As such, it must be that $g \in B_j$, for some $j < i$. Namely, there is a guard $g_j \in Z_j$, that separates g from f , and its cell contains p . That is $\|p - g_j\| < \|p - f\|$, and $g_j \in P_j$. If $g_j \in \partial_i$ then the claim holds.

Otherwise, we apply the same argument again, this time to g_j and f . Indeed, g_j was removed (from P_k) in some iteration k , such that $j < k < i$. Namely $g_j \in k$, and $f \notin k$. The point p is closer to g_j than to f . If $p \in k$ then there is a guard $g_k \in Z_k$ that is closer to p than f , by the separation property. Otherwise, it is easy to verify that the Voronoi cells of the guards of Z_k in $\mathcal{V}(P_{k-1} \cup Z_k)$ cover completely the portion of the Voronoi cells of points in $P_{k-1} \cap k$ outside k , in the Voronoi diagram $\mathcal{V}(P_{k-1})$. This readily implies that there is a closer guard $g_k \in Z_k$ to p than g_j . In either case, we continue the argument inductively on (g_k, f) .

By finiteness, it follows that there must be a guard $g' \in \partial_i$ that is closer to p than f , which implies the claim. ◀

► **Observation 16.** (A) For all i , we have $\tau_i = \left\lceil \frac{\alpha/4}{|Z_i|} \right\rceil \geq \frac{\alpha/4}{|Z_i|} \geq \frac{\alpha/4}{c_1 \alpha^{1-1/d}} \geq \frac{\alpha^{1/d}}{4c_1}$.

(B) As such, for all i , I_i contains at most $c_2 \alpha / \min_j \alpha_j = O(\alpha^{1-1/d})$ points that are not in P . That is, we have $|I_i \setminus B_i| = O(\alpha^{1-1/d})$.

(C) It follows that $|\partial_i| = |I_i \setminus B_i| + |Z_i| = O(\alpha^{1-1/d})$.

► **Lemma 17.** *Given a set P of n points in \mathbb{R}^d , and a parameter α , one can compute in polynomial time, a division $\mathcal{D} = \{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$, such that the following holds:*

(A) $\bigcup B_i = P$, and the clusters B_1, \dots, B_m are disjoint.

(B) $m = O(n/\alpha)$.

(C) For all i , we have the following properties:

(C.i) the set ∂_i separates B_i from $P \setminus B_i$.

(C.ii) $|B_i| = O(\alpha)$.

(C.iii) $|\partial_i| = O(\alpha^{1-1/d})$.

(D) For $\partial = \bigcup_i \partial_i$, we have that $|\partial| = O(n/\alpha^{1/d})$.

Furthermore, one can modify the above construction, so that Cii is replaced by $|B_i| = \Theta(\alpha)$.

Proof. For the bound on number of clusters, observe that every iteration of the algorithm reduces the weight of the working set P_i by at least $\alpha/2$ – indeed, the weight of B_i is at least α , and the total weight of points of Z_i (after multiplying their weight by τ_i) is at most $\alpha/2$. Thus implying the claim.

All the other claims are either proved above, or readily follows from the algorithm description.

The modification of Cii follows by observing that we can merge clusters, and there are $\Theta(n/\alpha)$ clusters with $\Omega(\alpha)$ points of P , by averaging. As such, one can merge $O(1)$ clusters

8:14 Submodular Clustering in Low Dimensions

that have $o(\alpha)$ points of P into a cluster that has $\Omega(\alpha)$ points of P , thus implying the modified claim. ◀

► **Theorem 4.** *Given a set P of n points in \mathbb{R}^d , parameters $\delta \in (0, 1)$ and $\alpha = \Omega(1/\delta^{d+1})$, and a balanced coloring χ of P , one can compute in polynomial time, a Voronoi α -division $\mathcal{D} = \{(B_1, \partial_1), \dots, (B_m, \partial_m)\}$, such that the following holds:*

- (A) $\bigcup B_i = P$, and the batches B_1, \dots, B_m are disjoint.
- (B) $m = O(n/\alpha)$.
- (C) For all i , we have the following properties:
 - (C.i) the set ∂_i Voronoi separates B_i from $P \setminus B_i$.
 - (C.ii) $(1 - \delta)\alpha \leq |B_i| \leq \alpha$ (except for the last batch, which might be of size at least $(1 - \delta)\alpha$, and at most size 2α).
 - (C.iii) $|\partial_i| \leq \delta |B_i|$.
 - (C.iv) $|\chi(B_i)| \leq \delta |B_i|$.

Proof. We compute a division of P using Lemma 17, with parameter $\alpha' = O(\delta\alpha) = \Omega(1/\delta^d)$, such that (i) the maximum size of a batch is strictly smaller than $\delta\alpha/2$, and (ii) $c_4(\alpha')^{1-1/d} < \delta\alpha'$, where c_4 is some prespecified constant. Let $\mathcal{D}' = \{(B'_1, \partial'_1), \dots, (B'_\tau, \partial'_\tau)\}$, be the resulting division. Let $\Delta_i = \chi(\partial'_i)$, and observe that $\sum_i \Delta_i = \chi(P) = 0$. There is a permutation π of the batches, such that for any prefix j , we have $|\sum_{i=1}^j \Delta_{\pi(i)}| \leq \max_i |B_i| \leq \delta\alpha/2 = D$. This follows readily by reordering the summation, such that one adds batches with positive (resp., negative) balance if the current prefix sum is negative (resp., positive), and repeating this till all the terms are used².

We break the permutation π into minimum number of consecutive intervals, such that total size of batches in each interval is at least $(1 - \delta)\alpha$. Merging the last two interval if needed to comply with the desired property. The batch formed by a union of an interval can have discrepancy at most $2D = 2(\delta\alpha/2) = \delta\alpha$, as desired.

All the other properties follows readily by observing that merging batches, results in valid batches, as far as separation. ◀

² This is the same idea that is used in the Riemann rearrangement theorem.

Kernelizing the Hitting Set Problem in Linear Sequential and Constant Parallel Time

Max Bannach

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
bannach@tcs.uni-luebeck.de

Malte Skambath

Department of Computer Science, Kiel University, Germany
malte.skambath@email.uni-kiel.de

Till Tantau

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

We analyze a reduction rule for computing kernels for the hitting set problem: In a hypergraph, the *link* of a set c of vertices consists of all edges that are supersets of c . We call such a set *critical* if its link has certain easy-to-check size properties. The rule states that the link of a critical c can be replaced by c . It is known that a simple linear-time algorithm for computing hitting set kernels (number of edges) at most k^d (k is the hitting set size, d is the maximum edge size) can be derived from this rule. We parallelize this algorithm and obtain the first AC^0 kernel algorithm that outputs polynomial-size kernels. Previously, such algorithms were not even known for artificial problems. An interesting application of our methods lies in traditional, non-parameterized approximation theory: Our results imply that uniform AC^0 -circuits can compute a hitting set whose size is polynomial in the size of an optimal hitting set.

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

Keywords and phrases Kernelization, Approximation, Hitting Set, Constant-Depth Circuits

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.9

1 Introduction

In the theory of fixed-parameter tractability, *kernelization algorithms* play an important role. They shrink input instances to membership-equivalent instances (called *kernels*) whose size depend only on the input's parameters. A rich theory has been developed around this idea, with results ranging from highly efficient kernel algorithms to lower bounds showing that some problems do not have polynomial-size kernels unless complexity class collapses occur [8]. Another recent direction is the question of how quickly kernels can be computed in parallel, and which trade-offs regarding kernel size are incurred by parallelisation [5, 6, 11].

In the present paper we are interested in computing kernels for the *hitting set problem* both, sequentially and in parallel. We study the following version of the problem on d -*hypergraphs*, which are pairs (V, E) consisting of a set V of *vertices* and a set E of *hyperedges* (which we will call just *edges*) such that for all edges $e \in E$ we have $e \subseteq V$ and $|e| \leq d$:

► **Problem 1.1** (p_k - d -HITTING-SET for fixed $d \in \mathbb{N}$).

Instances: A d -hypergraph $H = (V, E)$ and a *parameter* $k \in \mathbb{N}$.

Question: Is there a *size- k hitting set* $X \subseteq V$, that is, $|X| \leq k$ and $X \cap e \neq \emptyset$ for all $e \in E$?



© Max Bannach, Malte Skambath, and Till Tantau;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 9; pp. 9:1–9:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When d is not fixed, but part of the input, the resulting problem p_k -HITTING-SET is complete for $W[2]$ and, thus, no kernelization is possible unless $W[2] = \text{FPT}$ [13]. In contrast, p_k - d -HITTING-SET can be kernelized in polynomial time for every d . A standard way of doing so is based on the following reduction rule, where a *sunflower* is a set $S \subseteq E$ such that any two elements of S have the same intersection c_S , called the *core* of the sunflower:

► **Rule 1.2.** Find a sunflower S of size $|S| \geq k + 1$ and replace E by $(E \setminus S) \cup \{c_S\}$.

It is easy to see that this rule is safe and not-so-easy to see that it results in a kernel with at most $d!k^d$ edges (this follows from the Sunflower Lemma [15], by which a sunflower of size $k + 1$ always exists as long as $|E| > d!k^d$). However, finding sunflowers is a difficult problem in itself, requiring either expensive methods like color coding [3] or approximate solutions [18]. Therefore, the current state-of-the-art algorithm due to Fafianie and Kratsch uses the simpler and yet more powerful *critical core rule* [16]:

► **Rule 1.3.** Find a *critical core* $c \subseteq e \in E$ and replace E by $(E \setminus L_c) \cup \{c\}$.

The rule hinges on how we setup the definition of “critical” cores. This will depend on the computational model; in the simplest setup (see Definition 3.2 for the general case), a core c is critical if (1) for all supersets $c' \supsetneq c$ we have $|L_{c'}| \leq k^{d-|c'|}$ and (2) we have $|L_c| > k^{d-|c|}$. Checking this for a given core c is relatively easy, making the rule fairly easy to implement. While a bit of effort is needed to show that the rule is safe, it clearly yields a kernel with k^d edges¹: When it is no longer applicable, the empty set is no longer critical and by (2) we have $|E| = |L_\emptyset| \leq k^{d-0}$.

Less is known in the parallel setting. Only for $d = 2$ (the *vertex cover problem*) do we know anything concerning the parallel computation of polynomial-size kernels: TC^0 -circuits can compute quadratic kernels [14]. A complex argument shows that AC^0 -circuits can compute exponential-size kernels for the hitting set problem [6]. However, AC^0 -circuits were not known to be able to compute polynomial-size kernels for the vertex cover problem (nor, for that matter, for any other problem). For $d > 2$, the best parallel kernelization is the logspace algorithm due to Fafiane and Kratsch [16].

Our Contributions. We rephrase the kernelization algorithm from Fafianie and Kratsch [16] in a terminology that is more suited for changing the computational model on which the algorithm is implemented. As by-product, we obtain a slightly tighter bound on the kernel size: While the original paper achieves a kernel size of $(k + 1)^d$ edges, we obtain a kernel of size k^d . Furthermore, we can construct a kernel of size $\sum_{i=0}^d k^i$ that has the desirable property that it is a subsets of the original edge set. However, the main objective of this paper is a parallel constant-time implementation of the critical core rule:

Contribution I: First Polynomial-Size Constant-Time Kernels. By adjusting the setup when a core is considered critical, we are able to derive the first polynomial-size kernel that can be computed by AC^0 -circuits. It was previously known that *exponential*-size kernels for the hitting set problem can be computed by AC^0 -circuits, but *no* problem (not even an artificial one) was known for which AC^0 -circuits can compute a polynomial-size kernel.

There is more to the construction than just adjusting some thresholds: Any natural implementation of the critical core rule internally needs threshold gates, leading to TC^0 -circuits. To get down to AC^0 , we use *fuzzy* threshold gates, which behave like a normal threshold gate only when the number of input 1-bits is below the threshold or “well above” it. We show that the “fuzziness” of the gates is not a problem when computing kernels.

¹ In this paper, we are only interested in minimizing the number of edges and call it the kernel size. Reducing the number of vertices is also of interest, but not addressed by us. The set V is immutable.

Contribution II: Constant-Time Approximations. There are known connections between kernelization and approximation algorithms [1, 17, 21]. These connections carry over to AC^0 and we can derive an *approximation algorithm* for the hitting set problem that works in constant time: There is a family of AC^0 -circuits that on input of any d -hypergraph $H = (V, E)$ outputs a hitting set X of H such that if X^* is a minimum hitting set of H , then $|X| \in O(|X^*|^d)$. While clearly worse than the best approximation ratio (namely d) achieved sequentially, observe that AC^0 -circuits can only reliably “count up to polylogarithmic numbers,” but we must still compute an approximate hitting set when $|X^*| = n^\epsilon$.

Related Work. The “classical” way of computing kernels for p_k - d -HITTING-SET, due to Flum and Grohe [18, Section 9.1], is based on Erdős and Rado’s [15] Sunflower Lemma. Variations of this algorithm were proposed by van Bevern [24] and by Damaschke [12].

Attempts to parallelise parameterized algorithms date back to the late 1990s [9, 10]. A structural study of parameterized circuit complexity was started around 2015 by Elberfeld et al. [14]. The parameterized circuit model we use within this paper was introduced in [4]. It is known that a decidable problem is in $\text{para-}AC^i$ if, and only if, it has a kernel function in AC^i [6]. A first AC^0 kernelization for p_k - d -HITTING-SET was presented by Chen, Flum, and Huang [11], which was later improved to a AC^0 kernelization for $p_{k,d}$ -HITTING-SET (here, d is parameter and not a fixed constant) [5]. All of these kernels have exponential size.

Approximation algorithms based on kernelizations were studied by Abu-Khzam et al. [1], by Fellow et al. [17] as “fidelity kernels,” and by Lokshtanov et al. [21] as “lossy kernels.”

Structure of this Article. This article has four main sections: In Section 2 we explore the properties of “fuzzy” threshold gates. In Section 3 we review the critical core reduction rule and use it to compute polynomial-size kernels in linear or constant parallel time. In Section 4, we show how our results allow us to develop constant-depth approximation algorithms. We close the article with a look at the set packing problem in Section 5.

2 Technical Tools

On a technical level, we need a few standard notions, briefly reviewed in the following. The technical tool of *fuzzy threshold gates* is presented in more detail.

Circuits. We use standard notions of Boolean circuits: AC -circuits have n input gates and m output gates as well as AND-, OR-, and NOT-gates. The AND-gates and OR-gates may have unbounded fan-in. A TC-circuit may additionally have unbounded fan-in $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gates, which output 0 if at most t of its inputs are set to 1 and which outputs 1 otherwise. For a circuit C and a bitstring $x \in \{0, 1\}^n$ we write $C(x)$ for the length- m bitstring that is output by the circuit on input x .

A problem $L \subseteq \{0, 1\}^*$ is in AC^0 if there is a family $(C_n)_{n \in \mathbb{N}}$ of AC -circuits, where each C_n has n inputs and only one output, such that (1) we have $x \in L$ if, and only if, $C_{|x|}(x) = 1$ and (2) $\text{depth}(C_n) \leq c$ and $\text{size}(C_n) \leq n^c$ for some constant c . (Here, the size and depth functions for circuits can be defined in any sensible way.) We also require that the families are $DLOGTIME$ -uniform, which is the strongest form of uniformity commonly required [7]. The definition of the class TC^0 is analogue. In slight abuse of notation, we also use AC^0 and TC^0 to denote the functional classes (more correctly known as FAC^0 and FTC^0) containing functions $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ by allowing that the C_n have more than one output gate.

Parameterized Problems. A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$. It lies in the class FPT (also known as para-P), if on input (x, k) we can decide whether $(x, k) \in Q$ holds in time $f(k) \cdot |x|^c$ for some computable function f and some constant c . The problem lies in the class para-AC⁰ if there is a family $(C_{n,k})_{n,k \in \mathbb{N}}$ such that (1) we have $(x, k) \in Q$ if, and only if, $C_{|x|,k}(x) = 1$ and (2) we have $\text{depth}(C_{n,k}) \leq c$ and $\text{size}(C_{n,k}) \leq f(k) \cdot n^c$. We also require a DLOGTIME-uniformity condition, which in this context means that the i th bit of a suitable encoding of $C_{n,k}$ can be computed by a deterministic Turing machine that obtains i , n , and k encoded as binary numbers as input and that runs in time $f(k) + O(\log n + \log i)$. The class para-TC⁰ is defined in the same way by additionally allowing threshold gates.

Kernels. A *kernel function for a parameterized problem* Q is a mapping $K: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ such that for $K(x, k) =: (x', k')$ we have $(x, k) \in Q \Leftrightarrow (x', k') \in Q$, $|x'| \leq s(k)$, and $|k'| \leq \rho(k)$ for two computable functions s and ρ . The function s is called the *size (function)* of the kernel and we are particularly interested in the case that s is a polynomial.² The function ρ is less important and will always be the identity in this paper. It is well-known that for decidable problems Q we have $Q \in \text{para-P}$ if, and only if, Q has a kernel function $K \in \text{FP}$ (that is, computable in polynomial time) [18]; we have $Q \in \text{para-AC}^0$ if, and only if, Q has a kernel function K in the function class AC⁰ (where $(x, k) \in \Sigma^* \times \mathbb{N}$ is suitably encoded as a bitstring) [6]; and that an analogous result holds for para-TC⁰.

Most kernel algorithms are based on *reduction rules*: They take an input $(x, k) \in \Sigma^* \times \mathbb{N}$ and are either not applicable or output some (x', k') with $|x'| < |x|$ and $k' \leq k$. A rule is called *safe* for a problem Q if we always have $(x, k) \in Q \iff (x', k') \in Q$. A set of such rules *yields a kernel* if for any input (x, k) at least one rule is still applicable as long as $|x'| > s(k)$, that is, as long as the input has not yet shrunk to a kernel.

Kernel functions can map inputs (x, k) to some (x', k') that have “very little to do with the original (x, k) ” except for being membership-equivalent, while it is often desirable that kernels should preserve some of the properties of the input. When x is a hypergraph $H = (V, E)$ and the objective is to find a set $X \subseteq V$ with certain properties, we say that a kernel function K *preserves solutions* if (x, k) and $K(x, k)$ always have the same solutions, and it *preserves edges* if the edges in $K(x, k)$ constitute a subset of E . The *full kernels* of Damaschke [12] preserve minimal solutions in our sense and the *explanatory kernels* of van Bevern [24] are edge-preserving kernels in our sense. An edge-preserving kernel does not need to be solution-preserving, but this will always be the case in the present paper.

Fuzzy Thresholds. It is well-known [23] that $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gates can be simulated by AC⁰-circuits for polylogarithmic t , that is, for each exponent c we have $\{1^t 0^b \mid t \in \mathbb{N}, b \in \{0, 1\}^*, \sum_{i=1}^{|b|} b[i] \leq t \leq \log_2^c |b|\} \in \text{AC}^0$ and TC⁰-families using only polylogarithmic thresholds can be replaced by equivalent AC⁰-families. However, $\text{MAJORITY} \in \text{TC}^0 \setminus \text{AC}^0$ shows that for $t = \lfloor n/2 \rfloor$ an AC⁰-circuit simulating a $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gate would need superpolynomial size [19]. To step beyond the “polylogarithmic boundary” we use *fuzzy threshold gates*: They behave the same way as ordinary threshold gates when the number of 1-bits in the input is below a threshold t_1 or above a larger threshold $t_2 > t_1$, but no guarantee is made about their behavior in between. Clearly, such gates are less useful than normal threshold gates – the MAJORITY problem demonstrates the importance of precisely distinguishing between $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 1$ many 1-bits – but they turn out to be sufficient for the computation of certain kernels. Crucially, *fuzzy threshold gates can be simulated by AC⁰-circuits for superlogarithmic thresholds*, allowing us to turn such “fuzzy TC⁰ algorithms” into AC⁰-circuits.

² As remarked earlier, in this paper “only the edge sets” of hypergraphs will be kernelized since we insisted that the vertex set is immutable; $|x'|$ should in this case be read as $|E'|$.

In detail, just as for the standard class TC^0 , we consider DLOGTIME -uniform circuit families $(C_n)_{n \in \mathbb{N}}$ of constant depth and polynomial size. However, *instead of* $\text{THRESHOLD}_{\leq t}^t$ -gates, the circuits now contain the following gates (in addition to AND-, OR-, and NOT-gates):

► **Definition 2.1.** *On input $b \in \{0, 1\}^l$, a (“fuzzy”) $\text{THRESHOLD}_{\leq t_1}^{>t_2}$ gate g with l inputs outputs one bit $g(b) \in \{0, 1\}$ such that for $s = \sum_{i=1}^l b[i]$ we have:*

1. *If $s \leq t_1$, then $g(b) = 0$.*
2. *If $s > t_2$, then $g(b) = 1$.*

In all other cases, when s exceeds the threshold only “slightly” ($t_1 < s \leq t_2$), no guarantees are made about $g(b)$: it can be 0 or 1.

A fuzzy threshold circuit C_n with m output gates may produce as output any string from a set $C_n^{\text{possible}}(b) \subseteq \{0, 1\}^m$ of possible outputs, depending on how the fuzzy gates happen to behave. The objective is, of course, that no matter how the fuzzy gates actually behave and no matter which $z \in C_n^{\text{possible}}(b)$ we actually get, it will be a valid “solution” for the given input b . A bit more formally, we define a *solution relation* as a relation $S \subseteq \{0, 1\}^* \times \{0, 1\}^*$ such that for each possible input $b \in \{0, 1\}^*$ the set $S(b) := \{s \mid (b, s) \in S\}$ of (“allowed” or “permissible”) *solutions* is a subset of $\{0, 1\}^{p(|b|)}$ for some fixed polynomial p . As an example, consider the task of finding quadratic kernels for the vertex cover problem. We model this by an S containing all pairs (b, s) where b is (the encoding of) a graph G and a number k and s is (the encoding of) a graph K of size at most k^2 such that G has a size- k vertex cover if, and only if, K does.

► **Definition 2.2.** *Let S be a solution relation. We say that a family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits computes solutions for S if $C_n^{\text{possible}}(b) \subseteq S(b)$ holds for all $b \in \{0, 1\}^*$.*

We derive rather tight fuzzy threshold gates from an result of Ajtai about approximate counting with first-order formulas over arithmetic structures [2]:

► **Fact 2.3** (Theorem 2.1 in [2]). *For each positive integer i there exists an $\text{FO}[+, \times]$ formula $\varphi(x, X)$ with free variable x and free unary set variable X such that we have for each relational structure \mathcal{S} over size- n universe U , each threshold $t \in U$, and each set $A \subseteq U$:*

$$\begin{aligned} |A| \leq (1 - (\log n)^{-i}) \cdot t &\implies \mathcal{S} \models \neg \varphi(t, A), \\ |A| \geq (1 + (\log n)^{-i}) \cdot t &\implies \mathcal{S} \models \varphi(t, A). \end{aligned}$$

► **Corollary 2.4.** *Let t be a threshold and $\epsilon > 0$ be a small error term. There is a constant c such that for each n -input circuit C with $\text{THRESHOLD}_{\leq t}^{>(1+\epsilon)t}$ gates there is a circuit C' without such gates (a normal AC-circuit) with*

1. *$\text{depth}(C') = \text{depth}(C) \cdot c$ and $\text{size}(C') = \text{size}(C)^c$ such that*
2. *for all $b \in \{0, 1\}^n$ we have $C'(b) \in C^{\text{possible}}(b)$.*

Proof. Let C be an n -input circuit with an \tilde{n} -input $\text{THRESHOLD}_{\leq t}^{>(1+\epsilon)t}$ gate g . We construct a circuit C' by replacing g with an AC^0 -subcircuit. By Fact 2.3 and the well-known relation that the set of decision problems definable in $\text{FO}[+, \times]$ is exactly DLOGTIME -uniform AC^0 [20], we know that for every $i \in \mathbb{N}$ and every $a \in \mathbb{N}$ there is an AC^0 -circuit \tilde{C} such that for all $w \in \{0, 1\}^{\tilde{n}}$ we have:

$$\begin{aligned} \sum_{i=1}^{|w|} w_i \leq (1 - (\log \tilde{n})^{-i}) \cdot a &\implies \tilde{C}_{|w|}(w) = 0, \\ \sum_{i=1}^{|w|} w_i \geq (1 + (\log \tilde{n})^{-i}) \cdot a &\implies \tilde{C}_{|w|}(w) = 1. \end{aligned}$$

In order to replace the gate g with an AC^0 -circuit, observe that there is a constant $n_0 > 0$ such that $1 - (\log n)^{-1} \geq (1 + \epsilon/2)^{-1}$ and $(1 + \epsilon) \cdot (1 + \epsilon/2)^{-1} \geq 1 + (\log n)^{-1}$ for all $n \geq n_0$. If the number of inputs \tilde{n} of g is smaller than n_0 , we can replace g by a hard-wired constant-size AC^0 -circuit. For $\tilde{n} \geq n_0$, we set $a = (1 + \epsilon/2) \cdot t$ and replace g by the circuit from above. Let $x := \sum_{i=1}^{|w|} w_i$ and observe that $x \leq t$ implies $x \leq t = (1 + \epsilon/2)^{-1} \cdot a \leq (1 - (\log n)^{-1}) \cdot a$ and $x > (1 + \epsilon)t = (1 + \epsilon) \cdot (1 + \epsilon/2)^{-1} \cdot a$ implies $x \geq (1 + (\log n)^{-1}) \cdot a$. ◀

► **Corollary 2.5.** *Let S be a solution relation and let $(C_n)_{n \in \mathbb{N}}$ be a family of fuzzy threshold circuits that compute solutions for S . Let $\text{depth}(C_n) \in O(1)$ and $\text{size}(C_n) \in n^{O(1)}$. Then there is a function $f \in \text{AC}^0$ that maps every $b \in \{0, 1\}^*$ to a solution $f(b) \in S(b)$.*

3 Three Ways of Implementing the Critical Core Rule

Recall the “classical” sunflower reduction rule, Rule 1.2, which asks us to find and then replace a sunflower S of size $k + 1$ by its core. The reason this rule is *safe* is that there is no way of hitting all $p \in S$ with k vertices without hitting the core, as all $p \in S$ are disjoint outside the core. In other words, the hypergraph with edge set $S \ominus c := \{p \setminus c \mid p \in S\}$ has no hitting set of size k . The key insight behind the critical core rule is that *there are other hypergraphs that also do not have hitting sets of size k , but are easier to find*: A hypergraph with $|E| > k \cdot \Delta$, where Δ is the maximum vertex degree, cannot have a hitting set of size k . Naturally, the maximum degree Δ of an arbitrary hypergraph is unbounded *a priori*, but we can still turn this observation into a safe reduction rule, namely Rule 1.3.

In the introduction, we left open the details of the central definition of a *critical core*, which we remedy presently. First, it will be useful to call the number $i(c) := d - |c|$ the *index i* of a core c . Our algorithms will typically process cores in increasing order of index, which means in decreasing order of size. Next, a *setup* is a system of bounds – tailored to a specific computational model – that determines which cores are critical. In detail:

► **Definition 3.1 (Setup, Factor).** *A setup (w, u, l) consists of three monotone sequences of positive integers: the weight sequence $w = (w_i)_{i \in \mathbb{N}}$, prescribing a weight for cores of index i , the uncritical bound sequence $u = (u_i)_{i \in \mathbb{N}}$, and the light bound sequence $l = (l_i)_{i \in \mathbb{N}}$. These sequences must satisfy $w_0 = u_0 = l_0 = 1$ and $w_i, u_i \in \{1, \dots, l_i\}$. The factor of a setup is the largest number f such that $u_{i+1} \geq f \cdot l_i$ holds for all $i \in \mathbb{N}$.*

For a core c and a setup (w, u, l) , the three numbers we will be particularly interested in are $w_{i(c)}$, $u_{i(c)}$, and $l_{i(c)}$; and we will write $w(c)$, $u(c)$, and $l(c)$ for them, respectively.

► **Definition 3.2 (Critical Cores).** *Let (w, u, l) be a setup and let c be a core. The weight of L_c is $w(L_c) = \sum_{e \in L_c} w(e)$. The core is light if $w(L_c) \leq l(c)$, otherwise it is heavy. The core is critical if $L_c \setminus \{c\}$ is not empty, all $c' \supseteq c$ are light, and $w(L_c) > u(c)$.*

We spell out what these definitions mean for an exemplary setup: $w_i = 1$ and $u_i = l_i = k^i$. The factor is k since $u_{i+1} = k^{i+1} = k \cdot k^i = kl_i$ holds. The weight $w(L_c)$ of a link L_c is simply $|L_c|$ since all weights are 1. A core c is light if $|L_c| \leq k^{i(c)} = k^{d-|c|}$, and it is critical if for all $c' \supseteq c$ we have $|L_{c'}| \leq k^{d-|c'|}$, but $|L_c| > k^{d-|c|}$ ($L_c \setminus \{c\} \neq \emptyset$ then holds automatically).

► **Lemma 3.3.** *For a factor- k setup, let c be critical. Then $L_c \ominus c$ has no size- k hitting set.*

Proof. Let i be the index of c . Suppose $L_c \ominus c$ had a hitting set $X \subseteq V \setminus c$ of size k . Then $L_c = \bigcup_{v \in X} L_{c \cup \{v\}}$ and, therefore, also $w(L_c) \leq \sum_{v \in X} w(L_{c \cup \{v\}})$. As c is critical by assumption, we have $w(L_c) > u_i$ and $w(L_{c \cup \{v\}}) \leq l_{i-1}$ since $c \cup \{v\} \supseteq c$. However, this yields $u_i < w(L_c) \leq \sum_{v \in X} w(L_{c \cup \{v\}}) \leq \sum_{v \in X} l_{i-1} = kl_{i-1}$, contradicting $u_i \geq kl_{i-1}$, which we assumed (the setup has factor k). ◀

► **Corollary 3.4.** *For any factor- k setup, any size- k hitting set must hit all critical cores. In particular, Rule 1.3 is a safe kernel rule for the hitting set problem.*

A simple, but crucial property of the critical core rule is that it removes all heavy links:

► **Lemma 3.5.** *Exhaustively applying the critical core reduction rule to a d -hypergraph H yields an edge set K without heavy cores. In particular, the empty set will be a light core and $|K| = |L_\emptyset| \leq l_d$.*

Proof. If there is a core that is heavy for K , then there is also a heavy core c of minimal index i – meaning that all $c' \supseteq c$ are light. By definition of c being heavy, $w(L_c) > l_i$. But $l_i \geq u_i$ and all $c' \supseteq c$ are light. Thus, c is critical contrary to the assumption. ◀

The following algorithm and theorem summarizes the above findings.

■ **Algorithm 1** The critical core reduction algorithm, which we run on an hypergraph $H = (V, E)$ for some fixed setup.

```

1  while there is a critical core  $c \subseteq e \in E$  do
2     $E \leftarrow (E \setminus L_c) \cup \{c\}$ 
3  return  $E$ 

```

► **Theorem 3.6.** *For every factor- k setup, Algorithm 1 outputs an edge set K of size $|K| \leq l_d$ that is solution-preserving for the hitting set problem.*

Clearly, the smaller l_d , the smaller our kernels. Since by Definition 3.1 we need to ensure $l_{i+1} \geq u_{i+1} \geq kl_i$, the smallest l_d is obtained when we set $u_i = l_i = k^i$. Interestingly, the weights w_i are not relevant yet (we will need them later on) and can be chosen arbitrarily between 1 and $l_i = k^i$.

3.1 Computing Kernels in Linear Time

In this section we discuss a linear time implementation of Algorithm 1, similar to the one provided by Fafianie and Kratsch [16]. However, we differ in two regards: First, the cited implementation directly computes an edge-preserving kernel of size $(k+1)^d$, while we compute a solution-preserving kernel of size k^d . Secondly, we introduce another rule – *the critical core expansion rule* – that allows us to transform the previous computed kernel to an edge-preserving kernel of size $(k+1)^d$. This approach turns out to be slightly more complicated than the implementation of Fafianie and Kratsch, but it allows a straight forward parallelization, which we discuss in the following sections.

It is not too hard to implement a *single application* of the critical core rule in time $|E| \cdot 2^d \text{poly}(d)$: Iterate over all $e \in E$ and for each $c \subseteq e$ increase a counter $n[c]$ by the edge's weight. Determine a maximal c with $w(L_c) = n[c] > u_i$ in a second loop (i is the index of c) and then apply the rule. Since each application of the rule reduces the number of edges by at least 1, we get a total runtime of $|E|^2 \cdot 2^d \text{poly}(d)$ to compute the kernel.

To improve the runtime to $|E| \cdot 2^d \text{poly}(d)$, we need a new definition: For a d -hypergraph $H = (V, E)$, let us call a set K of subsets of V *light* if in the hypergraph (V, K) all edges are light, and we say that K *can be obtained from E* if there is a sequence E_0, E_1, \dots, E_q with $E_0 = E$ and $E_q = K$ such that each E_{j+1} is obtained from E_j through one of two actions:

1. We can set $E_{j+1} = E_j \cup \{c\}$ if c is critical in (V, E_j) .
2. We can set $E_{j+1} = E_j \setminus \{e\}$ if there is a $c \subsetneq e$ with $c \in E_j$.

The critical core rule is now the special case where we always do the first action for some critical c , immediately followed by doing the second action for all $e \supseteq c$, that is, of all of L_c (except for c itself, of course).

► **Theorem 3.7.** *For any factor- k setup, there is a $|E| \cdot 2^d \text{poly}(d)$ time algorithm that, on input of a d -hypergraph $H = (V, E)$, obtains a size- l_d solution-preserving kernel K for the hitting set problem. In particular, for $l_i = k^i$, a kernel of size k^d can be computed in time $|E| \cdot 2^d \text{poly}(d)$.*

Proof. The algorithm iterates over all indices $i \leftarrow 1, \dots, d$ in d rounds. At the start of round i , all cores c of index $i' < i$ will be light and the objective is to ensure that at the end of the round there is no heavy core c of index i . Towards this aim, we wish to modify E (compute E_{j+1} from the current E_j , but let us just write that we “modify E ”) by

1. *removing* all edges in $R := \{e \in E \mid e \supseteq c \text{ for some critical set } c \text{ of index } i\}$,
2. *adding* all edges in a set $A_0 \subseteq A := \{c \mid c \text{ is critical and has index } i\}$ such that
3. A_0 has the properties $|A_0| \leq |R|$ and $R = \{e \in E \mid \exists c \in A_0 : e \supseteq c\}$.

Suppose we could find A_0 and R efficiently. Then we can, indeed, add all of A_0 and then remove all of R in accordance with the two rules: All $c \in A_0$ are initially critical and stay this way when we add other $c' \in A_0$ of the same size to E ; and we can then safely remove all of R as all $e \in R$ are proper supersets of some $c \in A_0$ that we have just added. Finally note that after we have added A_0 and removed R , there are no heavy c of size $|c| = r$ for the resulting set E : If c with $|c| = r$ were still heavy in j , then c would also have been heavy in the original E prior to the modifications and, thus, also critical. But, then, $e \in R$ would have been true for all e in c 's link in E and, thus, all these e would have been removed. Putting it all together, we see that we can, indeed, use the updated E for the next round.

It remains to show how R and A_0 can be found. First, we iterate over all $e \in E$ and all $c \subseteq e$ with $|c| = i$ and for each such c increment a counter $n[c]$ by $w_{d-|e|}$. Note that at the end of the first loop we have $n[c] = w(L_c)$ for all such c . Second, we once more iterate over all $e \in E$ and for each of them we now check whether there is a $c \subsetneq e$ of index i such that $n[c] > u_{i-1}$. If so, we add e to R and we mark *one* such c as to belonging to A_0 . Clearly, at the end of the second loop we will have correctly computed R and a set A_0 of critical edges with $|A_0| \leq |R|$ and $R = \{e \in E \mid \exists c \in A_0 : e \supseteq c\}$.

The runtime of the algorithm is $|E| \cdot 2^d \text{poly}(d)$, assuming an efficient implementation of the array of counters $n[c]$: In the round for core size r , we iterate twice over at most $|E|$ edges e and each time consider at most $\binom{d}{i}$ subsets $c \subsetneq e$. Removing R and adding A_0 takes time linear in their sizes. Finally, $|A_0| \leq |R|$ ensures that $|E|$ can only shrink in each round, yielding a total runtime of at most $\sum_{i=0}^{d-1} \binom{d}{i} |E| \cdot \text{poly}(d) = |E| \cdot 2^d \text{poly}(d)$ as claimed. ◀

While we can now compute solution-preserving kernels K in time $|E| \cdot 2^d \text{poly}(d)$, the kernels are *not yet* edge-preserving: many $e \in K$ will not be elements of the original edge set E . In the following we present an *expansion* rule that can be used to turn K into an edge-preserving kernel. Recall that the critical core *reduction* rule replaces E by $(E \setminus L_c) \cup \{c\}$ for a critical c . The “reverse” version replaces K by $(K \setminus \{c\}) \cup L'_c$, where $L'_c \subseteq L_c$ is a subset large enough to ensure that c becomes critical.

► **Rule 3.8** (Critical Core Expansion Rule). In a d -hypergraph $H = (V, E)$, let $c \notin E$ be critical and let $E' = (E \setminus L_c) \cup \{c\}$. Determine an $L' \subseteq L_c$ with $l(c) \geq w(L') > u(c)$ and replace E' by $E'' := (E' \setminus \{c\}) \cup L'$.

(If $c \in E$ is critical, the critical core reduction rule just removes all supersets of c from E . In this case we also consider the expansion rule to be applicable and set $E'' = E'$.)

► **Lemma 3.9.** *For every setup with $w_i = l_i$, if we apply the critical core expansion rule to E' for some c , then $w(E'') \leq w(E')$ will hold.*

Proof. $w(E'') = w(E') - w(c) + w(L')$ and $w(c) = l(c) \geq w(L')$. ◀

► **Lemma 3.10.** *For every factor- k setup with $w_{i-1} + u_i \leq l_i$, if $c \notin E$ is critical for E , then there is a set L' such that the critical core expansion rule is applicable to the set $E' = (E \setminus L_c) \cup \{c\}$ and E'' will have the same size- k hitting sets as E (and E').*

Proof. Let c be critical for E and have index i . The set L' can be obtained from L_c by iteratively adding elements of L_c to L' until we have $w(L') > u(c)$ for the first time (at the latest when all of L_c has been added). Let $e \supseteq c$ be the last element added to L' . Then $w(L' \setminus \{e\}) \leq u(c) = u_i$ and $w(L') = w(L' \setminus \{e\}) + w(e) \leq u_i + w_{i-1} \leq l_i$. To see that E' and E'' have the same size- k hitting sets, observe that each hitting set of E' must hit $c \in E'$ and is thus also a hitting set of E'' . For the other direction note that c is critical in E'' : No $c' \supseteq c$ can be heavy in E'' since no such c' was heavy in $E \supseteq E''$. With c being critical in E'' , any size- k hitting set of E'' must hit c and, thus, all of E' and thus also all of E . ◀

► **Theorem 3.11.** *For every factor- k setup with $w_i = l_i$ and $w_{i-1} + u_i \leq l_i$, there is an algorithm running in time $|E| \cdot 2^d \text{poly}(d)$ that on input of a d -hypergraph $H = (V, E)$ outputs a kernel K of size l_d that is edge- and solution-preserving for the hitting set problem.*

Proof of Theorem 3.11. By the conditions we impose on the weights, the two lemmas tell us that any application of the critical core reduction rule can be reversed by expansion (Rule 3.8). In particular, an algorithm can use the rule to “undo” the replacement of E by $E' := (E \setminus R) \cup A_0$ by finding a set L' for each $c \in A_0$. Observe that we can remove all of A_0 from E' (except for those $c \in A_0$ that were already present in E) and add an appropriate L' for each such $c \in A_0$ to, still, ensure that all $c \in A_0$ are still critical. ◀

The smallest setup that satisfies the conditions of the theorem is $w_i = l_i = (k+1)^i$ and $u_i = k(k+1)^{i-1}$ for $i \geq 1$. Thus, the theorem tells us that we can compute edge-preserving kernels of size $(k+1)^d$ for the hitting set problem in linear time.

We point out that there is a much simpler way of implementing the critical core rule sequentially in order to compute an edge-preserving kernel. Algorithm 2 is essentially the algorithm of Fafianie and Kratsch [16] in our terminology.

■ **Algorithm 2** The critical core filter algorithm. When started on $H = (V, E)$, it will output a kernel $K \subseteq E$ of size at most l_d – provided that the setup satisfies $u_i + w_i \leq l_i$, see Theorem 3.12.

```

1  $K \leftarrow \emptyset$ 
2 for  $e \in E$  do
3   if there is no  $c \subseteq e$  that is critical with respect to  $K$  then
4      $K \leftarrow K \cup \{e\}$ 
5 return  $K$ 

```

It is easy to implement the algorithm so that it runs in time $|E| \cdot 2^d \text{poly}(d)$ by keeping track of the weights of all links in K . Unfortunately, the algorithm does not seem to be suitable for parallelisation. However, analyzing the algorithm with our proposed setups still allows us to bound the kernel size slightly better than it was done by Fafianie and Kratsch [16]:

► **Theorem 3.12.** *For every factor- k setup with $u_i + w_i \leq l_i$, the output of Algorithm 2 is an edge-preserving hitting set kernel for H of size at most l_d .*

Proof. By construction, we have $K \subseteq E$. We have $|K| \leq l_d$, since $c = \emptyset$ is light in K . To see that K is a kernel, consider an $e \in E$ that we do not add to K . Then there must be a core $c \subseteq e$ such that for the link L_c of c in K we have $w(L_c^K) + w(e) > l(c)$ (otherwise we would have added e to K). For $i = i(c)$, we then have $l_i < w(L_c^K) + w_i$ and, by assumption, $u_i \leq l_i - w_i < w(L_c^K)$. This means that c was critical in K and, hence, every size- k hitting set of K also hits c and, thus, in particular also $e \supseteq c$ and there is no need to add e to K . ◀

The slowest growing setup with factor k satisfying $u_i + w_i \leq l_i$ is given by $w_i = 1$ and $u_i = \sum_{j=1}^i k^j = k + k^2 + \dots + k^i$ and $l_i = u_i + 1 = \sum_{j=0}^i k^j = 1 + k + k^2 + \dots + k^i$. Thus, a kernel with $\sum_{j=0}^i k^j$ edges can be achieved, which is slightly better than the previously known bound of $(k + 1)^d$.

3.2 Computing Kernels by Constant-Depth Threshold Circuits

The algorithm from Theorem 3.7 allows an efficient parallel implementation:

► **Theorem 3.13.** *For each d , the hitting set kernels from Theorems 3.7 and 3.11 can be computed by TC^0 -circuits. In particular, TC^0 -circuits can compute solution-preserving kernels of size k^d and edge-preserving kernels of size $(k + 1)^d$ for the hitting set problem for d -hypergraphs.*

Proof. Just observe that a TC^0 -circuit can determine whether $n[c] > u(c)$ holds for a given c without iterating over all $e \in E$ sequentially, but by using a single threshold gate. In particular, we can compute R in parallel and thus also A_0 . This yields a circuit whose depth is linear in d (a constant) and whose size is polynomial in $|E| \cdot 2^d$ and thus polynomial in the input length for constant d . We can also implement the reverse critical link rule using threshold gates since to identify the sets L' from the rule, we just need threshold gates to find the first edge $e \in L_c$ for which the sum of the weights of all edges in L_c prior to this edge together with $w(e)$ exceeds u_i . ◀

We point out that for solution-preserving kernels, the above theorem was already known for $d = 2$ [6], while only the logspace kernelization from Fafiane and Kratsch [16] were known for $d > 2$.

3.3 Computing Kernels by Constant-Depth Fuzzy Threshold Circuits

Our final goal is an implementation of our kernelization using AC^0 -circuits. As pointed out earlier, previously it was not known how kernels of polynomial size can be computed by AC^0 -circuits for *any* problem. The difficulty in computing polynomial-size kernels using AC^0 -circuits lies in the inability of such circuits to count precisely when the parameter k is no longer polylogarithmic. We overcome this problem by using fuzzy threshold gates.

Critical Core Reduction by Fuzzy Thresholds. We wish to replace the TC^0 -circuit family from Theorem 3.13 by a fuzzy one, and then we wish to apply Corollary 2.5 to turn it into an AC^0 -circuit family. The threshold circuits from Theorem 3.13 really need to be precise: If we naively replace all threshold gates by fuzzy ones, it can happen that a critical c is not detected, but a smaller one is – resulting in an incorrect application of the rule. The other way round, it may also happen that the rule is not applied when it actually should.

We solve these problems by using a setup with worse (but still polynomial) bounds. The jump from one bound to the next is so big that the fuzziness is no longer a problem:

► **Theorem 3.14.** *For any $\delta > 1$, factor- k setup with $l_i \geq \delta \cdot u_i$ and $d \in \mathbb{N}$, there is a constant-depth, polynomial-size family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits that, on input of a hypergraph $H = (V, E)$ and a number k , outputs a solution-preserving size- l_d hitting set kernel K . In particular, a kernel of size $\delta^{d-1}k^d$ can be computed using $u_i = \delta^{i-1}k^i$ and $l_i = \delta^i k^i$.*

Proof. Let us first reiterate the steps from the proof of Theorem 3.13, but now with a closer look at where and how threshold gates are needed (and with which thresholds).

The idea behind the kernelization was that at the beginning of a round for some index $i \in \{1, \dots, d\}$, all cores of smaller index are light and the objective is to ensure that at the end of the round this is also true for all cores of index i . To ensure this, we identified all c of index i that were critical: For each possible $c \subseteq e \in E$ (of which there can be at most $\binom{d}{i}|E|$ many), a normal TC^0 -circuit would use a single threshold gate at this point to determine whether $w(L_c^E)$ exceeds the threshold u_i . If so, c gets marked and then included in the process by which A_0 and R are determined, but this process does not include any use of threshold gates – it is only the test whether $w(L_c^E) > u_i$ where we need threshold gates.

In the fuzzy setting, we will need to use a fuzzy $\text{THRESHOLD}_{\leq t}^{\delta t}$ -gate to implement the test $w(L_c) > u_i$ using, of course, $t = u_i$. This has the following effects:

1. If $w(L_c^E) \leq u_i = t$ holds, then we safely and correctly identify c as noncritical as the test will always yield 0.
2. If $w(L_c^E) > \delta t = \delta u_i$, then c is safely and correctly identified as critical as the test will always yield 1.
3. If $w(L_c^E)$ is in the fuzzy range, then c is guaranteed to be *critical, but not heavy* (since $\delta u_i \leq l_i$).

The important observation is that in the third item, it is *correct* to apply the critical link rule to c (which we do if the fuzzy threshold outputs 1), but it is not *necessary* to do so since c is light. In particular, after the round for i is done, there will be *no heavy c of index i in E* , which was exactly our objective. ◀

► **Corollary 3.15.** *For every d and any $\epsilon > 0$, there is a function in AC^0 that maps every d -hypergraph and number k to a solution-preserving hitting set kernel of size at most $(1 + \epsilon)k^d$.*

Critical Core Expansion by Fuzzy Thresholds. Like the critical core reduction rule, the critical core expansion rule can be implemented by fuzzy threshold circuits. Recall that Rule 3.8 allowed us to safely replace a set E' of edges, which had been obtained by setting $E' = (E \setminus L_c) \cup \{c\}$, by the set $E'' = (E' \setminus \{c\}) \cup L'$ for any set $L' \subseteq L_c$ of appropriate size – namely for $l(c) \geq w(L') > u(c)$. As we did earlier, using a setup with more “slack” will allow us to replace threshold gates by fuzzy threshold gates here.

► **Theorem 3.16.** *For any $\delta > 1$, factor- k setup with $w_i = l_i$ and $w_{i-1} + \delta u_i \leq l_i$ and $d \in \mathbb{N}$, there is a constant-depth, polynomial-size family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits that, on input of a d -hypergraph $H = (V, E)$ and a number k , outputs a size- l_d hitting set kernel K that is edge- and solution-preserving. In particular, a kernel of size $\delta^d(k+1)^d$ can be computed using the setup with $w_i = l_i = \delta^i(k+1)^i$ and $u_i = \delta^{i-1}k(k+1)^{i-1}$.*

Proof. We once more need to look more closely at how threshold gates are used in Theorem 3.13 to obtain $L' \subseteq L_c$: For $L_c = \{e_1, e_2, \dots, e_{|L_c|}\}$ we use $|L_c|$ threshold gates in parallel, each of which tests for some number $j \in \{1, \dots, |L_c|\}$ whether $w(\{e_1, \dots, e_j\}) > u(c)$ holds – and the smallest j for which is the case determines $L' = \{e_1, \dots, e_j\}$. Then,

clearly, $w(L') > u(c)$ and also $w(L') \leq u_i + w_{i-1}$, which will be less than l_i by assumption. Now, in the fuzzy setting, we use fuzzy gates for tests, which will still ensure that $w(L') = w(\{e_1, \dots, e_j\}) > u(c)$ holds; but for the upper bound we can only ensure $w(L') \leq \delta u_i + w_{i-1}$ since as long as $w(\{e_1, \dots, e_j\}) \leq \delta u_i$ holds, the fuzzy threshold gates might still output 0, making us (incorrectly) believe that $\{e_1, \dots, e_j\}$ is not yet heavy enough. Thus, in order for the critical core expansion rule to work, we need $\delta u_i + w_{i-1} \leq l_i$, which is exactly our assumption.

For the setup $w_i = l_i = \delta^i(k+1)^i$ and $u_i = kl_{i-1} = \delta^{i-1}k(k+1)^{i-1}$, observe that it does, indeed, satisfy all properties needed by the different lemmas on which the correctness of the rule is based:

1. It satisfies $w_i \leq l_i$ and $u_i = \delta^{i-1}k(k+1)^{i-1} \leq \delta^i(k+1)^i = l_i$ (needed by Definition 3.1).
2. It has factor k since $u_{i+1} = \delta^i k(k+1)^i = kl_i$ (needed by Lemma 3.3).
3. It has $\delta u_i + w_{i-1} = \delta^i k(k+1)^{i-1} + \delta^{i-1}(k+1)^{i-1} = \delta^i(k+1)^{i-1}(k+1/\delta) < \delta^i(k+1)^i = l_i$ (needed by this theorem). ◀

4 Constant-Time Approximation Algorithms for Hitting Set

Approximation algorithms compute solutions for optimization problems that, while perhaps not optimal, have a size that is at least “close” to the optimum. For a minimization problem like VERTEX-COVER, the ultimate objective is to output a vertex cover X whose size is at most $(1 + \epsilon)|X^*|$, where X^* denotes some optimal solution. It turns out that unless some complexity classes collapse, such near-optimal approximations cannot be computed for the vertex cover problem. The best approximation to date is to compute a maximal matching and to then take all vertices involved in it. This algorithm delivers solutions of size at most $2|X^*|$ and can be implemented using NC^2 -circuits. However, no approximation algorithm that produces a solution that is at least polynomially bounded in $|X^*|$ was known to be implementable with circuits below NC^2 . In this section we present such an algorithm.

Our strategy is based on a simple observation: The set of vertices in any solution-preserving kernel is already a solution for the original graph and, thus, also an approximation. However, we still have the problem that we do not know the size of the optimal solution and, thus, do not know which number k we should use with our kernel algorithms.

► **Theorem 4.1.** *For each d and $\epsilon > 0$, there are functions f and g that map (the encodings of) d -hypergraphs H to hitting sets of H , such that (let X^* be a minimum hitting set of H):*

1. $f \in \text{TC}^0$ and $|f(H)| \leq d|X^*|^d$;
2. $g \in \text{AC}^0$ and $|g(H)| \leq (1 + \epsilon) \cdot d|X^*|^d$.

Proof. The idea is identical for both claims. On input $H = (V, E)$, we run the following algorithm in parallel for each $k \in \{1, \dots, |V|\}$: Compute a solution-preserving kernel K_k for H using the circuits from Theorem 3.13 for TC^0 or using those from Corollary 3.15 for AC^0 . Then test whether K_k is actually a hitting set of H (this test can easily be done using AC^0 -circuits). Output the set $\bigcup K_k$ (the set of all vertices mentioned in any edges $e \in K_k$) for the smallest k that passes the test, that is, for which K_k is still a hitting set of K .

Trivially, the outputs of the described circuits will be hitting sets. For the size bounds, observe that all K_k are solution-preserving: They have the same size- k hitting sets as the original hypergraph H . In particular, for $k = |X^*|$, the kernel K_k will have X^* as a hitting set and $\bigcup K_k$ will contain X^* and will thus hit all of H . The size bounds now follow from the size bounds on K_k in Theorem 3.13 and Corollary 3.15. ◀

5 Adapting the Approach for the Set Packing Problem

The dual problem of p_k - d -HITTING-SET is the *set packing problem*, in which we are asked to find k disjoint edges in a hypergraph:

► **Problem 5.1** (p_k - d -SET-PACKING for fixed $d \in \mathbb{N}$).

Instances: A d -hypergraph $H = (V, E)$ and a *parameter* $k \in \mathbb{N}$.

Question: Is there a set $X \subseteq E$ with $|X| \geq k$ such that any different $e, f \in X$ are disjoint?

Kernelizations based on the critical core rule tend to carry over to the set packing problem [22]. It is thus not too surprising that our approach also works for set packings, though there are also some subtleties.

5.1 Computing Set Packing Kernels

Recall that the safety of the critical core rule for the hitting set problem hinged on Lemma 3.3: For critical c , the set $L_c \ominus c$ has no size- k hitting set and, thus, for the question of whether H has a size- k hitting set all the edges in L_c can be represented by c . We show that a similar situation arises for the set packing problem: For the question of whether H has a size- k set packing, all edges in L_c can be represented by c , which gives us an analogue of Corollary 3.4:

► **Lemma 5.2.** *For a setup with factor $d(k-1)$, let c be critical in a d -hypergraph H . Then for every $U \subseteq V$ of size $|U| \leq d(k-1)$ there is a set $x \in L_c \ominus c$ such that $x \cap U = \emptyset$.*

Proof. Let i be the index of c . We may assume that $U \cap c = \emptyset$ holds (otherwise we can just replace U by $U \setminus c$). Now consider the set $I = \bigcup_{v \in U} L_{c \cup \{v\}} \subseteq L_c$. As c is critical by assumption, we have $w(L_c) > u_i$ and also $w(L_{c \cup \{v\}}) \leq l_{i-1}$. This gives us $w(I) \leq \sum_{v \in U} w(L_{c \cup \{v\}}) \leq |U|l_{i-1} \leq d(k-1)l_{i-1} \leq u_i < w(L_c)$. Hence, there must be an edge $e \in L_c \setminus I$, which means $e \cap U = \emptyset$. Then $x = e \setminus c$ is the desired element of $L_c \ominus c$. ◀

► **Lemma 5.3.** *For any setup with factor $d(k-1)$, let c be critical for $H = (V, E)$.*

1. *If $c \neq \emptyset$, then $(E \setminus L_c) \cup \{c\}$ has a set packing of size k if, and only if, E does.*
2. *If $c = \emptyset$, then E has a set packing of size k .*

Proof. For the first item, first note that if $X \subseteq E$ is a set packing (of any size), then $(X \setminus L_c) \cup \{c\} \subseteq (E \setminus L_c) \cup \{c\}$ is a set packing of the same size as X . For the other direction, $X \subseteq (E \setminus L_c) \cup \{c\}$ is a size- k set packing. Clearly, if $c \notin X$, we have $X \subseteq E$ and we are done, so suppose $c \in X$. Consider the set $U := \bigcup_{x \in X \setminus \{c\}} x$ of all vertices mentioned in any edge of X , except for those in c . Then $|U| \leq d(k-1)$ since H is a d -hypergraph and each of the $k-1$ many elements of $X \setminus \{c\}$ contributes at most d vertices to U . By Lemma 5.2, there is a edge $x \in L_c \ominus c$ that is disjoint from U and trivially also from c . This means that $X' = (X \setminus \{c\}) \cup \{x \cup c\}$ is also a set packing of size k and $X' \subseteq E$.

For the second item, we repeat the following instructions k times, starting with $U = \emptyset$ and $X = \emptyset$: Invoke Lemma 5.2 to obtain $x \in L_c \ominus c = L_c = E$ with $x \cap U = \emptyset$ and update $X \leftarrow X \cup \{x\}$ and $U \leftarrow U \cup x$. Note that in invocations of the lemma we have $|U| \leq d(k-1)$, so we always get a fresh $x \in E$ that is disjoint from all previous elements of X . ◀

The lemma states that just as for the hitting set problem, the critical core rule is *safe* – as long as c is nonempty; and when c is empty and critical, we actually *know* that there is a set packing of size k and can output a trivial kernel. The observations prove the following:

► **Theorem 5.4.** *For any setup with factor $d(k-1)$ and input $H = (V, E)$, let K be a kernel computed by (1) applying the critical core rule for nonempty cores as long as possible and (2) possibly setting K to a trivial yes-instance if the empty set becomes critical at some point. Then K has a size- k set packing if, and only if, H has one; and $|K| \leq l_d$.*

The smallest possible setup with factor $d(k-1)$ is of course $u_i = l_i = (d(k-1))^i$, leading to a kernel size of $l_d = (d(k-1))^d \leq d^d k^d$. Since this kernelization works with the *exact same* reduction rule we used for the hitting set problem – and since the special case of a critical empty core is easy to take care of – we deduce the following results:

► **Corollary 5.5.** *For each $d \in \mathbb{N}$ and $\epsilon > 0$, one can compute on input of a d -hypergraph $H = (V, E)$ a set packing kernel*

- *in time $|E| \cdot 2^d \text{poly}(d)$ of size $(d(k-1))^d$ and an edge-preserving one of size $(dk)^d$,*
- *by TC^0 -circuits of size $(d(k-1))^d$ and an edge-preserving one of size $(dk)^d$, and*
- *by AC^0 -circuits of size $(1+\epsilon)(d(k-1))^d$ and an edge-preserving one of size $(1+\epsilon)(dk)^d$.*

5.2 Approximation Algorithms for the Set Packing Problem

When one compares our kernelization results on the hitting set and the set packing problems, it may seem that these problems behave in identical ways and only the sizes of the outputs differ slightly. However, in the approximation setting, the situation is quite different: For the set packing problem, we do not know how we can extract an approximation from a kernel. To appreciate the underlying difficulties, observe that it is not even clear how one can compute in AC^0 a matching for a graph that is known to be a complete bipartite graph with two given shores U and W of identical size k .

We do not know whether it is possible to compute approximate matchings, let alone set packings, using AC^0 -circuits. It is thus a bit surprising that we can, nevertheless, approximate the *size* of optimal matchings and set packings:

► **Theorem 5.6.** *For each d and $\epsilon > 0$, there are functions f and g that map (the encoding of) each d -hypergraphs H to a number such that (let X^* denote a largest set packing of H):*

1. $f \in \text{TC}^0$ and $\frac{1}{d} |X^*|^{1/d} < f(H) \leq |X^*|$.
2. $g \in \text{AC}^0$ and $\frac{1}{(1+\epsilon)d} |X^*|^{1/d} < g(H) \leq |X^*|$.

Proof. On input $H = (V, E)$, we compute in parallel for each $k \in \{1, \dots, |V|\}$ a set packing kernel K_k for H using the circuits from Corollary 5.5. For each kernel we perform a simple test: Is K_k the trivial kernel? (Recall that this is the case when $c = \emptyset$ because critical during the computation of the kernel and, then, we have a “witness” that the original hypergraph has a set packing of size at least k .) We output the largest k that passes this test.

For the upper bounds, note that whatever k we output, we know that there is a set packing of this size in the output. For the lower bounds, we show that whenever $|X^*| > l_d$, then the trivial kernel will be output. Assume that this not the case for K_k . We observe that $X := X^*$ is a set packing of size $|X^*|$ and for any nonempty c , the set $(X \setminus L_c) \cup \{c\}$ is also a set packing of the same size (see the argument at the beginning of the proof of Lemma 5.3). Thus, the final K_k , which arose through a series of applications of $E \leftarrow (E \setminus L_c) \cup \{c\}$ for different, nonempty c , contains some set packing of size $|X^*|$. In particular, $|K_k| \geq |X^*| > l_d$. But, then, $c = \emptyset$ is critical in K_k , contrary to our assumption. We know that all k pass the test “Is K_k the trivial kernel?” for which $|X^*| > l_d$ holds. For the function f we used $l_d = (d(k-1))^d$, so all k with $|X^*|^{1/d}/d > k-1$ pass the test. For g we used $l_d = ((1+\epsilon)d(k-1))^d$, so now all k with $|X^*|^{1/d}/((1+\epsilon)d) > k-1$ pass the test. ◀

6 Conclusion

We analyzed a simple reduction rule for the hitting set problem in a parallel setting: The *critical core rule* states that for any *critical* set c , we can safely replace the link of c by c . Whether or not a set is critical depended only on the weights of links and by varying the thresholds, we got different kernelization algorithms with different properties, see Table 1.

From the perspective of circuit complexity, this paper gives two new insights: First, it is possible to compute polynomial-size kernels for difficult problems using AC^0 -circuits; and second, it is possible to find polynomial-factor approximations for the hitting set problem using AC^0 -circuits.

■ **Table 1** Summary of our results concerning kernels and approximations for the hitting set problem (above) and the set packing problem (below) in d -hypergraphs. The number opt is the size $|X^*|$ of a smallest hitting set or a largest set packing of the input, respectively.

Hitting Set Kernelization Results

<i>Runtime</i>	<i>What is Computed?</i>	<i>Size</i>	<i>Reference</i>
$ E \cdot 2^d \text{ poly}(d)$	solution-preserving hitting set kernel	k^d edges	Theorem 3.7
TC^0	solution-preserving hitting set kernel	k^d edges	Theorem 3.13
AC^0	solution-preserving hitting set kernel	$(1 + \epsilon)k^d$ edges	Corollary 3.15
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving hitting set kernel	$\sum_{j=0}^d k^j$ edges	Theorem 3.12
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving hitting set kernel	$(k + 1)^d$ edges	Theorem 3.11
TC^0	edge-preserving hitting set kernel	$(k + 1)^d$ edges	Theorem 3.13
AC^0	edge-preserving hitting set kernel	$(1 + \epsilon)(k + 1)^d$ edges	Corollary 3.15
NC^2	a hitting set	$d \cdot opt$ vertices	folklore
TC^0	a hitting set	opt^d vertices	Theorem 4.1
AC^0	a hitting set	$(1 + \epsilon)opt^d$ vertices	Theorem 4.1

Set Packing Kernelization Results

<i>Runtime</i>	<i>What is Computed?</i>	<i>Size</i>	<i>Reference</i>
$ E \cdot 2^d \text{ poly}(d)$	set packing kernel	$(d(k - 1))^d$ edges	Corollary 5.5
TC^0	set packing kernel	$(d(k - 1))^d$ edges	Corollary 5.5
AC^0	set packing kernel	$(1 + \epsilon)(d(k - 1))^d$ edges	Corollary 5.5
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving set packing kernel	$(dk)^d$ edges	Corollary 5.5
TC^0	edge-preserving set packing kernel	$(dk)^d$ edges	Corollary 5.5
AC^0	edge-preserving set packing kernel	$(1 + \epsilon)(dk)^d$ edges	Corollary 5.5
NC^2	number z	$\frac{1}{d} opt \leq z \leq opt$	folklore
TC^0	number z	$\frac{1}{d} opt^{1/d} \leq z \leq opt$	Theorem 5.6
AC^0	number z	$\frac{1}{(1+\epsilon)^d} opt^{1/d} \leq z \leq opt$	Theorem 5.6

References

- 1 F. Abu-Khazam, C. Bazgan, M. Chopin, and H. Fernau. Approximation algorithms inspired by kernelization methods. In *ISAAC*, 2014. doi:10.1007/978-3-319-13075-0_38.
- 2 M. Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances In Computational Complexity Theory*, 1990.
- 3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4), 1995. doi:10.1145/210332.210337.
- 4 M. Bannach, C. Stockhusen, and T. Tantau. Fast Parallel Fixed-parameter Algorithms via Color Coding. In *IPEC*, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.
- 5 M. Bannach and T. Tantau. Computing Hitting Set Kernels By AC^0 -Circuits. In *STACS*, 2018. doi:10.4230/LIPIcs.STACS.2018.9.
- 6 M. Bannach and T. Tantau. Computing Kernels in Parallel: Lower and Upper Bounds. In *IPEC*, 2018.
- 7 D. Barrington, N. Immerman, and H. Straubing. On Uniformity within NC^1 . In *Structure in Complexity Theory*, 1988. doi:10.1109/SCT.1988.5262.
- 8 H. Bodlaender, R. Downey, M. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8), 2009. doi:10.1016/j.jcss.2009.04.001.
- 9 L. Cai, J. Chen, R. Downey, and M. R. Fellows. Advice Classes of Parameterized Tractability. *Annals of Pure and Applied Logic*, 84(1), 1997. doi:10.1016/S0168-0072(95)00020-8.
- 10 M. Cesati and M. Di Ianni. Parameterized parallel complexity. In *Euro-Par*, 1998. doi:10.1007/BFb0057945.
- 11 Y. Chen, J. Flum, and X. Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *CSL*, 2017. doi:10.4230/LIPIcs.CSL.2017.19.
- 12 P. Damaschke. Parameterized Enumeration, Transversals, and Imperfect Phylogeny Reconstruction. *Theo. Comp. Science*, 351(3), 2006. doi:10.1016/j.tcs.2005.10.004.
- 13 R. Downey and M. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4), 1995. doi:10.1137/S0097539792228228.
- 14 M. Elberfeld, C. Stockhusen, and T. Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3), 2015. doi:10.1007/s00453-014-9944-y.
- 15 P. Erdős and R. Rado. Intersection Theorems for Systems of Sets. *Journal of the London Mathematical Society*, 1(1), 1960.
- 16 S. Fafanie and S. Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In *MFCS*, 2015. doi:10.1007/978-3-662-48054-0_25.
- 17 M. Fellows, A. Kulik, F. Rosamond, and H. Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93, 2018. doi:10.1016/j.jcss.2017.11.001.
- 18 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.
- 19 M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1), 1984. doi:10.1007/BF01744431.
- 20 N. Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
- 21 D. Lokshantov, F. Panolan, M. Ramanujan, and S. Saurabh. Lossy kernelization. In *STOC*, 2017. doi:10.1145/3055399.3055456.
- 22 H. Moser. A problem kernelization for graph packing. In *SOFSEM*, volume 5404 of *Lecture Notes in Computer Science*, pages 401–412. Springer, 2009.
- 23 I. Newman, P. Ragde, and A. Wigderson. Perfect hashing, graph entropy, and circuit complexity. In *Structure in Complexity Theory*, 1990. doi:10.1109/SCT.1990.113958.
- 24 R. van Bevern. Towards Optimal and Expressive Kernelization for d -Hitting Set. *Algorithmica*, 70(1), September 2014. doi:10.1007/s00453-013-9774-3.

Graph Realizations: Maximum Degree in Vertex Neighborhoods

Amotz Bar-Noy

City University of New York (CUNY), NY, USA
amotz@sci.brooklyn.cuny.edu

Keerti Choudhary

Tel Aviv University, Israel
keerti.choudhary@cs.tau.ac.il

David Peleg

Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Dror Rawitz

Bar Ilan University, Ramat-Gan, Israel
dror.rawitz@biu.ac.il

Abstract

The classical problem of *degree sequence realizability* asks whether or not a given sequence of n positive integers is equal to the degree sequence of some n -vertex undirected simple graph. While the realizability problem of degree sequences has been well studied for different classes of graphs, there has been relatively little work concerning the realizability of other types of information *profiles*, such as the vertex neighborhood profiles.

In this paper, we initiate the study of *neighborhood degree* profiles, wherein, our focus is on the natural problem of realizing *maximum* neighborhood degrees. More specifically, we ask the following question: “Given a sequence D of n non-negative integers $0 \leq d_1 \leq \dots \leq d_n$, does there exist a simple graph with vertices v_1, \dots, v_n such that for every $1 \leq i \leq n$, the maximum degree in the neighborhood of v_i is exactly d_i ?”

We provide in this work various results for maximum-neighborhood-degree for general n vertex graphs. Our results are first of its kind that studies extremal neighborhood degree profiles. For closed as well as open neighborhood degree profiles, we provide a *complete realizability criteria*. We also provide tight bounds for the number of maximum neighbouring degree profiles of length n that are realizable. Our conditions are verifiable in linear time and our realizations can be constructed in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph realization, neighborhood profile, extremum-degree

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.10

Funding W911NF-09-2-0053 (the ARL Network Science CTA), US-Israel BSF grant 2018043.

1 Introduction

Background and Motivation. In many application domains involving networks, it is common to view vertex degrees as a central parameter, providing useful information concerning the relative significance (and in certain cases, centrality) of each vertex with respect to the rest of the network, and consequently useful for understanding the network’s basic properties. Given an n -vertex graph G with adjacency matrix $Adj(G)$, its *degree sequence* is a sequence consisting of its vertex degrees,

$$\text{DEG}(G) = (d_1, \dots, d_n).$$



© Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 10; pp. 10:1–10:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Given a graph G or its adjacency matrix, it is easy to extract the degree sequence. An interesting *dual* problem, sometimes referred to as the *realization* problem, concerns a situation where given a sequence of nonnegative integers D , we are asked whether there exists a graph whose degree sequence conforms to D . A sequence for which there exists a realization is called a *graphic* sequence. Erdős and Gallai [10] gave a necessary and sufficient condition for deciding whether a given sequence of integers is graphic (also implying an $O(n)$ decision algorithm). Havel and Hakimi [12, 13] gave a recursive algorithm that given a sequence of integers computes in $O(m)$ time a realizing m -edge graph, if such a graph exists.

Over the years, various extensions of the degree realization problem were studied as well, cf. [1, 3, 23], concerning different characterizations of degree-profiles. The motivation underlying the current paper is rooted in the observation that realization questions of a similar nature pose themselves naturally in a large variety of *other* application contexts, where given *some* type of information profile specifying the desired vertex properties (be it concerning degrees, distances, centrality, or any other property of significance), it can be asked whether there exists a graph conforming to the specified profile. Broadly speaking, this type of investigation may arise, and find potential applications, both in scientific contexts, where the information profile reflects measurement results obtained from some natural network of unknown structure, and the goal is to obtain a model that may explain these measurements, and in engineering contexts, where the information profile represents a specification with some desired properties, and the goal is to find an implementation in the form of a network conforming to that specification.

This basic observation motivates a vast research direction, which was little studied over the last five decades. In this paper we make a step towards a systematic study of one specific type of information profiles, concerning *neighborhood degree* profiles. Such profiles are of theoretical interest in context of social networks (where degrees often reflect influence and centrality, and consequently neighboring degrees reflect “closeness to power”). Neighborhood degrees were considered before in [5], where the profile associated with each vertex i is the *list* of degrees of all vertices in i ’s neighborhood. In contrast, we focus here on “single parameter” profiles, where the information associated with each vertex relates to a single degree in its neighborhood. The first natural problem in this direction concern the *maximum* degrees in the vertex neighborhoods. For each vertex i , let d_i denote the maximum vertex degree in i ’s neighborhood. Then $\text{MAXNDEG}(G) = (d_1, \dots, d_n)$ is the maximum neighborhood degree profile of G . The same realizability questions asked above for degree sequences can be posed for neighborhood degree profiles as well. This brings us to the following central question of our work:

MAXIMUM NEIGHBORHOOD DEGREE REALIZATION

Input: A sequence $D = (d_1, \dots, d_n)$ of non-negative integers.

Question: Is there a graph G of size n such that the *maximum* degree in the neighborhood of i -th vertex in G is exactly equal to d_i ?

Our Contributions. We now discuss our contributions in detail. For simplicity, we represent the input vector D alternatively in a more compact format as $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$, where n_i ’s are positive integers with $\sum_{i=1}^\ell n_i = n$; here the specification requires that G contains exactly n_i vertices whose maximum degree in neighborhood is d_i . We may assume that $d_\ell > d_{\ell-1} > \dots > d_1 \geq 1$ (noting that vertices with max degree zero are necessarily singletons and can be handled separately).

We perform an extensive study of maximum neighborhood degree profiles.

1. We obtain the necessary and sufficient conditions for $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ to be MAXNDEG realizable for closed neighborhoods in Section 3. For general graphs we obtain the following characterization.

$$d_\ell \leq n_\ell - 1, \text{ and } (d_1 \geq 2 \text{ or } n_1 \text{ is even})$$

We also study the version of the problem in which the realization is required to be connected. Our characterization is as follows.

$$d_\ell \leq n_\ell - 1, \text{ and } (d_1 \geq 2 \text{ or } \sigma = (1^2))$$

2. Next, we consider the open neighborhoods, wherein a vertex is not counted in its own neighborhood. These are more involved, and are discussed in Section 4. Our results for open neighborhood are summarised in Table 1.

■ **Table 1** Max-neighboring-degree realizability for open neighborhood.

Graph	Complete characterisation
Connected Graphs	$d_\ell \leq \min\{n_\ell, n - 1\}$ $d_1 \geq 2$ or $\sigma = (d^d, 1^1)$ or $\sigma = (1^2)$ $\sigma \neq (d_\ell^{d_\ell+1}, 2^1)$
General graphs	σ can be split ¹ into two profiles σ_1 and σ_2 such that (i) σ_1 has a <i>connected</i> MAXNDEG-open realization, and (ii) $\sigma_2 = (1^{2\alpha})$ or $\sigma_2 = (d^d, 1^{2\alpha+1})$, for integers $d \geq 2, \alpha \geq 0$.

3. Enumerating realizable maximum neighborhood degree profiles: The simplicity of above characterizations enables us to enumerate and count the number of realizable profiles. This gives a way to sample uniformly a random MAXNDEG realizable profile. In contrast, counting and sampling are open problems for the traditional degree sequence realizability problem. In the full version of this paper, we show that the number of realizable profiles of length n is $\lceil (2^{n-1} + (-1)^n)/3 \rceil$ for general graphs and 2^{n-3} for connected graphs. In comparison, the total number of non-increasing sequences of length n on the numbers $1, \dots, n - 1$ is $\Theta(4^n/\sqrt{n})$.

Through this work, we make the first crucial step by obtaining a closed form characterization for maximum-neighborhood-degree profiles, and solving the realization problem for such profiles with an efficient algorithm. As was done with degree sequences, we solve the problem for both *connected* graphs as well as *general* graphs. Our conditions are verifiable in linear time and our realizations are computable in polynomial time. Finally, in contrast to the degree-sequence case, we are able to count the number of distinct realizable maximum-neighborhood-degree sequences.

Further Related Work. Many works have addressed related questions such as finding all the (non-isomorphic) graphs that realize a given degree sequence, counting all the (non-isomorphic) realizing graphs of a given degree sequence, sampling a random realization for a given degree sequence as uniformly as possible, or determining the conditions under which

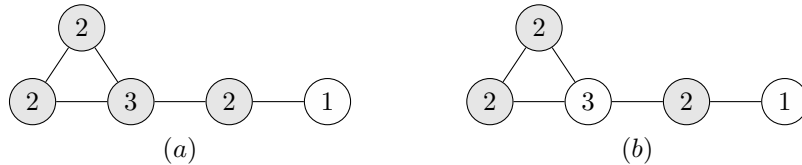
¹ A profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ is said to be split into two profiles $\sigma_1 = (d_\ell^{p_\ell}, \dots, d_1^{p_1})$ and $\sigma_2 = (d_\ell^{q_\ell}, \dots, d_1^{q_1})$ if $n_i = p_i + q_i$ for each $i \in [1, \ell]$.

a given degree sequence defines a unique realizing graph (a.k.a. the *graph reconstruction* problem), see [6, 8, 10, 11, 12, 13, 14, 15, 16, 19, 20, 21, 22, 24]. Other works such as [7, 9, 17] studied interesting applications in the context of social networks.

To the best of our knowledge, the MAXNDEG realization problems have not been explored so far. There are only two related problems that we are aware of. The first is the *shotgun assembly* problem [18], where the characteristic associated with the vertex i is some description of its neighborhood up to radius r . The second is the *neighborhood degree lists* problem [5], where the characteristic associated with the vertex i is the list of degrees of all vertices in i 's neighborhood. We point out that in contrast to these studies, our MAXNDEG problem applies to a more restricted profile (with a single number characterizing each vertex), and the techniques involved are totally different from those of [5, 18]. Several other realization problems are surveyed in [2, 4].

2 Preliminaries

Let H be an undirected graph. We use $V(H)$ and $E(H)$ to respectively denote the vertex set and the edge set of graph H . For a vertex $x \in V(H)$, let $deg_H(x)$ denote the degree of x in H . Let $N_H[x] = \{x\} \cup \{y \mid (x, y) \in E(H)\}$ be the (closed) neighborhood of x in H . For a set $W \subseteq V(H)$, we denote by $N_H(W)$, the set of all the vertices lying outside set W that are adjacent to some vertex in W , that is, $N_H(W) = (\bigcup_{w \in W} N[w]) \setminus W$. Given a vertex v in H , the maximum degree in the neighborhood of v , namely $MAXNDEG_H(v)$, is defined to be the maximum over the degrees of all the vertices in the neighborhood of v . Similarly, the maximum degree in the open neighborhood $(N_H[v] \setminus v)$ of vertex v , namely $MAXNDEG^-_H(v)$ is the maximum over the degrees of all the vertices present in the open neighborhood of v . Given a set of vertices A in a graph H , we denote by $H[A]$ the subgraph of H induced by the vertices of A . For a set A and a vertex $x \in V(H)$, we denote by $A \cup x$ and $A \setminus x$, respectively, the sets $A \cup \{x\}$ and $A \setminus \{x\}$. When the graph is clear from context, for simplicity, we omit the subscripts H in all our notations. Finally, given two integers $i \leq j$, we define $[i, j] = \{i, i + 1, \dots, j\}$.



■ **Figure 1** A comparison of the MAXNDEG realization of $(3^4, 2^1)$ and a $MAXNDEG^-$ realization of $(3^3, 2^2)$.

Next we formally define the realizable profiles.

► **Definition 1.** A profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ satisfying $d_\ell > d_{\ell-1} > \dots > d_1 > 0$ is said to be $MAXNDEG$ realizable if there exists a graph G on $n = n_1 + \dots + n_\ell$ vertices that for each $i \in [1, \ell]$ contains exactly n_i vertices whose $MAXNDEG$ is d_i . Equivalently, $|\{v \in V(G) : MAXNDEG(v) = d_i\}| = n_i$.

► **Definition 2.** A profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ is said to be $MAXNDEG^-$ realizable if there exists a graph G on $n = n_1 + \dots + n_\ell$ vertices that for each $i \in [1, \ell]$ contains exactly n_i vertices whose $MAXNDEG^-$ is d_i . Equivalently, $|\{v \in V(G) : MAXNDEG^-(v) = d_i\}| = n_i$.

The figure depicts a MAXNDEG realization of $(3^4, 2^1)$. (The numbers in the vertices represent their degrees.) Note that in the open neighborhood model, the corresponding MAXNDEG⁻ profile becomes $(3^3, 2^2)$.

3 Realizing maximum neighborhood degree profiles

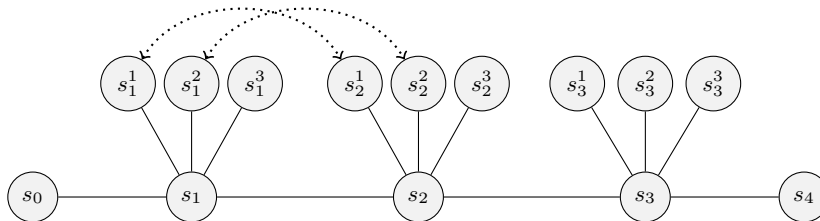
In this section, we provide a complete characterization of MAXNDEG profiles. For simplicity, we first discuss the uniform scenario of $\sigma = (d^k)$. Observe that a star graph $K_{1,d}$ is MAXNDEG realization of the profile (d^{d+1}) . We show in the following lemma that, by identifying together vertices in different copies of $K_{1,d}$, it is always possible to realize the profile (d^k) , whenever $k \geq d + 1$.

► **Lemma 3.** *For any positive integers d and k , the profile $\sigma = (d^k)$ is MAXNDEG realizable whenever $k \geq d + 1$. Moreover, we can always compute in $O(k)$ time a connected realization that has an independent set, say S , of size d such that all vertices in S have degree at most 2, and at least two vertices in S have degree 1.*

Proof. Let α be the smallest integer such that $k \leq 2 + \alpha(d - 1)$. We first construct a caterpillar² T as follows. Take a path $P = (s_0, s_1, \dots, s_\alpha, s_{\alpha+1})$ of length $\alpha + 1$. Connect each internal vertex s_i (here $i \in [1, \alpha]$) with a set of $d - 2$ new vertices, so that the degree of s_i is d . (See Figure 2). Note that the MAXNDEG of each vertex $v \in T$ is d .

Now if $k = 2 + \alpha(d - 1)$, then T serves as our required realizing graph. If $k < 2 + \alpha(d - 1)$, then $\alpha \geq 2$ since $k \geq d + 1$. The tree T is “almost” a realizing graph for the profile, except that it has too many vertices. Let $r = 2 + \alpha(d - 1) - k$ denote the number of excess vertices in T that need to be removed. The r vertices can be removed as follows. Take any two distinct internal vertices s_i and s_j on P , and let s_i^1, \dots, s_i^{d-2} and s_j^1, \dots, s_j^{d-2} , respectively, denote the neighbors of s_i and s_j not lying on P . Let G be the graph obtained by merging vertices s_i^ℓ and s_j^ℓ into a single vertex for $\ell \in [1, r]$. (See Figure 2). Since the number of vertices was decreased by r , G now contains exactly n vertices. The degree of vertices $s_1, s_2, \dots, s_\alpha$ remains d , and the degree of all other vertices is at most 2, therefore $\text{MAXNDEG}(v) = d$ for each $v \in G$, so G is a realization of the profile σ .

Finally, in the resultant graph G , the end points of P (i.e. s_0 and $s_{\alpha+1}$) have degree 1, and there are $d - 2$ other vertices, namely s_i^1, \dots, s_i^{d-2} (or s_j^1, \dots, s_j^{d-2}), that have degree bounded by 2. Therefore we set S to these d vertices. It is easy to verify that S is indeed an independent set. ◀



■ **Figure 2** A caterpillar for $d = 5$ and $\alpha = 3$. If $k = 12$, then $r = 2$, and we merge (i) s_1^1, s_2^1 , and (ii) s_1^2, s_2^2 .

² A caterpillar is a tree in which all the vertices are within distance one of a central path.

3.1 An incremental procedure for computing MaxNDeg realizations

We explain here our main building block, procedure `ADDLAYER`, that will be useful in incrementally building graph realizations in a decreasing order of maximum degrees. Given a partially computed connected graph H and integers d and k satisfying $d \geq 2$ and $k \geq 1$, the procedure adds to H a set W of k new vertices such that $\text{MAXNDEG}(w) = d$, for each $w \in W$. The reader may assume that $\text{MAXNDEG}(v) \geq d$, for each existing vertex $v \in V(H)$. The procedure takes in as an input a sufficiently large vertex list L (of size $d - 1$) that forms an independent set in H , and whose vertices have small degree (that is, at most $d - 1$). Moreover, in order to accommodate its iterative use, each invocation of the procedure also generates and outputs a *new* list, to be used in the further iterations.

Procedure AddLayer. The input to procedure `ADDLAYER` (H, L, k, d) is a connected graph H and a list $L = (a_1, \dots, a_{d-1})$ of vertices in H whose degree is bounded above by $d - 1$. The first step is to add to H a set of k new vertices $W = \{w_1, w_2, \dots, w_k\}$. Next, the new vertices are connected to the vertices of L and to themselves so as to ensure that $\text{MAXNDEG}(w) = d$ for every $w \in W$. Depending upon whether or not $k < d$, there are two separate cases. (Refer to Algorithm 1 for pseudocode).

■ **Algorithm 1** `ADDLAYER` (H, L, k, d).

```

1 Let the list  $L$  be  $(a_1, a_2, \dots, a_{d-1})$ .
2 Add to  $H$  a set  $W = \{w_1, \dots, w_k\}$  of  $k$  new vertices.
3 case ( $k < d$ ) do
4   Set  $count = k$  and  $i = d - 1$ .
5   while ( $count \neq 0$ ) do
6     Let  $r = \min\{d - \text{deg}(a_i), count\}$ .
7     Add edges  $(a_i, w_{count-t})$  to  $H$  for  $t \in [0, r - 1]$ .
8     Decrement  $i$  by 1 and  $count$  by  $r$ .
9   foreach  $j \in [d - 1, \dots, 2, 1]$  do
10    If  $\text{deg}(a_i) = d$  then break the for loop.
11    If  $(j < i)$  then add edge  $(a_j, a_i)$  to  $H$ .
12    If  $(j > i)$  then add an edge between  $a_i$  and an arbitrary vertex in  $N(a_j) \cap W$ .
13  Set  $L$  to be prefix of  $(w_1, w_2, \dots, w_k, a_1, a_2, \dots, a_{i-1})$  of size  $d - 2$ .
14 case ( $k \geq d$ ) do
15  Use Lemma 3 to compute over independent set  $(W \cup \{a_1\})$  the graph, say  $\bar{H}$ ,
    realizing the profile  $(d^{k+1})$  such that  $\text{deg}_{\bar{H}}(a_1) = 1$ .
16  Add edges between  $a_1$  and any arbitrary  $d - \text{deg}(a_1)$  vertices in set
     $\{a_2, a_3, \dots, a_{d-1}\}$ .
17  Let  $b_1, \dots, b_{d-1} \in \bar{H} \setminus a_1$  be such that  $1 = \text{deg}_{\bar{H}}(b_1) \leq \dots \leq \text{deg}_{\bar{H}}(b_{d-1}) \leq 2$ .
18  Set  $L = (b_1, b_2, \dots, b_{d-2})$ .
19 Output  $L$ .
```

Let us first consider the case $k \leq d - 1$. In this case we add edges from vertices in W to a subset of vertices from L such that those vertices in L will have degree d and therefore will imply $\text{MAXNDEG}(w) = d$, for every $w \in W$. We initialize two variables, *count* and *i*, respectively, to k and $d - 1$. The variable *count* holds, at any instant of time, the number of vertices in W that still need to be connected to vertices in L . While $count > 0$, the procedure performs the following steps:

- (i) compute $r = \min\{d - \deg(a_i), \text{count}\}$, the maximum number of vertices in W that can be connected to vertex a_i ;
- (ii) connect a_i to following r vertices in W : $w_{\text{count}-(r-1)}, w_{\text{count}-(r-2)}, \dots, w_{\text{count}-1}, w_{\text{count}}$; and
- (iii) decrease count by r , and i by 1.

When $\text{count} = 0$, the vertices $a_i, a_{i+1}, \dots, a_{d-1}$ are connected to at least one vertex in W (this implies $d - i \leq k$). It is also easy to verify that at this stage, $\deg(a_{d-1}) = \deg(a_{d-2}) = \dots = \deg(a_{i+1}) = d$, and $\deg(a_i) \leq d$. Since the input graph H was connected, in the beginning of the execution $\deg(a_i) \geq 1$, and by connecting a_i to at least one vertex in W , specifically to w_1 , its degree is increased at least by one. So at most $d - 2$ edges need to be added to a_i to ensure that its degree is exactly d . The procedure performs the following operation for each $j \in [d - 1, d - 2, \dots, 2, 1]$ (in the given order) until $\deg(a_i) = d$:

- (i) if $j < i$ then add edge (a_j, a_i) to H , and
- (ii) if $j > i$ then add an edge between a_i and an arbitrary neighbor of a_j lying in W .

Since $\deg(a_i) = \deg(a_{i+1}) = \dots = \deg(a_{d-1}) = d$, and $\deg(w) \leq 2$ for every $w \in W$, it follows that $\text{MAXNDEG}(w) = d$, for each $w \in W$. In the end, we set a *new* list L containing the first $d - 2$ vertices in the sequence $(w_1, w_2, \dots, w_k, a_1, a_2, \dots, a_{i-1})$. This is possible since $k + i - 1 \geq d - 2$ due to the fact that $d - i \leq k$. (Later on we bound the degrees of the vertices in the new list.)

Now we consider the case $k \geq d$. The procedure uses Lemma 3 to compute over the independent set $W \cup \{a_1\}$ a graph \bar{H} realizing the profile (d^{k+1}) such that $\deg_{\bar{H}}(a_1) = 1$. Notice that in the beginning of the execution, $\deg(a_1) \in [1, d - 1]$, and it is increased by one by adding \bar{H} over the set $W \cup \{a_1\}$. So now $\deg(a_1) \in [2, d]$. To ensure $\deg(a_1) = d$, at most $d - 2$ more edges need to be added to a_1 . Edges are added between a_1 and any arbitrary $d - \deg(a_1)$ vertices in set $\{a_2, a_3, \dots, a_{d-1}\}$. This ensures that every $w \in W$ has $\text{MAXNDEG}(w) = d$. By Lemma 3, $\bar{H} \setminus \{a_1\}$ contains an independent set of $d - 1$ vertices, say b_1, \dots, b_{d-1} , such that $1 = \deg_{\bar{H}}(b_1) \leq \deg_{\bar{H}}(b_2) \leq \dots \leq \deg_{\bar{H}}(b_{d-1}) \leq 2$. In the end, the procedure creates a *new* list $L = (b_1, b_2, \dots, b_{d-2})$.

For sake of better understanding, in the rest of paper, we denote by $H_{\text{old}}, L_{\text{old}}$ and $H_{\text{new}}, L_{\text{new}}$ respectively the graph and the list before and after the execution of Procedure ADDLAYER. Observe that $V(H_{\text{new}}) = V(H_{\text{old}}) \cup W$.

The following two lemmas follow from the description of algorithm.

► **Lemma 4.** *Each $w \in W$ satisfies $\text{MAXNDEG}(w) = d$, and $N(w) \subseteq W \cup L_{\text{old}}$.*

► **Lemma 5.** *Each $a \in L_{\text{old}} \setminus L_{\text{new}}$ satisfies $\deg_{H_{\text{new}}}(a) \leq d$, and each $a \in L_{\text{old}} \cap L_{\text{new}}$ satisfies $\deg_{H_{\text{new}}}(a) \leq \deg_{H_{\text{old}}}(a) + 1$.*

It is also easy to verify that the total execution time of Procedure ADDLAYER is $O(k + d)$.

The Inheritance Property. Till now, we showed that given an independent list of $d - 1$ vertices of degree at most $d - 1$ in a graph H , we can add $k \geq 1$ vertices to H such that the MAXNDEG of these k vertices is d . In order to iteratively use this algorithm to add vertices of smaller MAXNDEG values ($\leq d$) we require that the list L_{new} computed by Procedure ADDLAYER should satisfy following three constraints:

- (i) The size of L_{new} should be $d - 2$;
- (ii) the vertices of L_{new} should form an independent set; and most importantly,
- (iii) the vertices in L_{new} should have degree at most $d - 2$.

In order to ensure these constraints on L_{new} , we further impose the constraint that the list L_{old} is a valid list; this is formally defined as below.

► **Definition 6** (Valid List). *A list $L = (a_1, a_2, \dots, a_t)$ in a graph G is said to be “valid” with respect to G if the following two conditions hold:*

- (i) *for each $i \in [1, t]$, $\deg(a_i) \leq i$, and*
- (ii) *the vertices of L form an independent set in G .*

We next prove the inheritance property of our procedure.

► **Lemma 7** (Inheritance property). *If the input list L_{old} in Procedure ADDLAYER is valid, then the output list L_{new} is valid as well.*

Proof. We first consider the case $k \leq d - 1$. Let i be the smallest index such that vertices $a_i, a_{i+1}, \dots, a_{d-1}$ are adjacent to some vertex of W in H_{new} . (That is, i is the index when Procedure ADDLAYER exits the while loop). Recall that in the graph H_{new} , $w_1 \in W$ is a neighbor of a_i . Also, to increase the degree of a_i to d , we connect a_i to some/all vertices in a_1, \dots, a_{i-1} , and some/all neighbors of a_{i+1}, \dots, a_{d-1} lying in W . Therefore the vertex set $W \cup \{a_1, \dots, a_{i-1}\}$ is independent in H_{new} . Also, its size at least $d - 1$, as we showed that $k \geq d - i$. Since the list $L_{old} = (a_1, a_2, \dots, a_{d-1})$ is valid in the beginning of the execution of Procedure ADDLAYER, it follows that in H_{old} , $\deg(a_j) \leq j$ for $j \in [1, d - 1]$. So by Lemma 5, in H_{new} , (i) $\deg(a_j) \leq j + 1$ for $j \in [1, i - 1]$, (ii) $\deg(w_1) = 1$, and (iii) the degree of each other vertex in $W \setminus w_1$ is at most 2. Consequently, (w_1, \dots, w_k) is a valid list of length at least $d - i \geq 1$. Since $\deg(a_j) \leq j + 1$ for $j \in [1, i - 1]$, the list $(w_1, \dots, w_k, a_1, \dots, a_{i-1})$ is valid and has length at least $d - 1$. Truncating this to length $d - 2$ again gives us a valid list.

We now consider the case $k \geq d$. By Lemma 3, $H[W \cup \{a_1\}] = \bar{H}$ contains an independent set $\{b_1, b_2, \dots, b_{d-1}\} \subseteq W$ such that $\deg(b_1) = 1$ and $\deg(b_j) \leq 2$ for $j \in [2, d - 1]$. Therefore, $(b_1, b_2, \dots, b_{d-2})$ is a valid list of length $d - 2$ in H_{new} . ◀

The following proposition summarizes the above discussion.

► **Proposition 8.** *For any integers $d \geq 2$, $k \geq 1$, and any connected graph H containing a valid list L of size $d - 1$, procedure ADDLAYER adds to H in $O(k + d)$ time, a set W of k new vertices such that $\text{MAXNDEG}(w) = d$, for every $w \in W$. All the edges added to H lie in $W \times (W \cup L)$. Moreover, $\deg_H(a) \leq d$, for every $a \in L$, and the updated graph remains connected and contains a new valid list of size $d - 2$.*

3.2 The main algorithm

We now present the main algorithm for computing the realizing graph using Procedure ADDLAYER.

Let $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ be any profile satisfying $d_\ell \leq n_\ell - 1$ and $d_1 \geq 2$. The construction of a connected graph realizing σ is as follows (refer to Algorithm 2 for pseudocode). We first use Lemma 3 to initialize G to be the graph realizing the profile $(d_\ell^{n_\ell})$. Recall G contains an independent set, say $W = \{w_1, w_2, \dots, w_{d_\ell}\}$, satisfying the condition that the degree of the first two vertices is one, and the degree of the remaining vertices is at most two. Set $L_{\ell-1} = (w_1, w_2, \dots, w_{d_{\ell-1}-1})$ (notice that $d_{\ell-1} - 1 \leq d_\ell$). It is easy to verify that this list is valid. Next, for each $i = \ell - 1$ to 1, perform the following steps:

- (i) Taking as input the valid list L_i of size $d_i - 1$, execute Procedure ADDLAYER (G, L_i, n_i, d_i) to add n_i new vertices to G . The procedure returns a valid list L_{i-1} of size $d_i - 2$.
- (ii) Truncate the list L_{i-1} to contain only the first $d_{i-1} - 1 (\leq d_i - 2)$ vertices. The truncated list remains valid since any prefix of a valid list is valid.

■ **Algorithm 2** MAXNDEG realization of $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$.

Input: A sequence $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ satisfying $d_\ell \leq n_\ell - 1$ and $d_1 \geq 2$.

- 1 Initialize G to be the graph obtained from Lemma 3 that realizes the profile $(d_\ell^{n_\ell})$.
 - 2 Let $L_{\ell-1}$ be a valid list in G of size $d_{\ell-1} - 1$.
 - 3 **for** ($i = \ell - 1$ to 1) **do**
 - 4 $L_{i-1} \leftarrow \text{ADDLAYER}(G, L_i, n_i, d_i)$.
 - 5 Truncate list L_{i-1} to contain only the first $d_{i-1} - 1 (\leq d_i - 2)$ vertices.
 - 6 Output G .
-

Proof of Correctness. Let V_ℓ denote the set of vertices in graph G initialized in step 1, and for $i \in [1, \ell - 1]$, let V_i denote the set of n_i new vertices added to graph G in iteration i of the for loop. Also for $i \in [1, \ell]$, let G_i be the graph induced by vertices $V_i \cup \dots \cup V_\ell$. The following lemma proves the correctness.

► **Lemma 9.** For any $i \in [1, \ell]$, graph G_i is a MAXNDEG realization of profile $(d_\ell^{n_\ell}, \dots, d_i^{n_i})$, and for any $j \in [i, \ell]$ and any $v \in V_j$, $\text{deg}_{G_i}(v) \leq \text{MAXNDEG}_{G_i}(v) = d_j$.

Proof. We prove the claim by induction on the iterations of the for loop. The base case is for index ℓ , and by Lemma 3 we have that $\text{deg}_{G_\ell}(v) \leq \text{MAXNDEG}_{G_\ell}(v) = d_\ell$, for every $v \in V_\ell$. For the inductive step, we assume that the claim holds for $i + 1$, and prove the claim for i . Consider any vertex v in G_i . We have two cases.

1. $v \in V_i$: In this case by Proposition 8 we have that $\text{deg}_{G_i}(v) \leq \text{MAXNDEG}_{G_i}(v) = d_i$.
2. $v \in V_j$, for $j > i$: We first show that for any vertex $w \in N_{G_i}[v]$, $\text{deg}_{G_i}(w) \leq d_j$. If $w \in V_i$, then we already showed $\text{deg}_{G_i}(w) \leq d_i$. So let us consider the case $w \in V_{i+1} \cup \dots \cup V_\ell$. Now if $w \in L_i$ participates in Procedure ADDLAYER (G, L_i, n_i, d_i) , then by Proposition 8, in the updated graph $\text{deg}_{G_i}(w) \leq d_i \leq d_j$. If $w \notin L_i$, then the degree of w is unaltered in the i^{th} iteration, and thus $\text{deg}_{G_i}(w) = \text{deg}_{G_{i+1}}(w) \leq \text{MAXNDEG}_{G_{i+1}}(v) = d_j$ by the inductive hypothesis. It follows that $\text{MAXNDEG}(v)$ remains unaltered due to iteration i , and thus $\text{MAXNDEG}_{G_i}(v) = \text{MAXNDEG}_{G_{i+1}}(v) = d_j$. ◀

The execution time of the algorithm is $O(\sum_{i=1}^{\ell} (n_i + d_i))$. This is also optimal. Indeed, any connected graph realizing σ must contain $\Omega(n_1 + n_2 + \dots + n_\ell)$ edges as the degrees of all vertices must be non-zero. Also, the graph must contain at least one vertex of each of the degrees d_1, d_2, \dots, d_ℓ , and therefore must have $\Omega(d_1 + d_2 + \dots + d_\ell)$ edges. In other words, any realizing graph must contain $\Omega(\sum_{i=1}^{\ell} (n_i + d_i))$ edges, and thus the computation time must be at least $\Omega(\sum_{i=1}^{\ell} (n_i + d_i))$. The following theorem is immediate from the above discussions.

► **Theorem 10.** There exists an algorithm that given any profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ satisfying $d_\ell \leq n_\ell - 1$ and $d_1 \geq 2$ computes in optimal time a connected MAXNDEG realization of σ .

3.3 A complete characterization for MaxNDeg realizable profiles

The necessary conditions for MAXNDEG realizability is as follows.

► **Lemma 11.** A necessary condition for a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ to be MAXNDEG realizable is $d_\ell \leq n_\ell - 1$.

10:10 Graph Realizations: Maximum Degree in Vertex Neighborhoods

Proof. Suppose σ is MAXNDEG realizable by a graph G . Then G must contain a vertex, say w , of degree d_ℓ in G . Since d_ℓ is the maximum degree in G , the MAXNDEG of all the $d_\ell + 1$ vertices in $N[w]$ must be d_ℓ . Thus $n_\ell \geq d_\ell + 1$. \blacktriangleleft

Consider a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ realizable by a connected graph. If $d_1 = 1$, then the graph must contain a vertex, say v , of degree 1, and the vertices in $N[v]$ must also have degree 1. The only possibility for such a graph is a single edge graph on two vertices. Thus in this case $\sigma = (1^2)$. If $d_1 \geq 2$, then by Lemma 11, for σ to be realizable in this case we need that $n_\ell \geq d_\ell + 1$. Also, by Theorem 10, under these two conditions σ is always realizable. We thus have the following theorem.

► **Theorem 12.** *For a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ to be MAXNDEG realizable by a connected graph the necessary and sufficient condition is that either*

- (i) $n_\ell \geq d_\ell + 1$ and $d_1 \geq 2$, or
- (ii) $\sigma = (1^2)$.

Now if $d_1 = 1$, then n_1 must be even, since the vertices v with $\text{MAXNDEG}(v) = 1$ must form a disjoint union of exactly $n_1/2$ edges. So for general graphs we have the following theorem.

► **Theorem 13.** *For a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ to be MAXNDEG realizable by a general graph the necessary and sufficient conditions are that $d_\ell \geq n_\ell - 1$, and either n_1 is even or $d_1 \geq 2$.*

3.4 Discussion

We briefly discuss the reasons behind the intractability in our incremental construction. Let us consider the MAXNDEG profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ for a graph $G = (V, E)$. For $1 \leq i \leq \ell$, let $W_i \subseteq V$ be the set of vertices whose MAXNDEG in G is at least d_i . Note that for any vertex $v \in W_i$, a vertex having maximum degree in $N_G[v]$ (say x) must be contained in W_i . Moreover, all the neighbors of x must also lie in W_i . It follows that the degree of x remains unaltered when restricted to the induced subgraph $G[W_i]$, and $\text{MAXNDEG}_G(v) = \text{MAXNDEG}_{G[W_i]}(v)$. Hence, MAXNDEG profiles satisfy the following nice *substructure property*, which justifies our incremental algorithm for computing their realizations.

► **Substructure Property.** *The induced graph $G_i = G[W_i]$ is a MAXNDEG realization of the partial profile $(d_\ell^{n_\ell}, \dots, d_i^{n_i})$, for each $1 \leq i \leq \ell$.*

Observe that in the case of MAXNDEG^- profiles, unfortunately, the nice *sub-structure property* does not always hold, which in turn increases the complexity of the problem. For example, for the graph considered in Figure 1, the profile $\sigma = (3^3, 2^2)$ is MAXNDEG^- realizable, however, the subsequence (3^3) is not MAXNDEG^- realizable.

4 Realizing maximum open neighborhood-degree profiles

4.1 Pseudo-valid List

We begin by stating the following lemmas that are an extension of Lemma 3 and Proposition 8 presented in Section 3 for MAXNDEG profiles.

► **Lemma 14.** *For any positive integers d and k , the profile $\sigma = (d^k)$ is MAXNDEG^- realizable whenever $k \geq d + 2$. Moreover, we can always compute in $O(k)$ time a connected realization that contains an independent set having*

- (i) *two vertices of degree 1, and*
- (ii) *$d - 2$ other vertices of degree at most 2.*

► **Proposition 15.** *For any integers $d \geq 2$, $k \geq 1$, and any connected graph H containing a valid list L of size $d - 1$, procedure ADDLAYER adds to H in $O(k + d)$ time, a set W of k new vertices such that $\text{MAXNDEG}^-(w) = d$, for every $w \in W$. All the edges added to H lie in $W \times (W \cup L)$. Moreover, $\deg_H(a) \leq d$, for every $a \in L$, and the updated graph remains connected and contains a new valid list of size $d - 2$.*

It is important to note that though the Proposition 15 holds for the open-neighborhoods it can not be directly used to incrementally compute the realizations. This is because for the profiles $\sigma = (d_\ell^{d_\ell+1})$ unlike the scenario of MAXNDEG realization, there is no MAXNDEG^- realization that contains a valid list (See Lemma 18 for further details). This motivates us to define pseudo-valid lists.

► **Definition 16.** *A list $L = (a_1, a_2, \dots, a_t)$ in a graph H is said to be “pseudo-valid” with respect to H if*

- (i) *for each $i \in [1, t]$, $\deg(a_i) = 2$, and*
- (ii) *the vertices of L form an independent set.*

Note that the only deviation that prevents L from being a valid list is that $\deg(a_1)$ is 2 instead of 1.

We next state two lemmas that are crucial in obtaining MAXNDEG^- realizations in the scenarios $n_\ell = d_\ell$ and $n_\ell = d_\ell + 1$.

► **Lemma 17.** *For any integers $d > \bar{d} \geq 2$, the profile $\sigma = (d^d, \bar{d}^1)$ is MAXNDEG^- realizable. Moreover, in $O(d)$ time we can compute a connected realization that contains a valid list of size $d - 1$.*

Proof. The construction of G is as follows. Take a vertex z and connect it to $d - 1$ other vertices v_1, \dots, v_{d-1} . Next take another vertex y and connect to $v_1, \dots, v_{\bar{d}-1}$ (recall $2 \leq \bar{d} < d$). Also connect z to y . In the resulting graph G , $\deg(z) = d$, $\deg(y) = \bar{d}$, and $\deg(v_i) \leq 2$ for $i \in [1, d - 1]$. Also, v_{d-1} is not adjacent to y as $\bar{d} < d$, thus $\deg(v_{d-1}) = 1$. Therefore, $\text{MAXNDEG}^-(z) = \bar{d}$, $\text{MAXNDEG}^-(y) = d$, and $\text{MAXNDEG}^-(v_i) = d$, for $i \in [1, d - 1]$. It is also easy to verify that $(v_{d-1}, \dots, v_{\bar{d}-1}, \dots, v_2, v_1)$ is a valid list in G . ◀

► **Lemma 18.** *For any integer $d \geq 2$, the profile $\sigma = (d^{d+1})$ is MAXNDEG^- realizable. Moreover, a connected realization that contains an independent set having $d - 1$ vertices of degree 2 can be compute in $O(d)$ time. However, none of the graphs realizing σ can contain a vertex of degree 1.*

Proof. The construction of graph G realizing σ is very similar to the previous lemma. Take two vertex-sets, namely, $U = \{u_1, u_2\}$ and $W = \{w_1, \dots, w_{d-1}\}$. Add to G the edge (u_1, u_2) , and for each $i \in [1, d - 1]$, add to G the edges (u_1, w_i) and (u_2, w_i) . This ensures that $\deg(u_1) = \deg(u_2) = d$ and $\deg(w_i) = 2$ for $i \in [1, d - 1]$. So G contains $d + 1$ vertices with MAXNDEG^- equal to d . Also, W is an independent set of size $d - 1$ in G and $\deg(w_i) = 2$, for every vertex $w_i \in W$.

10:12 Graph Realizations: Maximum Degree in Vertex Neighborhoods

Next, let H be any MAXNDEG^- realizing graph of σ . Then H must contain two vertices, say x and y , of degree d , since a single vertex of degree d in H can guarantee $\text{MAXNDEG}^- = d$ for at most d vertices. Next notice that $N[x] = N[y]$, because otherwise H will contain more than $d + 1$ vertices. This implies that all the vertices in H , other than x and y , are adjacent to both x and y . Therefore, each of the vertices in H must have degree at least two. ◀

The next lemma shows that ADDLAYER outputs a valid list, even when the input list is pseudo-valid.

► **Lemma 19.** *In procedure ADDLAYER , the list L_{new} is valid even when the list L_{old} is pseudo-valid and the parameter d satisfies $d \geq 3$.*

Proof. We borrow notations from the proof of Lemma 7. As before, we have two separate cases depending on whether or not $k < d$. We first consider the case $k \leq d - 1$. We showed in Lemma 7 that $(w_1, \dots, w_k, a_1, \dots, a_{i-1})$ is a valid list of length at least $d - 1$ when $\text{deg}_{H_{\text{old}}}(a_1) = 1$. We now consider the scenario when L_{old} is pseudo-valid, and $\text{deg}_{H_{\text{old}}}(a_1) = 2$. The list L_{new} is still valid if $k \geq 2$, since the degree of a_1 in H_{new} is at most 3 and its position in L_{new} is also 3 or greater. So the non-trivial case is $k = 1$. In such a case $i = d - 1$, as the only vertex w_1 belonging to W is connected to a_{d-1} in Algorithm 1. Also, $\text{deg}_{H_{\text{old}}}(a_{d-1}) = 2$, and a_{d-1} is connected to vertex w_1 , so to ensure that $\text{deg}(a_{d-1}) = d$, in the for loop in step 9 of Algorithm 1, it is connected to only $d - 3$ vertices, namely, a_2, a_3, \dots, a_{d-2} . Since a_{d-1} is never connected to vertex a_1 , $\text{deg}_{H_{\text{new}}}(a_1) = \text{deg}_{H_{\text{old}}}(a_1) = 2$. This shows that the sequence $(w_1, \dots, w_k, a_1, \dots, a_{i-1}) = (w_1, a_1, \dots, a_{d-2})$ is a valid list of length exactly $d - 1$. Truncating it to length $d - 2$ again yields a valid sequence. In case $k \geq d$, a_1 's degree does not play any role, so the argument from the proof of Lemma 7 works as is. ◀

► **Remark 20.** The condition $d \geq 3$ is necessary in Lemma 19 because in a pseudo-valid list all the vertices have degree 2. However, Procedure ADDLAYER works only in the case when the degree of each vertex in the list is at most $d - 1$, which does not hold true for a pseudo-valid list when $d = 2$. So we provide a different analysis for the profile $(d^{d+1}, 2^k)$.

4.2 MaxNDeg^- realization of the profile $\sigma = (d^{d+1}, 2^k)$

The following lemmas shows that $\sigma = (d^{d+1}, 2^1)$, for $d \geq 3$, is not MAXNDEG^- realizable when $d \geq 3$; and $\sigma = (d^{d+1}, 2^k)$ is MAXNDEG^- realizable when $d \geq 3$ and $k \geq 2$.

► **Lemma 21.** *For any integer $d \geq 3$, the profile $\sigma = (d^{d+1}, 2^1)$ is not MAXNDEG^- realizable.*

Proof. Let us assume on the contrary that σ is MAXNDEG^- realizable by a graph G , and let $w \in V(G)$ be a vertex such that $\text{MAXNDEG}^-(w) = 2$. The graph G must contain at least two vertices, say x and y , of degree d , since a single vertex of degree d can guarantee MAXNDEG^- of d for at most d vertices in the graph. Consider the following two cases.

- (i) $N[x] = N[y]$: In this case the MAXNDEG^- of all the vertices in $N[x] = N[y]$ is at least $d \geq 3$, as they are adjacent to either x or y . Thus $w \notin N[x]$, which implies that $V(G) = N[x] \cup \{w\}$ since $|N[x]| = d + 1$ and $|V(G)| = d + 2$. Also, w cannot be adjacent to any vertex in $N[x]$, because if w is adjacent to a vertex $w_0 \in N[x]$, then $\text{deg}(w_0)$ must be 3, in contradiction to the assumption $\text{MAXNDEG}^-(w) = 2$. Thus the only possibility left is that w is a singleton vertex, which is again a contradiction.
- (ii) $N[x] \neq N[y]$: In this case the vertex set of G is equal to $N[x] \cup N[y]$ since size of $N[x] \cup N[y]$ must be at least $d + 2$ (as $|N[x] \cap N[y]| \leq d$) and is also at most $|V(G)| = d + 2$. This implies that all the vertices of G are adjacent to either x or y , which contradicts the fact that $\text{MAXNDEG}^-(w) = 2$, since $\text{deg}(x) = \text{deg}(y) = d \geq 3$. ◀

► **Lemma 22.** *For any integers $d \geq 3$ and $k \geq 2$, the profile $\sigma = (d^{d+1}, 2^k)$ is MAXNDEG⁻ realizable. Moreover, we can compute a connected realization in $O(d+k)$ time.*

Proof. The construction of G is as follows. Take a vertex u_1 and connect it to d other vertices v_1, \dots, v_d . Next, take another vertex u_2 and connect it to vertices v_2, \dots, v_d , and a new vertex v_{d+1} . Finally, take a path $(a_1, a_2, \dots, a_\alpha)$ on $\alpha = k - 2$ new vertices, and connect a_1 to v_{d+1} . In the graph G , $\deg(u_1) = \deg(u_2) = d$, and $\deg(v_i), \deg(a_j) \leq 2$, for $i \in [1, d+1]$ and $j \in [1, k-2]$. Vertices u_1 and u_2 has maximum degree in their neighborhood 2, thus $\text{MAXNDEG}^-(u_1) = \text{MAXNDEG}^-(u_2) = 2$. Each v_i is adjacent to u_1, u_2 , for $i \in [1, d+1]$, so its MAXNDEG^- is d . And, the MAXNDEG^- of vertices on the path $(a_1, a_2, \dots, a_\alpha)$ is 2, since they have a neighbor of degree 2. ◀

4.3 Algorithm

We now explain the construction of a graph realizing the profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1}) \neq (d_\ell^{d_\ell+1}, 2^1)$ that satisfies the conditions

- (i) $d_\ell \leq \min\{n_\ell, n-1\}$, and
- (ii) $d_1 \geq 2$,

where $n = n_1 + \dots + n_\ell$. If σ is equal to $(d_\ell^{d_\ell+1}, 2^k)$, for some $k \geq 2$, we use Lemma 22 to realize σ . If not, then depending upon the value of n_ℓ , we initialize G differently as follows (refer to Algorithm 3 for the pseudocode).

1. If $n_\ell \geq d_\ell + 2$, we use Lemma 14 to initialize G to be a MAXNDEG⁻ realization of the profile $(d_\ell^{n_\ell})$. Recall G contains an independent set, say $W = \{w_1, w_2, \dots, w_{d_\ell}\}$, satisfying the condition that the degree of first two vertices is one, and the degree of the remaining vertices is at most two. We set $L_{\ell-1}$ to be the list $(w_1, w_2, \dots, w_{d_\ell-1})$ (notice $d_\ell - 1 < d_\ell$). It is easy to verify that this list is valid.
2. If $n_\ell = d_\ell + 1$, then a realization of $(d_\ell^{d_\ell+1})$ does not contains a valid list. So we use Lemma 18 to initialize G to be a MAXNDEG⁻ realization of the profile $(d_\ell^{d_\ell+1})$ that contains a pseudo-valid list. This is possible since we showed G contains an independent set, say $W = \{w_1, w_2, \dots, w_{d_\ell-1}\}$, such that degree of each $w \in W$ is two. We set $L_{\ell-1}$ to be the list $(w_1, w_2, \dots, w_{d_\ell-1})$ (again notice $d_\ell - 1 < d_\ell - 1$).
3. If $n_\ell = d_\ell$, then the sequence $d_\ell^{d_\ell}$ is not realizable (see Lemma 25). So we initialize G to be the graph realization of $(d_\ell^{n_\ell}, d_{\ell-1})$ as obtained from Lemma 17. We set $L_{\ell-1}$ be a valid list in G of size $d_{\ell-1} - 1$. Also we decrement $n_{\ell-1}$ by one as G already contain a vertex whose MAXNDEG^- is $d_{\ell-1}$.

Next for each $i = \ell - 1$ to 1 we perform following steps.

- (i) We take as an input the valid list L_i of size $d_i - 1$, and execute Procedure ADDLAYER (G, L_i, n_i, d_i) to add n_i new vertices to G . The procedure returns a valid list L_{i-1} of size $d_i - 2$.
- (ii) Truncate list L_{i-1} to contain only the first $d_{i-1} - 1 (\leq d_i - 2)$ vertices. The truncated list remains valid since it is a prefix of a valid list.

Correctness. Let \bar{V}_ℓ denote the set of vertices in graph G initialized in steps 5, 8, or 11 of Algorithm 3, and for $i \in [1, \ell - 1]$, let \bar{V}_i denote the set of new vertices added to graph G in iteration i of for loop. For $i \in [1, \ell]$, let G_i be the graph induced by vertices $\bar{V}_i \cup \dots \cup \bar{V}_\ell$.

Recall that if $n_\ell = d_\ell$, then the graph is initialized in step 11 and contains $n_\ell + 1$ vertices, of which one vertex, say z , has $\text{MAXNDEG}^-(z) = d_{\ell-1}$, and the remaining vertices have $\text{MAXNDEG}^- = d_\ell$. If $n_\ell = d_\ell$, then let $Z = \{z\}$, otherwise let $Z = \emptyset$. We set $V_\ell = \bar{V}_\ell \setminus Z$, $V_{\ell-1} = \bar{V}_{\ell-1} \cup Z$, and $V_i = \bar{V}_i$ for $i \in [1, \ell - 2]$. Thus $|V_i| = n_i$, for $i \in [1, \ell]$. The next lemma proves the correctness.

10:14 Graph Realizations: Maximum Degree in Vertex Neighborhoods

■ **Algorithm 3** MAXNDEG⁻ realization of $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$.

Input: A sequence $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1}) \neq (d_\ell^{d_\ell+1} 2^1)$ satisfying $d_\ell \leq \min\{n-1, n_\ell\}$ and $d_1 \geq 2$.

```

1 if  $\sigma = (d_\ell^{d_\ell+1}, 2^k)$  for some  $k \geq 2$  then
2   | Use Lemma 22 to compute a realization  $G$  for profile  $\sigma$ .
3 else
4   | case  $n_\ell \geq d_\ell + 2$  do
5     | Initialize  $G$  to be the graph obtained from Lemma 14 that realizes the profile
6     |  $(d_\ell^{n_\ell})$ .
7     | Set  $L_{\ell-1}$  to be a valid list in  $G$  of size  $d_{\ell-1} - 1$ .
8   | case  $n_\ell = d_\ell + 1$  do
9     | Initialize  $G$  to be the graph obtained from Lemma 17 that realizes the profile
10    |  $(d_\ell^{d_\ell+1})$ .
11    | Set  $L_{\ell-1}$  to be a pseudo-valid list in  $G$  of size  $d_{\ell-1} - 1$ .
12  | case  $n_\ell = d_\ell$  do
13    | Initialize  $G$  to be the graph obtained from Lemma 18 that realizes the profile
14    |  $(d_\ell^{d_\ell} d_{\ell-1})$ .
15    | Set  $L_{\ell-1}$  to be a valid list in  $G$  of size  $d_{\ell-1} - 1$ .
16    | Decrement  $n_{\ell-1}$  by 1.
17  | for  $(i = \ell - 1$  to 1) do
18    |  $L_{i-1} \leftarrow \text{ADDLAYER}(G, L_i, n_i, d_i)$ .
19    | Truncate list  $L_{i-1}$  to contain only the first  $d_{i-1} - 1 (\leq d_i - 2)$  vertices.
20 Output  $G$ .
```

► **Lemma 23.** For any $i \in [1, \ell]$, graph G_i is a MAXNDEG⁻ realization of profile $(d_\ell^{n_\ell}, \dots, d_i^{n_i})$, except for the case $n_\ell = d_\ell$ in which G_ℓ is MAXNDEG⁻ realization of profile $(d_\ell^{n_\ell}, d_{\ell-1})$. Moreover, for any $j \in [i, \ell]$, we have

1. For every $v \in V_j \setminus Z$, $\deg_{G_i}(v) \leq \text{MAXNDEG}^-_{G_i}(v) = d_j$.
2. If $n_\ell = d_\ell$, then $\deg_{G_i}(z) = d_\ell$ and $\text{MAXNDEG}^-_{G_i}(z) = d_{\ell-1}$.

Proof. We prove the claim by induction on the iterations of the for loop. The base case is for index ℓ , and the claim follows from Lemmas 14, 17, and 18. Specifically, notice that every vertex $v \in V_\ell$ that is included in G in step 5, 8, or 11 of the algorithm has $\text{MAXNDEG}^-(v) = d_\ell$. In the case $n_\ell = d_\ell$, the vertex $z \in V_{\ell-1}$ included in step 11 of algorithm has $\text{MAXNDEG}^-(z) = d_{\ell-1}$. Also, in both the cases, $V_\ell \cup Z$ is the vertex set of G , and degree of all the vertices in this set is bounded by d_ℓ .

For the inductive step, we assume that the claim holds for $i + 1$, and prove the claim for i . Consider any vertex v in G_i . We have two cases.

1. $v \in V_i \setminus Z$: In this case by Proposition 15 and Lemma 19, $\deg_{G_i}(v) \leq \text{MAXNDEG}^-_{G_i}(v) = d_i$.
2. $v \in V_j \setminus Z$, for $j > i$: In this case we first show that for any vertex $w \in N(v)$, $\deg_{G_i}(w) \leq d_j$. If $w \in V_i \setminus Z$, then we already showed $\deg_{G_i}(w) \leq d_i$. So we next consider the case $w \in (V_{i+1} \cup \dots \cup V_\ell) \setminus Z$. Now if $w \in L_i$ participates in Procedure $\text{ADDLAYER}(G, L_i, n_i, d_i)$, then by Proposition 15 in the updated graph $\deg_{G_i}(w) \leq d_i \leq d_j$. If $w \notin L_i$, then the degree of w is unaltered in the i^{th} iteration, and thus $\deg_{G_i}(w) = \deg_{G_{i+1}}(w) \leq d_j$ by the inductive

hypothesis. If $n_\ell = d_\ell$ and $w = z \in Z$, then also $\deg_{G_i}(w) = \deg_{G_{i+1}}(w)$ since vertex z never participates in procedure ADDLAYER. It follows that $\text{MAXNDEG}^-(v)$ remains unaltered due to iteration i , and thus $\text{MAXNDEG}^-_{G_i}(v) = \text{MAXNDEG}^-_{G_{i+1}}(v) = d_j$.

Now when $n_\ell = d_\ell$, then $\deg_{G_\ell}(z) = d_\ell$ and $\text{MAXNDEG}^-_{G_\ell}(z) = d_{\ell-1}$. The degree of vertex z never changes since it does not participate in procedure ADDLAYER. The MAXNDEG^- of z never changes from the same reasoning as above. ◀

The execution time of algorithm takes $O(\sum_{i=1}^{\ell} (n_i + d_i))$ time, which can be easily shown to be optimal. The following theorem is immediate from the above discussions.

► **Theorem 24.** *There exists an algorithm that given any profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1}) \neq (d_\ell^{d_\ell+1}, 2^1)$ with $n = n_1 + \dots + n_\ell$ satisfying $d_\ell \leq \min\{n - 1, n_\ell\}$ and $d_1 \geq 2$, computes in optimal time a connected MAXNDEG^- realization of σ .*

4.4 Complete characterization of MaxNDeg^- profiles.

We first give the sufficient conditions for a profile to be MAXNDEG^- realizable.

► **Lemma 25.** *A necessary condition for the profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ with $n = n_1 + \dots + n_\ell$ to be MAXNDEG^- realizable is $d_\ell \leq \min\{n_\ell, n - 1\}$.*

Proof. Suppose σ is MAXNDEG^- realizable by a graph H . Then there exists at least one vertex, say u , of degree exactly d_ℓ in H . Now $|N(u)| = d_\ell$ and $|N[u]| = d_\ell + 1$, which implies that the number of vertices in H whose MAXNDEG^- is d_ℓ must be at least d_ℓ , so $n_\ell \geq d_\ell$. Also, the number of vertices in the graph H , n , must be at least $d_\ell + 1$. ◀

Consider a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1})$ realizable by a connected graph. If $d_1 = 1$, then the realizing graph must contain a vertex, say u , such that each vertex in $N(u)$ has degree 1. Let $d = \deg(u)$, and v_1, \dots, v_d be the neighbors of u . Then $\deg(v_1) = \dots = \deg(v_d) = 1$. So in this case the realizing graph is a star graph $K_{1,d}$ with MAXNDEG^- profile $\sigma = (d^d, 1^1)$. If $d_1 \geq 2$, then by Lemma 25, for σ to be realizable in this case, we need that $d_\ell \leq \min\{n_\ell, n - 1\}$. Also, Lemma 21 implies that σ must not be $(d^{d+1}, 2^1)$. By Theorem 24, under these conditions σ is always realizable. We thus have the following theorem.

► **Theorem 26.** *The necessary and sufficient condition for a profile $\sigma = (d_\ell^{n_\ell}, \dots, d_1^{n_1}) \neq (d^{d+1}, 2^1)$ with $n = n_1 + \dots + n_\ell$ to be MAXNDEG^- realizable by a connected graph is*

- (i) $d_\ell \leq \min\{n_\ell, n - 1\}$ and $d_1 \geq 2$; or
- (ii) $\sigma = (d^d, 1^1)$ for some positive integer $d > 1$; or
- (iii) $\sigma = (1^2)$.

For general graphs we have the following theorem.

► **Theorem 27.** *The necessary and sufficient condition for a profile σ to be MAXNDEG^- realizable by a general graph is that σ can be split into two profiles σ_1 and σ_2 such that*

- (i) σ_1 has a connected MAXNDEG^- realization, and
- (ii) $\sigma_2 = (1^{2\alpha})$ or $\sigma_2 = (d^d, 1^{2\alpha+1})$ for some integers $d \geq 2, \alpha \geq 0$.

Proof. Suppose σ is realizable by graph G . Let $\mathcal{C}(G)$ be a set consisting of all those components in G that contain a vertex of MAXNDEG^- equal to 1 but is not an edge. As long as $|\mathcal{C}(G)| > 1$, we perform following modifications to G . Take any two components $C_1, C_2 \in \mathcal{C}(G)$, and let σ_1 and σ_2 be their MAXNDEG^- profiles. For $i = 1, 2$, component C_i must be of form K_{1,δ_i} and contain $\delta_i (\geq 2)$ vertices of MAXNDEG^- equal to δ_i , and a single

vertex of MAXNDEG^- equal to 1. Let us assume $\delta_2 \geq \delta_1$. We replace C_1 and C_2 in G by two different components, namely, an edge and (i) a connected MAXNDEG^- realization of profile $\delta_2^{\delta_1 + \delta_2}$ if $\delta_2 = \delta_1$, or (ii) a connected MAXNDEG^- realization of profile $(\delta_2^{\delta_2}, \delta_1^{\delta_1})$ if $\delta_2 > \delta_1$. In each iteration we decrease $|\mathcal{C}(G)|$ by a value two. In the end if $\mathcal{C}(G)$ is non-empty we denote the only component in it by C_0 . Next let $\bar{C}_1, \dots, \bar{C}_k$ be all those components in G that contain only the vertices of MAXNDEG^- strictly greater than 1. Also let $\sigma_1, \dots, \sigma_k$ be their MAXNDEG^- profiles. If $k > 0$, we replace the components $\bar{C}_1, \dots, \bar{C}_k$ by a single connected component, say \bar{C}_0 , that realizes the profile $\sigma_1 + \dots + \sigma_k$. It is easy to verify from Theorem 24 that $\sigma_1 + \dots + \sigma_k$ will be MAXNDEG^- realizable. The final graph G contains (i) at most one component, namely \bar{C}_0 , having all vertices of MAXNDEG^- greater than 1, (ii) at most one component, namely C_0 , having exactly one vertex of MAXNDEG^- equal to 1, and (iii) a union of some $\alpha \geq 0$ disjoint edges. This shows that σ can be split into two profiles σ_1 and σ_2 such that

- (i) σ_1 has a connected MAXNDEG^- realization, and
- (ii) $\sigma_2 = (1^{2\alpha})$ or $\sigma_2 = (d^d, 1^{2\alpha+1})$ for some integers $d \geq 2, \alpha \geq 0$.

To prove the converse notice that $\sigma_2 = (1^{2\alpha})$ is realizable by a disjoint union of $\alpha \geq 0$ edges, and $\sigma_2 = (d^d, 1^{2\alpha+1})$ is realizable by a disjoint union of α edges and the star graph $K_{1,d}$. Thus any σ that can be split into two profiles σ_1 and σ_2 such that

- (i) σ_1 has a connected MAXNDEG^- realization, and
- (ii) $\sigma_2 = (1^{2\alpha})$ or $\sigma_2 = (d^d, 1^{2\alpha+1})$ for some integers $d \geq 2, \alpha \geq 0$

is MAXNDEG^- realizable. ◀

References

- 1 Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discrete Mathematics*, 136:3–20, 1994.
- 2 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Realizability of graph specifications: Characterizations and algorithms. In *25th SIROCCO*, pages 3–13, 2018.
- 3 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Efficiently realizing interval sequences. In *30th Int. Symp. on Algorithms and Computation, ISAAC*, pages 47:1–47:15, 2019.
- 4 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph profile realizations and applications to social networks. In *13th WALCOM*, pages 1–12, 2019.
- 5 Michael D. Barrus and Elizabeth A. Donovan. Neighborhood degree lists of graphs. *Discrete Mathematics*, 341(1):175–183, 2018.
- 6 Kevin E Bassler, Charo I Del Genio, Péter L Erdős, István Miklós, and Zoltán Toroczkai. Exact sampling of graphs with prescribed degree correlations. *New Journal of Physics*, 17(8):083052, August 2015.
- 7 Joseph K. Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6(4):489–522, 2011.
- 8 Sheshayya A. Choudum. A simple proof of the Erdős-Gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, 33(1):67–70, 1991.
- 9 Brian Cloteaux. Fast sequential creation of random realizations of degree sequences. *Internet Mathematics*, 12(3):205–219, 2016.
- 10 Paul Erdős and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.
- 11 C. Del Genio, H. Kim, Z. Toroczkai, and K.E. Bassler. Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *PLOS ONE*, 5:1–8, 2010.
- 12 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.

- 13 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.
- 14 Ravi Kannan, Prasad Tetali, and Santosh S. Vempala. Simple markov-chain algorithms for generating bipartite graphs and tournaments. *Random Struct. & Algorithms*, 14(4):293–308, 1999.
- 15 P.J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7:961–968, 1957.
- 16 H. Kim, Z. Toroczkai, P.L. Erdős, I. Miklós, and L.A. Székely. Degree-based graph construction. *J. Phys. A: Math. & Theor.*, 42:392001, 2009.
- 17 Milena Mihail and Nisheeth Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. *3rd Workshop on Approximation and Randomization Algorithms in Communication Networks*, 2002.
- 18 Elchanan Mossel and Nathan Ross. Shotgun assembly of labeled graphs. *CoRR*, abs/1504.07682, 2015.
- 19 Peter V. O’Neil. Ulam’s conjecture and graph reconstructions. *Amer. Math. Monthly*, 77:35–43, 1970.
- 20 Gerard Sierksma and Han Hoogeveen. Seven criteria for integer sequences being graphic. *J. Graph Theory*, 15(2):223–231, 1991.
- 21 Amitabha Tripathi and Himanshu Tyagi. A simple criterion on degree sequences of graphs. *Discrete Applied Mathematics*, 156(18):3513–3517, 2008.
- 22 S.M. Ulam. *A Collection of Mathematical Problems*. Wiley, 1960.
- 23 D.L. Wang and D.J. Kleitman. On the existence of n -connected graphs with prescribed degrees ($n > 2$). *Networks*, 3:225–239, 1973.
- 24 N.C. Wormald. Models of random regular graphs. *Surveys in Combinatorics*, 267:239–298, 1999.

A Dynamic Space-Efficient Filter with Constant Time Operations

Ioana O. Bercea

Tel Aviv University, Israel
ioana@cs.umd.edu

Guy Even

Tel Aviv University, Israel
guy@eng.tau.ac.il

Abstract

A dynamic dictionary is a data structure that maintains sets of cardinality at most n from a given universe and supports insertions, deletions, and membership queries. A filter approximates membership queries with a one-sided error that occurs with probability at most ε . The goal is to obtain dynamic filters that are space-efficient (the space is $1 + o(1)$ times the information-theoretic lower bound) and support all operations in constant time with high probability. One approach to designing filters is to reduce to the retrieval problem. When the size of the universe is polynomial in n , this approach yields a space-efficient dynamic filter as long as the error parameter ε satisfies $\log(1/\varepsilon) = \omega(\log \log n)$. For the case that $\log(1/\varepsilon) = O(\log \log n)$, we present the first space-efficient dynamic filter with constant time operations in the worst case (whp). In contrast, the space-efficient dynamic filter of Pagh et al. [29] supports insertions and deletions in amortized expected constant time. Our approach employs the classic reduction of Carter et al. [9] on a new type of dictionary construction that supports random multisets.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Data Structures

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.11

Related Version A full version of the paper is available at <https://arxiv.org/abs/1911.05060>.

Funding This research was supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the United States National Science Foundation (NSF).

Acknowledgements We would like to thank Michael Bender, Martin Farach-Colton, and Rob Johnson for introducing us to this topic and for interesting conversations. Many thanks to Tomer Even for helpful and thoughtful remarks.

1 Introduction

We consider the problem of maintaining datasets subject to insert, delete, and membership query operations. Given a set \mathcal{D} of n elements from a universe \hat{U} , a membership query asks if the queried element $x \in \hat{U}$ belongs to the set \mathcal{D} . When exact answers are required, the associated data structure is called a *dictionary*. When one-sided errors are allowed, the associated data structure is called a *filter*. Formally, given an error parameter $\varepsilon > 0$, a filter always answers “yes” when $x \in \mathcal{D}$, and when $x \notin \mathcal{D}$, it makes a mistake with probability at most ε . We refer to such an error as a *false positive* event¹.

¹ The probability is taken over the random choices that the filter makes.



When false positives can be tolerated, the main advantage of using a filter instead of a dictionary is that the filter requires much less space than a dictionary [9, 25]. Let $\hat{u} \triangleq |\hat{\mathcal{U}}|$ be the size of the universe and n denote an upper bound on the size of the set at all points in time. The information theoretic lower bound for the space of dictionaries is $\lceil \log_2 \binom{\hat{u}}{n} \rceil = n \log(\hat{u}/n) + \Theta(n)$ bits.²³ On the other hand, the lower bound for the space of filters is $n \log(1/\varepsilon)$ bits [9]. In light of these lower bounds, we call a dictionary *space-efficient* when it requires $(1 + o(1)) \cdot n \log(\hat{u}/n) + \Theta(n)$ bits, where the term $o(1)$ converges to zero as n tends to infinity. Similarly, a space-efficient filter requires $(1 + o(1)) \cdot n \log(1/\varepsilon) + O(n)$ bits.⁴

When the set \mathcal{D} is fixed, we say that the data structure is *static*. When the data structure also supports insertions, we say that it is *incremental*. Data structures that handle both deletions and insertions are called *dynamic*.

The goal is to design dynamic dictionaries and filters that achieve “the best of both worlds” [2]: they are space-efficient and perform operations in constant time in the worst case with high probability.⁵

The Dynamic Setting. One approach for designing dynamic filters was suggested by Pagh et al. [29], outlined as follows. Static (resp., incremental) filters can be obtained from static (resp., incremental) dictionaries for sets by a reduction of Carter et al. [9]. This reduction simply hashes the universe to a set of cardinality n/ε . Due to collisions, this reduction does not directly lead to dynamic filters. Indeed, if two elements x and y in the dataset collide, and x is deleted, how is y kept in the filter? To overcome the problem with deletions, an extension of the reduction to the dynamic setting was proposed by Pagh et al. [29]. This proposal is based on employing a dictionary that maintains multisets rather than sets (i.e., elements in multisets have arbitrary multiplicities). This extension combined with a dynamic dictionary for multisets yields a dynamic filter [29]. In fact, Pagh et al. obtain a dynamic filter that is space-efficient but performs insertions and deletions in amortized constant time (but not in the worst case). Until recently, the design of a dynamic dictionary on multisets that is space-efficient and performs operations in constant time in the worst case whp was open [2]. In this paper, we avoid the need for supporting arbitrary multisets by observing that it suffices to support random multisets (see Sec. 3).⁶

Another approach for designing filters employs retrieval data structures. In the retrieval problem, we are given a function $f : \mathcal{D} \rightarrow \{0, 1\}^k$, where $f(x)$ is called the *satellite data* associated with $x \in \mathcal{D}$. When an element $x \in \hat{\mathcal{U}}$ is queried, the output y must satisfy $y = f(x)$ if $x \in \mathcal{D}$ (if $x \notin \mathcal{D}$, any output is allowed). By storing as satellite data a random fingerprint of length $\log(1/\varepsilon)$, a retrieval data structure can be employed as a filter at no additional cost in space and with an $O(1)$ increase in time per operation [14, 35] (the increase in time is for computing the fingerprint). This reduction was employed in the static case and it also holds in the dynamic case (see Sec. 6).

² All logarithms are base 2 unless otherwise stated. In x is used to denote the natural logarithm.

³ This equality holds when \hat{u} is significantly larger than n .

⁴ An asymptotic expression that mixes big-O and small-o calls for elaboration. If $\varepsilon = o(1)$, then the asymptotic expression does not require the $O(n)$ addend. If ε is constant, the $O(n)$ addend only emphasizes the fact that the constant that multiplies n is, in fact, the sum of two constants: one is almost $\log(1/\varepsilon)$, and the other does not depend on ε . Indeed, the lower bound in [25] excludes space $(1 + o(1)) \cdot n \log(1/\varepsilon)$ in the dynamic setting for constant values of ε .

⁵ By with high probability (whp), we mean with probability at least $1 - 1/n^{\Omega(1)}$. The constant in the exponent can be controlled by the designer and only affects the $o(1)$ term in the space of the dictionary or the filter.

⁶ We recently resolved the problem of supporting arbitrary multisets in [6] (thus the dictionary in [6] can support arbitrary multisets vs. the dictionary presented here that only supports random multisets).

Using the dynamic retrieval data structure of Demaine et al. [12], one can obtain a filter that requires $(1 + o(1)) \cdot n \log(1/\varepsilon) + \Theta(n \log \log(\hat{u}/n))$ bits and performs operations in constant time in the worst case whp. When the size of the universe satisfies $\hat{u} = \text{poly}(n)$, this reduction yields a space-efficient filter when the false positive probability ε satisfies $\log(1/\varepsilon) = \omega(\log \log n)$ (which we call the *sparse case*).⁷ This approach is inherently limited to the sparse case since dynamic retrieval data structures have a space lower bound of $\Theta(n \log \log(\hat{u}/n))$ regardless of the time each operation takes and even when storing two bits of satellite data [28].

Thus, the only case in which a space-efficient dynamic filter with constant time operations is not known is when $\log(1/\varepsilon) = O(\log \log n)$. We refer to this case as the *dense case*. The dense case occurs, for example, in applications in which n is large and ε is a constant (say $\varepsilon = 1\%$).

1.1 Our Contributions

In this paper, we present the first dynamic space-efficient filter for the dense case with constant time operations in the worst case whp. In the following theorem, we assume that the size of the universe \hat{U} is polynomial in n .⁸ We allow ε to be as small as $n/|\hat{U}|$ (below this threshold, simply use a dictionary). Memory accesses are in the RAM model in which every memory access reads/writes a word of $\Theta(\log n)$ contiguous bits. All computations we perform over one word take constant time (see Sec. 4.1). *Overflow* refers to the event that the space allocated for the filter does not suffice.

► **Theorem 1.** *There exists a dynamic filter that maintains a set of at most n elements from a universe $\hat{U} = [\hat{u}]$, where $\hat{u} = \text{poly}(n)$ with the following guarantees: (1) For every polynomial in n sequence of insert, delete, and query operations, the filter does not overflow whp. (2) If the filter does not overflow, then every operation (query, insert, and delete) can be completed in constant time. (3) The required space is $(1 + o(1)) \cdot n \log(1/\varepsilon) + O(n)$ bits. (4) For every query, the probability of a false positive event is bounded by ε .*

Our result is based on the observation that it suffices to use the reduction of Carter et al. [9] on dictionaries that support *random* multisets rather than arbitrary multisets. A random multiset is a uniform random sample (with replacements) of the universe. In Sec. 3, we prove that the reduction of Carter et al. [9] can be applied in this new setting. We then design a dynamic space-efficient dictionary that works on random multisets from a universe $\mathcal{U} = [u]$ with $\log(u/n) = O(\log \log n)$ (Sec. 4). The dictionary supports operations in constant time in the worst case whp. Applying the reduction of Carter et al. [9] to this new dictionary yields our dynamic filter in the dense case. Together with the filter construction for the sparse case (included, for completeness, in Sec. 6), we obtain Theorem 1.

1.2 Our Model

Memory Access Model. We assume that the data structures are implemented in the RAM model in which the basic unit of one memory access is a word. Let w denote the memory word length in bits. We assume that $w = \Theta(\log n)$. See Sec. 4.1 for a discussion of how computations over words are implemented in constant time.

⁷ The terms “sparse” and “dense” stem from the fact that the reduction of Carter et al. [9] is employed. Thus, the filter is implemented by a dictionary that stores n elements from a universe of cardinality n/ε .

⁸ This is justified by mapping \hat{U} to $[\text{poly}(n)]$ using 2-independent hash functions [12].

Success Probability. We prove that overflow occurs with probability at most $1/\text{poly}(n)$ and that one can control the degree of the polynomial (the degree of the polynomial only affects the $o(1)$ term in the size bound). In the case of random multisets, the probability of an overflow is a joint probability distribution over the random choices of the dictionary and the distribution over the realizations of the multiset. In the case of sets, the probability of an overflow depends only on the random choices that the filter makes.

Hash Functions. The filter for the dense case employs pairwise independent hash functions and invertible permutations of the universe that can be evaluated in constant time and that have a small space representation (i.e., the one-round Feistel permutations of Arbitman et al. [2] or the quotient hash functions of Demaine et al. [12]). For simplicity, we first analyze the filter construction assuming fully random hash functions (Sec. 4.5). In Sec. 5, we prove that the same arguments hold when we use succinct hash functions.

Worst Case vs. Amortized. An interesting application that emphasizes the importance of worst-case performance is that of handling search engine queries. Such queries are sent in parallel to multiple servers, whose responses are then accumulated to generate the final output. The latency of this final output is determined by the slowest response, thus reducing the average latency of the final response to the worst latency among the servers. See [1, 2, 7, 23] for further discussion on the shortcomings of expected or amortized performance in practical scenarios.

The Extendable Setting. This paper deals with the non-extendable setting in which the bound n on the cardinality of the dataset is known in advance. The filter is allocated space that is efficient with respect to the lower bound on the space of a filter with parameters u, n, ε . The extendable scenario in which space must adapt to the current cardinality of the dataset is addressed in Pagh et al. [31]. In fact, they prove that extendable filters require an extra $\Omega(\log \log n)$ bits per element.

1.3 Related Work

The topic of dictionary and filter design is a fundamental theme in the theory and practice of data structures. We restrict our focus to the results that are closest to our setting (i.e., are space-efficient, take constant time per operation, support dynamic sets).

Dictionaries. The dictionary of Arbitman et al. [2] is the only space-efficient dynamic dictionary for sets that performs all operations in constant time in the worst case with high probability. They leave it as an open question whether one can design a dictionary on multisets with similar guarantees. Indeed, their construction does not seem to extend even to the case of random multisets. The main reason is that the second level of their dictionary (the backyard), implemented as a de-amortized cuckoo hash table, does not support duplicate elements. Moreover, the upper bound on the number of elements that the backyard stores is $\Omega\left(\frac{\log \log n}{(\log n)^{1/3}} \cdot n\right)$. As such, it cannot accommodate storing naive fixed-length counters of elements (which would require $\Theta(\log n)$ bits per element) without rendering the dictionary space-inefficient.

The space-efficient dynamic dictionary for multisets of Pagh et al. [29] supports queries in constant time, and insertions/deletions in amortized expected constant time. For dictionaries on sets, several dynamic constructions support operations in constant time with high

probability but are not space-efficient [1, 11–13]. On the other hand, some dictionaries are space-efficient but do not have constant time guarantees with high probability for all of their operations [16, 21, 33, 36]. For the static case, several space-efficient constructions exist that perform queries in constant time [8, 30, 34, 39].

Filters. The filters of Pagh et al. [29] and Arbitman et al. [2] follow from their respective dictionaries by employing the reduction of Carter et al. [9]. Specifically, the dynamic filter of Pagh et al. [29] supports queries in constant time and insertions and deletions in amortized expected constant time. The incremental filter of Arbitman et al. [2] performs queries in constant time and insertions in constant time with high probability. It does not support deletions.

The construction of Bender et al. [4] describes a dynamic adaptive filter that assumes access to fully random hash functions.⁹ The adaptive filter works in conjunction with an external memory dictionary (on the set of elements) and supports operations in constant time with high probability (however, an insert or query operation may require accessing the external memory dictionary). The space of the external memory dictionary is not counted in the space of their filter. The (in-memory) filter they employ is a variant of the dynamic quotient filter [5, 10, 29, 32]. The space-efficient quotient filter employs linear probing and performs operations only in expected constant time for large values of ε [29]. The filter in [4] tries to avoid a large running time per insert operation by bounding the displacement of the inserted element. Hence, if (Robin Hood) linear probing does not succeed after a constant number of words, then the element is inserted in a secondary structure (see Sec. 5.3 in [3]). There is a gap in [3] regarding the question of whether bounded displacements guarantee constant time operations in the worst case. Specifically, searching for an element requires finding the beginning of the “cluster” that contains the “run” associated with that particular element. No description or proof is provided in [3] that the beginning of the cluster is a constant number of words away from the “quotient” in the worst case.

Other filters of interest include the dynamic filter of Pagh et al. [31] that adjusts its space on the fly to the cardinality of the dataset (hence, works without knowing the size of the dataset in advance) and performs operations in constant time. Pagh et al. [31] also prove a lower bound that forces a penalty of $O(\log \log n)$ per element for such “self-adjusting” dynamic filters. Another filter is the cuckoo filter, whose performance depends on the number of elements currently stored in the filter but that has been reported to work well in practice [18, 19]. Space-efficient filters for the static case have been studied extensively [14, 16, 26, 35].

1.4 Paper Organization

Preliminaries are in Sec. 2. The proof that the reduction of Carter et al. [9] can be employed to construct dynamic filters from dynamic dictionaries on random multisets can be found in Sec. 3. The filter for the dense case is described and analyzed in Sec. 4. Section 5 includes a discussion on how to remove the assumption of access to fully random hash functions from Sec. 4.5. Section 6 reviews the construction of a filter in the sparse case based on a retrieval data structure. Theorem 1 is proved in Sec. 7.

⁹ Loosely speaking, an adaptive filter is one that fixes false positives after they occur [4, 27].

2 Preliminaries

Notation. The *indicator function* of a set S is the function $\mathbb{1}_S : S \rightarrow \{0, 1\}$ defined by

$$\mathbb{1}_S(x) \triangleq \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{if } x \notin S. \end{cases}$$

For any positive k , let $[k]$ denote the set $\{0, \dots, [k] - 1\}$. For a string $a \in \{0, 1\}^*$, let $|a|$ denote the length of a in bits.

We define the range of a hash function h to be a set of natural numbers $[k]$ and also treat the image $h(x)$ as a binary string, i.e., the binary representation of $h(x)$ using $\lceil \log_2 k \rceil$ bits.

2.1 Filter and Dictionary Definitions

Let \hat{U} denote the universe of all possible elements.

Operations. We consider three types of operations:

- $\text{insert}(x_t)$ - insert $x_t \in \hat{U}$ to the dataset.
- $\text{delete}(x_t)$ - delete $x_t \in \hat{U}$ from the dataset.
- $\text{query}(x_t)$ - is $x_t \in \hat{U}$ in the dataset?

Dynamic Sets and Random Multisets. Every sequence of operations $R = \{\text{op}_t\}_{t=1}^T$ defines a *dynamic set* $\mathcal{D}(t)$ over \hat{U} as follows.¹⁰

$$\mathcal{D}(t) \triangleq \begin{cases} \emptyset & \text{if } t = 0 \\ \mathcal{D}(t-1) \cup \{x_t\} & \text{if } \text{op}_t = \text{insert}(x_t) \\ \mathcal{D}(t-1) \setminus \{x_t\} & \text{if } \text{op}_t = \text{delete}(x_t) \\ \mathcal{D}(t-1) & \text{if } t > 0 \text{ and } \text{op}_t = \text{query}(x_t). \end{cases} \quad (1)$$

► **Definition 2.** A multiset \mathcal{M} over \hat{U} is a function $\mathcal{M} : \hat{U} \rightarrow \mathbb{N}$. We refer to $\mathcal{M}(x)$ as the multiplicity of x . If $\mathcal{M}(x) = 0$, we say that x is not in the multiset. We refer to $\sum_{x \in \hat{U}} \mathcal{M}(x)$ as the cardinality of the multiset and denote it by $|\mathcal{M}|$.

The *support* of the multiset is the set $\{x \mid \mathcal{M}(x) \neq 0\}$. The *maximum multiplicity* of a multiset is $\max_{x \in \hat{U}} \mathcal{M}(x)$.

A *dynamic multiset* $\{\mathcal{M}_t\}_t$ is specified by a sequence of insert and delete operations. Let \mathcal{M}_t denote the multiset after t operations.¹¹

$$\mathcal{M}_t(x) \triangleq \begin{cases} 0 & \text{if } t = 0 \\ \mathcal{M}_{t-1}(x) + 1 & \text{if } \text{op}_t = \text{insert}(x) \\ \mathcal{M}_{t-1}(x) - 1 & \text{if } \text{op}_t = \text{delete}(x) \\ \mathcal{M}_{t-1}(x) & \text{otherwise.} \end{cases}$$

We say that a dynamic multiset $\{\mathcal{M}_t\}_t$ has cardinality at most n if $|\mathcal{M}_t| \leq n$, for every t .

► **Definition 3.** A *dynamic multiset* \mathcal{M} over \hat{U} is a *random multiset* if for every t , the multiset \mathcal{M}_t is the outcome of independent uniform samples (with replacements) from \hat{U} .

¹⁰The definition of state in Equation 1 does not rule out a deletion of $x \notin \mathcal{D}(t-1)$. However, we assume that $\text{op}_t = \text{delete}(x_t)$ only if $x_t \in \mathcal{D}(t-1)$.

¹¹As in the case of dynamic sets, we require that $\text{op}_t = \text{delete}(x_t)$ only if $\mathcal{M}_{t-1}(x_t) > 0$.

Dynamic Filters. A *dynamic filter* is a data structure that maintains a dynamic set $\mathcal{D}(t) \subseteq \hat{\mathcal{U}}$ and is parameterized by an error parameter $\varepsilon \in (0, 1)$. Consider an input sequence that specifies a dynamic set $\mathcal{D}(t)$, for every t . The filter outputs a bit for every query operation. We denote the output that corresponds to $\text{query}(x_t)$ by $\text{out}_t \in \{0, 1\}$. We require that the output satisfy the following condition:

$$\text{op}_t = \text{query}(x_t) \Rightarrow \text{out}_t \geq \mathbb{1}_{\mathcal{D}(t)}(x_t). \quad (2)$$

The output out_t is an approximation of $\mathbb{1}_{\mathcal{D}(t)}(x_t)$ with a one-sided error. Namely, if $x_t \in \mathcal{D}(t)$, then b_t must equal 1.

► **Definition 4** (false positive event). Let FP_t denote the event that $\text{op}_t = \text{query}(x_t)$, $\text{out}_t = 1$ and $x_t \notin \mathcal{D}(t)$.

The error parameter $\varepsilon \in (0, 1)$ is used to bound the probability of a false positive error.

► **Definition 5.** We say that the false positive probability in a filter is bounded by ε if it satisfies the following property. For every sequence R of operations and every t ,

$$\Pr[FP_t] \leq \varepsilon.$$

The probability space in a filter is induced only by the random choices (i.e., choice of hash functions) that the filter makes. Note also that if $\text{op}_t = \text{op}_{t'} = \text{query}(x)$, where $x \notin \mathcal{D}(t) \cup \mathcal{D}(t')$, then the events FP_t and $FP_{t'}$ may not be independent (see [4, 27] for a discussion of repeated false positive events and adaptivity).

Dynamic Dictionaries. A *dynamic dictionary* with parameter n is a dynamic filter with parameters n and $\varepsilon = 0$. In the case of multisets, the response out_t of a dynamic dictionary to a $\text{query}(x_t)$ operation must satisfy $\text{out}_t = 1$ iff $\mathcal{M}_t(x_t) > 0$.¹²

When we say that a filter or a dictionary has parameter n , we mean that the cardinality of the input set/multiset is at most n at all points in time.

Success Probability and Probability Space. We say that a dictionary (filter) *works* for sets and random multisets if the probability that the dictionary does not overflow is high (i.e., it is $\geq 1 - 1/\text{poly}(n)$). The probability in the case of random multisets is taken over both the random choices of the dictionary and the distribution of the random multisets. In the case of sets, the success probability depends only on the random choices of the dictionary.

Dense vs. Sparse. We differentiate between two cases in the design of filters, depending on $1/\varepsilon$.

► **Definition 6.** The dense case occurs when $\log(1/\varepsilon) = O(\log \log n)$. The sparse case occurs when $\log(1/\varepsilon) = \omega(\log \log n)$.

3 Reduction: Filters Based on Dictionaries

In this section, we employ the reduction of Carter et al. [9] to construct dynamic filters out of dynamic dictionaries for random multisets. Our reduction can be seen as a relaxation of the reduction of Pagh et al. [29]. Instead of requiring that the underlying dictionary support multisets, we require that it only supports random multisets. We say that a function $h : A \rightarrow B$ is *fully random* if h is sampled u.a.r. from the set of all functions from A to B .

¹²One may also define $\text{out}_t = \mathcal{M}_t(x_t)$.

11:8 A Dynamic Space-Efficient Filter with Constant Time Operations

▷ **Claim 7.** Consider a fully random hash function $h : \hat{\mathcal{U}} \rightarrow [\frac{n}{\varepsilon}]$ and let $\mathcal{D} \subseteq \hat{\mathcal{U}}$. Then $h(\mathcal{D})$ is a random multiset of cardinality $|\mathcal{D}|$.

Consider a dynamic set $\mathcal{D}(t)$ specified by a sequence of insert and delete operations. Since h is random, an “adversary” that generates the sequence of insertions and deletions for $\mathcal{D}(t)$ becomes an oblivious adversary with respect to $h(\mathcal{D}(t))$ in the following sense. Insertion of x translates to an insertion of $h(x)$ which is a random element (note that $h(x)$ may be a duplicate of a previously inserted element¹³). When deleting at time t , the adversary specifies a previous time $t' < t$ in which an insertion took place, and requests to delete the element that was inserted at time t' .

Let **Dict** denote a dynamic dictionary for random multisets of cardinality at most n from the universe $[\frac{n}{\varepsilon}]$.

► **Lemma 8.** *For every dynamic set $\mathcal{D}(t)$ of cardinality at most n , the dictionary **Dict** with respect to the random multiset $h(\mathcal{D}(t))$ and universe $[\frac{n}{\varepsilon}]$ is a dynamic filter for $\mathcal{D}(t)$ with parameters n and ε .*

Proof Sketch. The **Dict** records the multiplicity of $h(x_t)$ in the multiset $h(\mathcal{D}(t))$ and so deletions are performed correctly. The filter outputs 1 if and only if the multiplicity of $h(x_t)$ is positive. False positive events are caused by collisions in h . Therefore, the probability of a false positive is bounded by ε because of the cardinality of the range of h . ◀

4 Fully Dynamic Filter (Dense Case)

In this section, we present a fully dynamic filter for the dense case, i.e., $\log(1/\varepsilon) = O(\log \log n)$. The reduction in Lemma 8 implies that it suffices to construct a dynamic dictionary for random multisets. We refer to this dictionary as the *RMS-Dictionary* (RMS - Random Multi-Set).

The RMS-Dictionary is a dynamic space-efficient dictionary for random multisets of cardinality at most n from a universe $\mathcal{U} = [u]$, where $u = n/\varepsilon$. The dense case implies that $\log(u/n) = O(\log \log n)$.

The RMS-Dictionary consists of two levels of dictionaries: a set of *bin dictionaries* (in which most of the elements are stored) and a *spare* (which stores $n_s = O(n/\log^3 n)$ elements). The number of bin dictionaries is m . Let $B \triangleq n/m$, so, in expectation, each bin stores (at most) B elements. To accommodate deviations from the expectation, extra capacity is allocated in the bin dictionaries. Namely, each bin dictionary can store up to $(1 + \delta) \cdot B$ elements.

The universe of the spare dictionary is $[u]$. However, the universe of each bin dictionary is $[u/m]$. The justification for the reduced universe of bin dictionaries is that the index of the bin contains $\log m$ bits of information about the elements in it (this reduction in the universe size is often called “quotienting” [5, 12, 24, 29, 30]).

4.1 The Bin Dictionary

The bin dictionary is a (deterministic) dynamic dictionary for (small) multisets. Let u' denote the cardinality of the universe from which elements stored in the bin dictionary are taken. Let n' denote an upper bound on the cardinality of the dynamic multiset stored in a

¹³ Duplicates in $h(\mathcal{D}(t))$ are caused by collisions (i.e., $h(x) = h(y)$) rather than by reinsertions.

bin dictionary (i.e., n' includes multiplicities). The bin dictionary must be space-efficient, namely, it must fit in $n' \log(u'/n') + O(n')$ bits, and must support queries, insertions, and deletions in constant time.

The specification of the bin dictionary is even more demanding than the dictionary we are trying to construct. The point is that we focus on parametrizations in which the bin dictionary fits in a constant number of words. Let $B \triangleq \Theta\left(\frac{\log n}{\log(u/n)}\right)$ and $\delta \triangleq \Theta\left(\frac{\log \log n}{\sqrt{B}}\right) = o(1)$. Recall that the number of bins is $m = n/B$.

► **Observation 9.** *Let $u' = u/m$ and $n' = (1 + \delta) \cdot B$. The bin dictionary for u' and n' fits in $O(\log n)$ bits, and hence fits in a constant number of words.*

We propose two implementations of bin dictionaries that meet the specifications; one is based on lookup tables, and the other on Elias-Fano encoding [17, 20]. The space required by the bin dictionaries that employ global lookup tables meets the information-theoretic lower bound. The space required by the Elias-Fano encoding is within half a bit per element more than the information-theoretic lower bound [17].

Global Tables. We follow Arbitman et al. [2] and employ a global lookup table common to all the bin dictionaries. For the sake of simplicity, we discuss how insertion operations are supported. An analogous construction works for queries and deletions.

The bin dictionary has $s \triangleq \binom{u'+n'}{n'}$ states. Hence, we need to build a table that encodes a function $f : s \times u' \rightarrow s$, such that given a state $i \in [s]$ and an element $x \in [u']$, $f(i, x)$ encodes the state of the bin dictionary after x is inserted. The size of the table that stores f is $s \cdot u' \cdot \log s$ bits.

We choose the following parametrization so that the table size is $o(n)$ (recall that n is the upper bound on the cardinality of the whole multiset). Set $B = \frac{1}{2(1+\delta)} \cdot \frac{\log n}{\log(1+u/n)}$ (recall that B is the expected occupancy of a bin). Recall that $u' = u/m$ and $n' = (1 + \delta) \cdot B$. Hence,

$$\begin{aligned} s &= \binom{u' + n'}{n'} \leq \left(\frac{e(u' + n')}{n'}\right)^{n'} \\ &\leq \text{poly}(\log n) \cdot \left(1 + \frac{u'}{n'}\right)^{n'} \\ &\leq \text{poly}(\log n) \cdot \left(1 + \frac{u}{n}\right)^{(1+\delta) \cdot \frac{1}{2(1+\delta)} \cdot \frac{\log n}{\log(1+u/n)}} \\ &\leq \text{poly}(\log n) \cdot \sqrt{n}. \end{aligned}$$

Since $u' = u/m \leq \text{poly}(\log n)$ and $\log s = O(\log n)$, we conclude that the space required to store f is $o(n)$ bits.

Operations are supported in constant time since the table is addressed by the encoding of the current state and operation.

Elias-Fano Encoding. In this section, we present a bin dictionary implementation that employs (a version of) the Elias-Fano encoding. We refer to this implementation as the *Pocket Dictionary*.

We view each element in the universe $[u']$ as a pair (q, r) , where $q \in [B]$ and $r \in [u'/B]$ (we refer to q as the *quotient* and to r as the *remainder*). Consider a multiset $F \triangleq \{(q_i, r_i)\}_{i=0}^{n'-1}$. The encoding of F uses two binary strings, denoted by $\text{header}(F)$ and $\text{body}(F)$, as follows. Let $n_q \triangleq |\{i \in [f] \mid q_i = q\}|$ denote the number of elements that share the same quotient q .

11:10 A Dynamic Space-Efficient Filter with Constant Time Operations

The vector (n_0, \dots, n_{B-1}) is stored in $\text{header}(F)$ in unary as the string $1^{n_0} \circ 0 \circ \dots \circ 1^{n_{B-1}} \circ 0$. The length of the header is $B + n'$. The concatenation of the remainders is stored in $\text{body}(F)$ in nondecreasing lexicographic order of $\{(q_i, r_i)\}_{i \in [n']}$. The length of the body is $n' \cdot \log(u'/B)$. The space required is $B + n'(1 + \log(u'/n))$ bits, which meets the required space bound since $B = O(n')$.

We argue that operations in a Pocket Dictionary can be executed in constant time if the Pocket Dictionary fits in a single word. Here we propose to extend the classical RAM model in which instructions such as comparison, addition, and multiplication take constant time [22].¹⁴ These instructions require Boolean circuits of depth $\log w$, where w denotes the number of bits per word. Moreover, multiplication is implemented using circuits with $\Theta(w^2)$ gates (i.e., all the partial products are computed). Hence, we consider an extension of the RAM model in which instructions over words can be executed in constant time if there exists a Boolean circuit with constant fan-in that computes the instruction in $O(\log w)$ depth using $O(w^2)$ gates.

Indeed, operations over Pocket Dictionaries can be supported by circuits of depth $\log w$ with $O(w \log w)$ gates. Consider an insertion operation of an element (q, r) . Insertion is implemented using the following steps (all implemented by circuits):

- (i) Locate in the header the positions of q th zero and the zero that proceeds it (this is a select operation). Let j and i denote these positions within the header. This implies that $\sum_{q' < q} n_{q'} = i - (q - 1)$ and $n_q = j - i - 1$.
- (ii) Update the header by shifting the suffix starting in position j by one position and inserting a 1 in position j .
- (iii) Read n_q remainders in the body, starting from position $i - (q - 1)$. These remainders are compared in parallel with r , to determine the position p within the body in which r should be inserted (this is a rank operation over the outcomes of the comparisons).
- (iv) Shift the suffix of the body starting with position p by $|r|$ bits, and copy r into the body starting at position r .

Modern instruction sets support instructions such as rank, select, and SIMD comparisons (shifts are standard instructions) [3, 32, 37]. Hence, one can implement Pocket Dictionary operations in constant time using such instruction sets.

4.2 The Spare

The spare is a dynamic dictionary that maintains (arbitrary) multisets of cardinality at most n_s from the universe \mathcal{U} with the following guarantees: (1) For every $\text{poly}(n)$ sequence of operations (insert, delete, or query), the spare does not overflow whp. (2) If the spare does not overflow, then every operation (query, insert, delete) takes $O(1)$ time. (3) The required space is $O(n_s \log u)$ bits.¹⁵

We propose to implement the spare by using a dynamic dictionary on sets with constant time operations in which counters are appended to elements. To avoid having to discuss the details of the interior modifications of the dictionary, we propose a black-box approach that employs a retrieval data structure in addition to the dictionary.

¹⁴In modern CPUs, addition and comparison take a single clock cycle, multiplication may take 2 cycles.

¹⁵Since $n_s = o(n/\log n)$ and $\log u = O(\log n)$, the space consumed by the spare is $o(n)$.

► **Observation 10.** *Any dynamic dictionary on sets of cardinality at most n_s from the universe \mathcal{U} can be used to implement a dynamic dictionary on arbitrary multisets of cardinality at most n_s from the universe \mathcal{U} . This reduction increases the space of the dictionary on sets by an additional $\Theta(\log n_s + \log \log(u/n_s))$ bits per element and increases the time per operation by a constant.*

Proof. The dictionary on multisets (MS-Dict) can be obtained by employing a dictionary on sets (Dict) and the dynamic retrieval data structure (Ret) of Demaine et al. [12]. The dictionary on sets (Dict) stores the support of the input multiset. The retrieval data structure (Ret) stores as satellite data the multiplicity of each element in the support. The space that Ret requires is $\Theta(n_s \log n_s + n_s \log \log(u/n_s))$, since the satellite data occupies $\log n_s$ bits.

On membership queries, the dictionary accesses Dict. When a new element x is inserted, MS-Dict inserts x in Dict and adds x with satellite value 1 to Ret. In the case of insertions of duplicates or deletions, Ret is updated to reflect the current multiplicity of the element. If upon deletion, the multiplicity of the element reaches 0, the element is deleted from Dict and from Ret. Since the Ret supports operations in constant time, this reduction only adds $O(1)$ time to the operations on Dict. ◀

To finish the description of the spare, we set $n_s \triangleq n/(\log^3 n)$ and employ Obs. 10 with a dynamic dictionary on sets that requires $O(n_s \log(u/n_s))$ bits and performs operations in constant time whp [1, 2, 11–13]. Under our definition of n_s and since $\log(u/n) = O(\log \log n)$, the spare then requires $o(n)$ bits. Moreover, the spare does not overflow whp (see Claim 11).

4.3 Hash Functions

We consider three hash functions, the bin index, quotient, and remainder, as follows: ¹⁶ (Recall that $n = m \cdot B$.)

$$\begin{aligned} h^b : \mathcal{U} &\rightarrow [m] && \text{(bin index)} \\ q : \mathcal{U} &\rightarrow [B] && \text{(quotient)} \\ r : \mathcal{U} &\rightarrow [u/n] && \text{(remainder)} \end{aligned}$$

We consider three settings for the hash functions:

- (i) Fully random hash functions.
- (ii) In the case that the dataset is a random multiset, the values of the hash functions are taken simply from the bits of x . Namely $h^b(x)$ is the first $\log(n/B)$ bits, $q(x)$ is the next $\log B$ bits, and $r(x)$ is the last $\log(u/n)$ bits (to be more precise, one needs to divide x and take remainders). Since x is chosen independently and uniformly at random, the hash functions are fully random.
- (iii) Hash functions sampled from special distributions of hash functions (with small representation and constant evaluation time).

4.4 Functionality

A query(x) is implemented by searching for $(q(x), r(x))$ in the bin dictionary of index $h^b(x)$. If the pair is not found, the query is forwarded to the spare. An insert(x) operation first attempts to insert $(q(x), r(x))$ in the bin dictionary of index $h^b(x)$. If the bin dictionary

¹⁶One could define the domain of the quotient function $q(x)$ and the remainder function $r(x)$ to be $[u/m]$ instead of \mathcal{U} .

is full, it forwards the insertion to the spare. A $\text{delete}(x)$ operation searches for the pair $(q(x), r(x))$ in the bin dictionary of index $h^b(x)$ and deletes it (if found). Otherwise, it forwards the deletion to the spare.

4.5 Overflow Analysis¹⁷

The proposed dictionary consists of two types of dictionaries: many small bin dictionaries and one spare. The overflow of a bin dictionary is handled by sending the element to the spare. Hence, for correctness to hold, we need to prove that the spare does not overflow whp.

The first challenge that one needs to address is that the dictionary maintains a dynamic multiset $D(t)$ (see [12]). Consider the insertion of an element x at time t . If bin $h^b(x)$ is full at time t , then x is inserted in the spare. Now suppose that an element y with $h^b(y) = h^b(x)$ is deleted at time $t + 1$. Then, bin $h^b(x)$ is no longer full, and x cannot “justify” the fact that it is in the spare at time $t + 1$ based on the present dynamic multiset $D(t + 1)$. Indeed, x is in the spare due to “historical reasons”.

The second challenge is that the events that elements are sent to the spare are not independent. Indeed, if x is sent to the spare at time t , then we know that there exists a full bin. The existence of a full bin is not obvious if the cardinality of $D(t)$ is small. Hence, we cannot even argue that the indicator variables for elements being sent to the spare are negatively associated.

The following claim bounds the number of elements stored in the spare. Using the same proof, one could show that the number of elements in the spare is bounded by $n/(\log n)^c$ whp, for every constant c .

▷ **Claim 11.** The number of elements stored in the spare is less than $n/\log^3 n$ with high probability.

Proof. Consider a dynamic multiset $D(t)$ at time t . To simplify notation, let $\{x_1, \dots, x_{n'}\}$ denote the multiset $D(t)$ (hence, the elements need not be distinct, and $n' \leq n$). Let t_i denote the time in which x_i was inserted to $D(t)$. Let X_i denote the random variable that indicates if x_i is stored in the spare (i.e., $X_i = 1$ iff bin $h^b(x_i)$ is full at time t_i). Our goal is to prove that the spare at time t does not overflow whp, namely:

$$\Pr \left[\sum_{i=1}^{n'} X_i \geq \frac{n}{\log^3 n} \right] \leq n^{-\omega(1)}. \quad (3)$$

The claim follows from Eq. 3 by applying a union bound over the whole sequence of operations.

To prove Eq. 3, we first bound the probability that a bin is full. Let $\gamma \triangleq e^{-\delta^2 \cdot B/3}$. Fix a bin b , by a Chernoff bound, the probability that bin b is full is at most γ . Indeed:

- (i) Each element belongs to bin b with probability B/n . Hence, the expected occupancy of a bin is B .
- (ii) The variables $\{h^b(x_j)\}_{j=1}^{n'}$ are independent.
- (iii) A bin is full if at least $(1 + \delta) \cdot B$ elements belong to it.

We overcome the problem that the random variables $\{X_i\}_i$ are not independent as follows. Let F_t denote the set of full bins at time t . If $|F_t| \leq 6\gamma m$, let \hat{F}_t denote an arbitrary superset of F_t that contains $6\gamma m$ bins. Note that it is unlikely that there exists a t such that $|F_t| > 6\gamma m$. Indeed, by linearity of expectation, $\mathbb{E}[|F_t|] \leq \gamma m$. By a Chernoff bound, $\Pr[F_t \leq 6\gamma m] \leq 2^{-6\gamma m}$.

¹⁷The proofs in this section assume that the hash functions are fully random. See Section 5 for a discussion of special families of hash functions.

Define \hat{X}_i to be the random variable that indicates if $h^b(x_i) \in \hat{F}_{t_i}$. Namely, $\hat{X}_i = 1$ if x_i belongs to a full bin or a bin that was added to \hat{F}_{t_i} . Thus, $\hat{X}_i \geq X_i$. The key observation is that the random variables $\{\hat{X}_i\}_{i=1}^{n'}$ are independent and identically distributed because the bin indexes $\{h^b(x_i)\}_i$ are independent and uniformly distributed.

The rest of the proof is standard. Recall that $B = O(\log n)$ and $\delta^2 = \Theta\left(\frac{(\log \log n)^2}{B}\right)$.

Let G_t denote the event that $F_t \leq 6\gamma m$. Since $\Pr[G_t] \leq 2^{-6\gamma m} \leq 2^{-\sqrt{n}}$, by a union bound $\Pr\left[\bigcup_{i=1}^{n'} G_{t_i}\right] \leq n \cdot 2^{-\sqrt{n}}$.

The expectation of \hat{X}_i is 6γ (conditioned on the event $\bigcap_{i=1}^{n'} G_{t_i}$). Since $n/\log^3 n = \omega(\gamma n)$, for a sufficiently large n , by Chernoff bound $\Pr\left[\sum_{i=1}^{n'} \hat{X}_i \geq \frac{n}{\log^3 n} \mid \bigcap_{i=1}^{n'} G_{t_i}\right] < 2^{-n/\log^3 n}$.

We conclude that

$$\Pr\left[\sum_{i=1}^{n'} \hat{X}_i \geq \frac{n}{\log^3 n}\right] \leq \Pr\left[\bigcup_{i=1}^{n'} G_{t_i}\right] + \Pr\left[\sum_{i=1}^{n'} \hat{X}_i \geq \frac{n}{\log^3 n} \mid \bigcap_{i=1}^{n'} G_{t_i}\right] \quad (4)$$

$$\leq n \cdot 2^{-\sqrt{n}} + 2^{-n/\log^3 n} = n^{-\omega(1)}, \quad (5)$$

and Eq. 3 follows. \triangleleft

5 Succinct Hash Functions

In this section we discuss how to replace the fully random hash functions from Sec. 4 with succinct hash functions (i.e., representation requires $o(n)$ bits) that have constant evaluation time in the RAM model. Specifically, we describe how to select hash functions $h^b(x), r(x), q(x)$ for the RMS-dictionary used for constructing the dynamic filter in the dense case.

The construction proceeds in two stages and uses existing succinct constructions for highly independent hash functions [15, 38]. First, we employ a permutation function π to partition the universe \hat{U} into $M = n^{9/10}$ equal parts. The permutation π can be either the one-round Feistel permutation from [2] or the quotient permutation from [12]. We think of π as a pair of functions, i.e., $\pi(x) = (h_1(x), h_2(x))$ with $h_1(x) \in [M]$. This induces a partition of the dynamic set into $M = n^{9/10}$ subsets of size at most $n^{1/10} + n^{3/40}$ whp. Each subset consists of the $h_2(x)$ values of the elements $x \in \mathcal{D}$ that share the same $h_1(x)$ value.

In the second step, we instantiate the RMS-Dictionary separately for each subset. Each dictionary instance employs the same k -wise independent hash function $f^b : [\hat{u}/M] \rightarrow [m]$ with $k = n^{1/10} + n^{3/40}$. We define $h^b(x) = f^b(h_2(x))$ to be the bin of x . From the perspective of each dictionary instantiation, $h^b(x)$ is sampled independently and uniformly at random, so throughout the sequence of $\text{poly}(n)$ operations, the spare does not overflow whp.

We now describe how the quotient $q(x)$ and the remainder $r(x)$ are chosen. We sample a 2-independent hash function $(f, g) : [\hat{u}/M] \rightarrow [B] \times [1/\varepsilon]$. Define $q(x) \triangleq f(h_2(x))$ and $r(x) \triangleq g(h_2(x))$.

\triangleright **Claim 12.** Consider a filter based on an RMS-dictionary that employs the hash functions $h^b(x), q(x), r(x)$ described in this section. Then the probability of a false positive event is bounded by 2ε .

Proof. Consider a query y that is not in the dataset $D(t)$ at time t . One cause of failure is when too many elements of $\mathcal{D}(t)$ are mapped to bin $h^b(y)$. The probability that more than $(1 + \delta)B$ elements are mapped to bin $h^b(y)$ is bounded by $2e^{-\delta^2 B/3}$ (see the proof of Claim 11). Since $\delta^2 B = (\log \log n)^2$, and since $\log(1/\varepsilon) = O(\log \log n)$, this probability is $o(\varepsilon)$.

Now assume that at most $(1 + \delta)B$ elements in $\mathcal{D}(t)$ were mapped to bin $h^b(y)$. Since h is bijective and (f, g) are selected from a family of 2-independent hash functions, the probability of a collision with an element in bin $h^b(y)$ is at most $(1 + \delta)B \cdot \frac{\varepsilon}{B} = (1 + \delta) \cdot \varepsilon$. The claim follows since $\delta = o(1)$. \triangleleft

6 Dynamic Filter via Retrieval (Sparse Case)

In this section, we present a space-efficient dynamic filter for the case that $\log(\frac{1}{\varepsilon}) = \omega(\log \log n)$ (sparse case). We let n denote an upper bound on the cardinality of a dynamic set over a $\text{poly}(n)$ sequence of insertions and deletions. We let $\hat{\mathcal{U}}$ denote a universe of cardinality \hat{u} that satisfies $\hat{u} = \text{poly}(n)$. The construction is based on a reduction from dynamic retrieval. The construction relies on the fact that in dynamic retrieval structures (e.g., [12]), the overhead per element is $O(\log \log n)$. This overhead is $o(\log(1/\varepsilon))$ in the sparse case.

Dietzfelbinger and Pagh [14] formulate a reduction that uses a static retrieval data structure storing k bits of satellite data per element to implement a static filter with false positive probability 2^{-k} . The reduction is based on the assumption that retrieval data structure is “well behaved” with respect to negative queries. Namely, a query for $x \in \hat{\mathcal{U}} \setminus \mathcal{D}$ returns either “fail” or the satellite data of an (arbitrary) element $y \in \mathcal{D}$. The reduction incurs no additional cost in space and adds $O(1)$ extra time to the query operations (to evaluate the fingerprint). We note that the same reduction can be employed in the dynamic case. Specifically, the following holds:

► **Observation 13.** *Assume access to a family of pairwise independent hash functions $h : \hat{\mathcal{U}} \rightarrow [k]$.¹⁸ Then any dynamic retrieval data structure that stores $h(x)$ as satellite data for element x can be used to implement a dynamic filter with false positive probability 2^{-k} . The space of the resulting filter is the same as the space of the retrieval data structure and the time per operation increases by a constant (due to the computation of $h(x)$).*

The question of designing a space-efficient dynamic filter now boils down to:

- (i) Choose a suitable dynamic retrieval data structure.
- (ii) Determine the range of false positive probabilities ε for which the reduction yields a space-efficient dynamic filter.

We resolve this question by employing the retrieval data structure of Demaine et al. [12] in the sparse case.

▷ **Claim 14.** There exists a dynamic filter in the sparse case that maintains a set of at most n elements from the universe $\hat{\mathcal{U}} = [\hat{u}]$, where $\hat{u} = \text{poly}(n)$ such that, for any ε such that $\log(1/\varepsilon) = \omega(\log \log n)$, the following hold: (1) For every sequence of $\text{poly}(n)$ operations (i.e., insert, delete, or query), the filter does not overflow whp. (2) If the filter does not overflow, then every operation (query, insert, and delete) takes $O(1)$ time. (3) The required space is $(1 + o(1)) \cdot n \log(1/\varepsilon)$ bits. (4) For every query, the probability of a false positive event is bounded by ε .

Proof. The dynamic retrieval data structure in [12] uses a dynamic perfect hashing data structure for n elements from the universe \mathcal{U} of size u that maps each element to a unique value in a given range $[n + t]$, for any $t > 0$.¹⁹ The space that the perfect hashing data structure

¹⁸We note that Dietzfelbinger and Pagh [14] assume access to fully random hash functions. Pairwise independence suffices, however, as noted by [35].

¹⁹We note that one could also use the dynamic perfect hashing scheme proposed in Mortesen et al. [28] with similar space and performance guarantees.

occupies is $\Theta\left(n \log \log \frac{\hat{u}}{n} + n \log \frac{n}{t+1}\right)$. All operations are performed in $O(1)$ time and the perfect hashing data structure fails with $1/\text{poly}(n)$ probability over a sequence of $\text{poly}(n)$ operations. A retrieval data structure can be obtained by allocating an array of $n+t$ entries, each used to store satellite data of k bits. The satellite data associated with an element x is stored at the position in the array that corresponds to the hash code associated with x . The retrieval data structure obtained this way occupies $(n+t) \cdot k + \Theta\left(n \log \log \frac{\hat{u}}{n} + n \log \frac{n}{t+1}\right)$ bits. It performs every operation in constant time and fails with probability $1/\text{poly}(n)$ over a sequence of $\text{poly}(n)$ operations.

For the filter construction, we set $k \triangleq \log(1/\varepsilon)$ and $t \triangleq n/\log n$. Since $\log \log(\hat{u}/n) = O(\log \log n)$ and $\log(1/\varepsilon) = \omega(\log \log n)$, the filter we obtain occupies $(1+o(1)) \cdot n \log(1/\varepsilon)$ bits. It performs all operations in constant time and does not overflow whp over a sequence of $\text{poly}(n)$ operations. \triangleleft

7 Proof of Theorem 1

The proof of Thm. 1 deals with the sparse case and dense case separately. The theorem for the sparse case, in which $\log(1/\varepsilon) = \omega(\log \log n)$, is proven in Sec. 6.

The proof for the dense case employs the reduction in Lemma 8 with the RMS-Dictionary construction described in Sec. 4. Let $\mathcal{U} = [u]$ where $u = n/\varepsilon$ denotes the universe of an RMS-Dictionary that can store a random multiset of cardinality at most n . In this case, the assumption that $\log(1/\varepsilon) = O(\log \log n)$ translates into $\log(u/n) = O(\log \log n)$.

The time per operation is constant because the RMS-dictionary supports operations in constant time. The space consumed by the RMS-Dictionary equals the sum of spaces for the bin dictionaries and the spare. This amounts to $m \cdot n' \cdot \log(u'/n') + m \cdot O(n') + o(n) \leq (1+\delta) \cdot n \cdot \log(1/\varepsilon) + O(n)$. Since $\delta = o(1)$, the filter is space-efficient, as required. Finally, by Claim 11, the spare does not overflow whp.

References

- 1 Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *International Colloquium on Automata, Languages, and Programming*, pages 107–118. Springer, 2009.
- 2 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.
- 3 Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. *CoRR*, abs/1711.01616, 2017. [arXiv:1711.01616](https://arxiv.org/abs/1711.01616).
- 4 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.
- 5 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, 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. *PVLDB*, 5(11):1627–1637, 2012. doi:10.14778/2350229.2350275.
- 6 Ioana O. Bercea and Guy Even. A dynamic dictionary for multisets. work in progress, 2020.
- 7 Andrei Broder and Michael Mitzenmacher. Using multiple hash functions to improve ip lookups. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications*.

- Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1454–1463. IEEE, 2001.
- 8 Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.
 - 9 Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 59–65. ACM, 1978.
 - 10 J.G. Cleary. Compact hash tables using bidirectional linear probing. *IEEE Transactions on Computers*, (9):828–834, 1984.
 - 11 Ketan Dalal, Luc Devroye, Ebrahim Malalla, and Erin McLeish. Two-way chaining with reassignment. *SIAM Journal on Computing*, 35(2):327–340, 2005.
 - 12 Erik D Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus. In *Latin American Symposium on Theoretical Informatics*, pages 349–361. Springer, 2006.
 - 13 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *International Colloquium on Automata, Languages, and Programming*, pages 6–19. Springer, 1990.
 - 14 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.
 - 15 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *International Colloquium on Automata, Languages, and Programming*, pages 354–365. Springer, 2009.
 - 16 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.
 - 17 Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM (JACM)*, 21(2):246–260, 1974.
 - 18 David Eppstein. Cuckoo filter: Simplification and analysis. In *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*, pages 8:1–8:12, 2016. doi:10.4230/LIPIcs.SWAT.2016.8.
 - 19 Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than Bloom. In *CoNEXT*, pages 75–88. ACM, 2014.
 - 20 Robert Mario Fano. On the number of bits required to implement an associative memory. memorandum 61. *Computer Structures Group, Project MAC, MIT, Cambridge, Mass.*, 1971.
 - 21 Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.
 - 22 Torben Hagerup. Sorting and searching on the word ram. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 366–398. Springer, 1998.
 - 23 Adam Kirsch and Michael Mitzenmacher. Using a queue to de-amortize cuckoo hashing in hardware. In *Proceedings of the Forty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, volume 75, 2007.
 - 24 Donald E Knuth. The art of computer programming, vol. 3: Searching and sorting. *Reading MA: Addison-Wisley*, 1973.
 - 25 Shachar Lovett and Ely Porat. A lower bound for dynamic approximate membership data structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 797–804. IEEE, 2010.
 - 26 Michael Mitzenmacher. Compressed Bloom filters. *IEEE/ACM Transactions on Networking (TON)*, 10(5):604–612, 2002.
 - 27 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.

- 28 Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu. On dynamic range reporting in one dimension. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 104–111. ACM, 2005.
- 29 Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *SODA*, pages 823–829. SIAM, 2005.
- 30 Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2001.
- 31 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.
- 32 Prashant Pandey, Michael A. Bender, Rob Johnson, and Robert Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 775–787, 2017. doi:10.1145/3035918.3035963.
- 33 Rina Panigrahy. Efficient hashing with lookups in two memory accesses. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–839. Society for Industrial and Applied Mathematics, 2005.
- 34 Mihai Pătraşcu. Succincter. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 305–313. IEEE, 2008.
- 35 Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *International Computer Science Symposium in Russia*, pages 263–273. Springer, 2009.
- 36 Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *International Colloquium on Automata, Languages, and Programming*, pages 357–368. Springer, 2003.
- 37 James Reinders. AVX-512 Instructions. <https://software.intel.com/en-us/articles/intel-avx-512-instructions>, 2013.
- 38 Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004.
- 39 Huacheng Yu. Nearly optimal static las vegas succinct dictionary. *arXiv preprint arXiv:1911.01348*, 2019.

A Simple Algorithm for Minimum Cuts in Near-Linear Time

Nalin Bhardwaj

University of California San Diego, CA, USA
nalinbhardwaj@nibnalin.me

Antonio J. Molina Lovett

Department of Computer Science, Princeton University, NJ, USA
antonio@amolina.ca

Bryce Sandlund

Cheriton School of Computer Science, University of Waterloo, Canada
bcsandlund@gmail.com

Abstract

We consider the minimum cut problem in undirected, weighted graphs. We give a simple algorithm to find a minimum cut that 2-respects (cuts two edges of) a spanning tree T of a graph G . This procedure can be used in place of the complicated subroutine given in Karger’s near-linear time minimum cut algorithm [23]. We give a self-contained version of Karger’s algorithm with the new procedure, which is easy to state and relatively simple to implement. It produces a minimum cut on an m -edge, n -vertex graph in $O(m \log^3 n)$ time with high probability, matching the complexity of Karger’s approach.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases minimum cut, sparsification, near-linear time, packing

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.12

Supplementary Material Our implementation is available at: <https://github.com/nalinbhardwaj/min-cut-paper>.

1 Introduction

The minimum cut problem on an undirected (weighted) graph G asks for a vertex subset S such that the total number (weight) of edges from S to $V \setminus S$ is minimized. The minimum cut problem is a fundamental problem in graph optimization and has received vast attention by the research community across a number of different computation models [23, 8, 36, 20, 25, 15, 41, 19, 5, 17, 4, 22, 11, 27, 18, 6, 10, 34, 39, 12]. Its applications include network reliability [37, 21], cluster analysis [3], and a critical subroutine in cutting-plane algorithms for the traveling salesman problem [2].

A seminal result in weighted minimum cut algorithms is an algorithm by Karger [23] which produces a minimum cut on an m -edge, n -vertex graph in $O(m \log^3 n)$ time with high probability¹. This algorithm stood as the fastest minimum cut algorithm for the past two decades, until very recently, work published on arXiv shaved a log factor in Karger’s approach [31, 9]. The main component of Karger’s algorithm is a subroutine that finds a minimum cut that 2-respects (cuts two edges of) a given spanning tree T of a graph G . In other words, the cut found is minimal amongst all cuts of G that cut exactly two edges of T . Despite the number of pairs of spanning tree edges totaling $\Omega(n^2)$, Karger shows

¹ Probability $1 - 1/n^c$ for some constant c .



this can be accomplished in $O(m \log^2 n)$ time. Unfortunately, the procedure developed is particularly complex, a detail Karger admits when comparing the algorithm to a simpler $O(n^2 \log n)$ algorithm he develops to find *all* minimum cuts [23]. Indeed, perhaps for this reason, implementation of the asymptotically fastest minimum cut algorithm has been avoided in practical performance analyses [5, 19].

In this paper, we give a simple algorithm to find a minimum cut that 2-respects a spanning tree T of a graph G . Our procedure runs in $O(m \log^2 n)$ time, matching the performance of Karger’s more-complicated subroutine. We achieve the simplification via a clever use of the heavy-light decomposition. Although our procedure requires the top tree data structure [1] to achieve optimal performance, at the cost of an extra $O(\log n)$ factor, heavy-light decomposition can be used a second time so that only augmented binary search trees are required. We also give a self-contained version of Karger’s algorithm [23] with this new procedure and implement it, avoiding issues associated with previous implementations [23, 5].

Karger’s algorithm [23], as well as the edge-sampling technique it is based on [22], has been extended and adapted to achieve results in a number of different settings [12, 6, 41, 11, 10, 34]. In particular, in the fully-dynamic setting, Thorup [41] uses the tree-packing technique developed by Karger [23], but maintains a larger set of trees so that the minimum cut 1-respects at least one of them. In the parallel setting, Geissmann and Gianinazzi [11] are able to parallelize both the dynamic tree data structure and the necessary computation required by Karger’s algorithm [23]. This work is based off prior work in the cache-oblivious model [10], also based on Karger’s algorithm [23]. In the distributed setting, Ghaffari and Kuhn [12] achieve a $(2 + \epsilon)$ -approximation to the minimum cut based on Karger’s sampling technique [22]. This is improved to a $(1 + \epsilon)$ -approximation with similar runtime by Nanongkai and Su [34]. Nanongkai and Su develop their algorithm from Thorup’s fully-dynamic min-cut algorithm [41], Karger’s sampling technique [22], and Karger’s dynamic program to find the minimum cut that 1-respects a tree [23]. Finally, Daga et al. [6] achieve a sublinear time distributed algorithm to compute the exact minimum cut in an unweighted undirected graph. This algorithm builds off a more recent development in minimum cut algorithms [27], combined again with the tree-packing technique introduced by Karger [23]. Specifically, a tree packing is found in an efficient number of distributed rounds, then Karger’s more-complicated algorithm to find a minimum 2-respecting cut is applied in the distributed setting.

This vast amount of work based on Karger’s original near-linear time algorithm suggests that simplifying it may yield additional techniques that can be applied both sequentially and in alternative settings. Indeed, the very recent improvements to Karger’s algorithm [31, 9] were published on arXiv two months after our paper was first made available online [28], one of which [9] cites our paper as what drew the authors to the problem. Indeed, their procedure for “descendent edges”, given in Section 3.1, is similar to our procedure given in Section 5. We have further found use of the approach given in this paper to achieve new results in dynamic higher connectivity algorithms [30].

This paper is organized as follows. In Section 2, we state the history of the minimum cut problem, in particular discussing other simple algorithms. In Section 3, we give an overview of Karger’s algorithm to pack spanning trees, leaving the details of the approach to Appendix A. Our main contribution is given in Sections 4 and 5. In Section 4, we show how to find minimum cuts that 1-respect (cut one edge of) a tree using our new procedure. In Section 5, we extend the approach to find minimum cuts that 2-respect (cut two edges of) a tree. We discuss our implementation in Section 6 and give concluding remarks in Section 7.

2 Related Work

Before we begin, we give a brief history of the minimum cut problem. The minimum cut problem was originally perceived as a harder variant of the maximum s - t flow problem and was solved by $\binom{n}{2}$ flow computations. Gomory and Hu [14] showed how to compute all pairwise max flows in $n - 1$ flow computations, thus reducing the complexity of the minimum cut problem by a $\Theta(n)$ factor. Hao and Orlin [16] further showed that the minimum cut in a directed graph can be reduced to a single flow computation.

Nagamochi and Ibaraki [33, 32] developed a deterministic algorithm that is not based on computing maximum s - t flows. They achieve $O(nm + n^2 \log n)$ time on a capacitated, undirected graph. This procedure was simplified by Stoer and Wagner [39], achieving the same runtime. The Stoer-Wagner algorithm gives a simple procedure to find an *arbitrary* minimum s - t cut. Vertices s and t are then merged, and the procedure repeats. Although the $O(nm + n^2 \log n)$ time complexity requires an efficient priority queue such as a Fibonacci heap [7], a binary heap can be used to achieve runtime $O(nm \log n)$.

Two algorithms based on *edge contraction* have been devised. The first is an algorithm of Karger [20] and is incredibly simple. The algorithm randomly contracts edges until only two vertices remain. Repeated $O(n^2 \log n)$ times, the algorithm finds all minimum cuts on an undirected, weighted graph in $O(n^2 m \log n)$ time with high probability. This technique was improved by Karger and Stein [25] by observing an edge of the minimum cut is more likely to be contracted later in the contraction procedure. Their improvement branches the contraction procedure after a certain threshold has been reached, spending more time to avoid contracting an edge of the minimum cut when fewer edges remain. The Karger-Stein algorithm achieves runtime $O(n^2 \log^3 n)$, finding the minimum cut with high probability.

In an unweighted graph, Gabow [8] showed how to compute the minimum cut in $O(cm \log(n^2/m))$ time, where c is the capacity of the minimum cut. Karger [22] improved Gabow's algorithm by applying random sampling, achieving runtime $\tilde{O}(m\sqrt{c})$ in expectation². The sampling technique developed by Karger [22], combined with the tree-packing technique devised by Gabow [8], form the basis of Karger's near-linear time minimum cut algorithm [23]. As previously mentioned, this technique finds the minimum cut in an undirected, weighted graph in $O(m \log^3 n)$ time with high probability.

A recent development uses low-conductance cuts to find the minimum cut in an undirected unweighted graph. This technique was introduced by Kawarabayashi and Thorup [27], who achieve near-linear deterministic time (estimated to be $O(m \log^{12} n)$). This was improved by Henzinger, Rao, and Wang [18], who achieve deterministic runtime $O(m \log^2 n (\log \log n)^2)$. Although the algorithm of Henzinger et al. is more efficient than Karger's algorithm [23] on unweighted graphs, the procedure, as well as the one it was based on [27], are quite involved, thus making them largely impractical for implementation purposes.

Since an earlier version of this paper became available online [28], several important improvements in minimum cut algorithms have been discovered. Ghaffari et al. [13] devise a randomized unweighted minimum cut algorithm by using contraction based on sampling from each vertex, rather than standard uniform edge sampling. Their algorithm reduces unweighted minimum cuts to weighted minimum cuts on a graph with $O(n)$ edges, achieving $O(\min(m + n \log^3 n, m \log n))$ time complexity. Gawrychowski et al. [9] improve Karger's procedure for finding the minimum cut that 2-respects a tree to $O(m \log n)$ time. This improves the state-of-the-art for weighted minimum cuts to $O(m \log^2 n)$ time and, by Ghaffari et al. [13],

² The $\tilde{O}(f)$ notation hides $O(\log f)$ factors.

improves the complexity of unweighted minimum cuts to $O(\min(m + n \log^2 n, m \log n))$ time. Mukhopadhyay and Nanongkai [31] also study Karger’s procedure for finding the minimum cut that 2-respects a tree, arriving at an $O(m \frac{\log^2 n}{\log \log n} + n \log^6 n)$ time weighted minimum cut algorithm. Mukhopadhyay and Nanongkai further apply their new procedure to minimum cuts in the cut-query and streaming models.

3 Overview of Karger’s Spanning Tree Packing

We first formalize the definition mentioned earlier in this paper and originally given by Karger.

► **Definition 1** (Karger [23]). *Let T be a spanning tree of G . We say that a cut in G k -respects T if it cuts at most k edges of T . We also say that T k -constrains the cut in G .*

We also define weighted tree packings.

► **Definition 2** (Karger [23]). *A weighted tree packing is a set of spanning trees, each with an assigned non-negative weight, such that the total weight of trees containing a given edge of G is no greater than the weight of that edge. The weight of the packing is the total weight of the trees in it.*

The first stage of Karger’s algorithm is to sample edges independently and uniformly at random from graph G to form a graph H , and then pack spanning trees in H . If we sample a tree T from a packing with probability proportional to its weight, a minimum cut in G will cut at most two edges of T with constant probability. Thus, if we sample $O(\log n)$ trees from the weighted packing, a minimum cut in G 2-respects at least one of the sampled trees with high probability. The remainder of the algorithm is a procedure that, given a spanning tree T of a graph G , finds a minimal cut of G that 2-respects T . This procedure is applied to all $O(\log n)$ sampled spanning trees.

We leave the intuition behind Karger’s approach and the relevant mathematics to Appendix A. We will use Algorithm 1 to pack spanning trees, credited to Thorup and Karger [42], Plotkin-Shmoys-Tardos [36], and Young [43]. The procedure appears in Gawrychowski et al. [9].

■ **Algorithm 1** Obtain a Packing of Weight at least $.4c$ from a Graph G .

Let G be a graph with m edges and n vertices.

1. Initialize $\ell(e) \leftarrow 0$ for all edges e of G . Initialize multiset $P \leftarrow \emptyset$. Initialize $W \leftarrow 0$.
 2. Repeat the following:
 - a. Find a minimum spanning tree T with respect to $\ell(\cdot)$.
 - b. Set $\ell(e) \leftarrow \ell(e) + 1/(75 \ln m)$ for all $e \in T$. If $\ell(e) > 1$, return W, P .
 - c. Set $W \leftarrow W + 1/(75 \ln m)$.
 - d. Add T to P .
-

► **Lemma 3** ([36, 42, 43]). *Given an undirected unweighted graph G with m edges, n vertices, and minimum cut c , Algorithm 1 returns a weighted packing of weight at least $.4c$ in $O(mc \log n)$ time.*

Algorithm 1 and Lemma 3 are given in Appendix A with general epsilon and proven. To achieve $O(mc \log n)$ time in Algorithm 1, we may use a linear time minimum spanning tree routine [24] or the following implementation trick given by Gawrychowski et al. [9]. In the use of Algorithm 1 in Algorithm 2, the graph in Algorithm 1 has edges which may be

duplicated $O(\log n)$ times, while the number of distinct edges can be bounded as a factor $\Theta(\log n)$ fewer. It suffices to invoke the minimum spanning tree algorithm of Algorithm 1 with only the minimum of each set of parallel edges. We can easily maintain the minimum of each set of parallel edges in $O(\log n)$ time per edge per iteration, which suffices to shave a log factor in the runtime of Algorithm 1. Note that if we chose to avoid these optimizations and/or avoid the use of top trees in Section 5, the final runtime becomes $O(m \log^4 n)$.

We use Algorithm 1 in Algorithm 2 to obtain $\Theta(\log n)$ trees for the 2-respect algorithm given in Sections 4 and 5.

■ **Algorithm 2** Obtain $\Theta(\log n)$ Spanning Trees for the 2-respect Algorithm.

Let d denote the exponent in the probability of success $1 - 1/n^d$. Let $b = 3 \cdot 6^2(d + 2) \ln n$.

1. Form graph G' from G by first normalizing the edge weights of G so the smallest non-zero edge weight has weight 1, then multiplying each edge weight by 100 and rounding to the nearest integer. Let U be an upper bound for the size of the minimum cut of G' .
 2. Initialize $c' \leftarrow U$. Repeat the following:
 - a. Construct H in the following way: for each edge e of G' , let e have weight in H drawn from the binomial distribution with probability $p = \min(b/c', 1)$ and number of trials the weight of e in G' . Cap the weight of any edge in H to at most $\lceil 7/6 \cdot 12b \rceil$.
 - b. Run Algorithm 1 on H , considering an edge of weight w as w parallel edges. There are three cases:
 - i. If $p = 1$, set P to the packing returned and skip to step 3.
 - ii. If the returned packing is of weight $24b/70$ or greater, set $c' \leftarrow c'/6$ and repeat steps 2a and 2b, setting P to the packing returned and then proceeding to step 3.
 - iii. Otherwise, repeat steps 2a and 2b with $c' \leftarrow c'/2$.
 3. Return $\lceil 36.53d \ln n \rceil$ trees sampled uniformly at random proportional to their weights from P .
-

► **Lemma 4.** *Algorithm 2 returns a collection of $\Theta(\log n)$ spanning trees of G in time $O(m \log^3 n)$ such that the minimum cut of G 2-respects at least one tree in the collection with high probability.*

Algorithm 2 and Lemma 4 are given in Appendix A with general epsilon and proven.

4 Minimum Cuts that 1-Respect a Tree

We now give our algorithm for finding a minimum cut that 1-respects a spanning tree T of a graph G . We present it here only to build intuition for the idea used to find 2-respecting cuts in the following section, which also finds 1-respecting cuts.

We use the following lemma, a consequence of Sleator and Tarjan's heavy-light decomposition [38].

► **Lemma 5** (Sleator and Tarjan [38]). *Given a tree T , there is an ordering of the edges of T such that the edges of the path between any two vertices in T consist of the union of up to $2 \log n$ contiguous subsequences of the order. The order can be found in $O(n)$ time.*

Proof. We use heavy-light decomposition, credited to Sleator and Tarjan [38]. Note that the algorithm assumes T is rooted. We can root T arbitrarily. We then take the heavy paths given from the usual construction and concatenate them in any order. ◀

12:6 Simple Min-Cut in Near-Linear Time

Our algorithm begins by labeling the edges of T in heavy-light decomposition order e_1, \dots, e_{n-1} as given by Lemma 5. Consider the cut of G induced by the vertex partition resulting from cutting a single edge of T . We iterate index i through heavy-light decomposition order and keep up-to-date the total weight of all edges of G that cross the cut induced by e_i . The minimum weight found is then returned.

Call the edges of G in T *tree edges* and edges of G not in T *non-tree edges*. Critical to our approach is the following proposition.

► **Proposition 6.** *For any cut of G that 2-respects T , the non-tree edge uv crosses the cut if and only if exactly one tree edge from the uv -path in T crosses the cut.*

Proof. Recall that for any edge of T crossing the cut, the components of each of its endpoints must fall on opposite sides of the cut. Therefore if the number of tree edges in the cut on the uv -path in T is odd, the non-tree edge uv crosses the cut. Since we are only considering cuts that cut at most 2 edges of T , the proposition follows. ◀

We now give our algorithm explicitly.

■ **Algorithm 3** Minimum Cuts that 1-Respect T .

1. Arrange the edges of T in the order of Lemma 5; label them e_1, \dots, e_{n-1} .
 2. For each non-tree edge uv , mark every i such that e_i is on the uv -path in T and e_{i+1} is not on the uv -path in T , or vice versa. Indicate whether edge e_1 is on the uv -path in T .
 3. Iterate index i from 1 to $n - 1$, in each iteration keeping track of the total weight of all non-tree edges uv such that e_i lies on the uv -path in T , added together with the weight of edge e_i .
 4. Return the minimum total weight found in step 3.
-

► **Lemma 7.** *Algorithm 3 finds the value of the minimum cut that 1-respects a spanning tree T of a graph G in $O(m \log n)$ time.*

Proof. Via Proposition 6, in a 1-respecting cut including only e_i from T , a non-tree edge uv is cut if and only if the edge e_i lies on the uv -path in T . Algorithm 3 keeps track of all such non-tree edges for each possible e_i that is cut, therefore it finds the minimum cut of G that cuts a single edge of T .

The time complexity can be determined as follows. Finding the heavy-light decomposition for step 1 takes $O(n)$ time. In doing so, we can label each edge and each heavy path so that every edge knows its index in the order as well as the heavy path to which it belongs. Each heavy path can store its starting and ending index in the order. With this information, step 2 can be completed by walking up from u and v in T towards the root of T . We spend $O(1)$ work per heavy path from root to vertex, which is bounded by $O(\log n)$ via the heavy-light decomposition. In total this step takes $O(m \log n)$ time.

In step 3, we spend $O(n)$ total work plus $O(1)$ work for each transition of the current edge e_i on or off the uv path for all non-tree edges uv . Each non-tree edge transitions on or off $O(\log n)$ times as guaranteed by Lemma 5, therefore the time complexity of this step is $O(m \log n)$. Overall, Algorithm 3 takes $O(m \log n)$ time. ◀

Note that if we wish to find the edges in the minimum cut, we can keep track of the minimum-achieving index i so we know the vertex separation of the minimum cut. With the vertex separation, it is easy to find in $O(m \log n)$ time which non-tree edges cross the cut.

Further note that we need not know the identity of the non-tree edge uv as e_i falls on or off the uv -path. Thus the space required for step 2 need only be $O(m)$, since at each transition point we can just keep track of the total weight added or subtracted from the minimum cut.

5 Minimum Cuts that 2-Respect a Tree

We now discuss an extension of Algorithm 3 to find a minimum cut that 2-respects a tree. We still iterate i through heavy-light decomposition order, but in addition to cutting e_i , we find the best j so that the cut resulting from cutting e_i and e_j is minimal. To find the best j efficiently we use a clever data structure.

► **Lemma 8** (Alstrup et al. [1]). *There is a data structure that supports the following operations on a weighted tree T in $O(\log n)$ time:*

- *PathAdd(u, v, x) := Add weight x to all edges on the unique uv -path in T .*
- *NonPathAdd(u, v, x) := Add weight x to all edges not on the unique uv -path in T .*
- *QueryMinimum() := Query for the minimum weight edge in T .*

Proof. Operations PathAdd() and QueryMinimum() are just Theorems 3 and 4 of [1]. Operation NonPathAdd(u, v, x) can be achieved by keeping a counter of global weight added to (subtracted from) T and executing PathAdd($u, v, -x$) to undo this action on the uv -path. See also [40]. ◀

Note that the weight x can be positive or negative.

If we seek to avoid implementing any sophisticated data structures, we can instead use heavy-light decomposition again and support the above two operations in $O(\log^2 n)$ time. To see how, by Lemma 5 each path of T represents at most $O(\log n)$ contiguous segments of the total order of edges. Range add and a global minimum query can be supported in $O(\log n)$ time via an augmented binary search tree. Thus the total time complexity per operation is $O(\log^2 n)$.

We use the range operations as follows. As we iterate index i through the order of Lemma 5, we keep up to date the cost of the cut resulting from cutting any other edge e_j via the data structure of Lemma 8. Instead of querying each other edge e_j directly, however, we just use a global minimum query to find the best choice of j . The procedure is given in Algorithm 4. The first two steps are the same as Algorithm 3.

► **Lemma 9.** *Algorithm 4 finds the value of the minimum cut that 2-respects a spanning tree T of a graph G in $O(m \log^2 n)$ time.*

Proof. By Proposition 6, in a 2-respecting cut including e_i and e_j of T , a non-tree edge uv is cut if and only if exactly one of e_i or e_j lies on the uv -path in T . Observe that the invariants enforced in step 4 guarantee that in each iteration the total weight of edges from the cut resulting from cutting any other edge e_j along with e_i is kept up-to-date in the data structure of Lemma 8. Since the minimum such j is found for every i , it follows that step 4 finds the weight of the minimum cut of G that cuts exactly two edges of T . In step 5, we return the minimum of this weight with a single call to QueryMinimum() where we assume edge e_i to be off the path of all non-tree edges uv . Observe that this computes the minimum cut of G that cuts exactly one edge of T . Thus, the minimum cut of G that 2-respects T is returned in step 5.

The time complexity follows similarly to Algorithm 3. Steps 1 and 2 take $O(m \log n)$ total time. However, step 4 requires $O(\log n)$ time for non-tree edge uv whenever edge e_i

Algorithm 4 Minimum Cuts that 2-Respect T .

1. Arrange the edges of T in the order of Lemma 5; label them e_1, \dots, e_{n-1} .
 2. For each non-tree edge uv , mark every i such that e_i is on the uv -path in T and e_{i+1} is not on the uv -path in T , or vice versa. Indicate whether edge e_1 is on the uv -path in T .
 3. Initialize the data structure of Lemma 8 on T so that the weight of edge e_j is equal to its weight in T .
 4. Iterate index i from 1 to $n - 1$. Via the computation done in step 2, maintain the following invariants in the data structure of Lemma 8 as i is iterated.
 - a. When edge e_i is on the uv -path in T , add the weight of non-tree edge uv to all edges off the uv -path in T .
 - b. When edge e_i is off the uv -path in T , add the weight of non-tree edge uv to all edges on the uv -path in T .

Each time i is incremented, after updating weights in Lemma 8 as per 4a and 4b, add ∞ to edge e_i , execute `QueryMinimum()`, then subtract ∞ from edge e_i . The value of the minimum cut found in each iteration is the result of `QueryMinimum()` plus the weight of e_i .
 5. Return the minimum of the smallest cut found in step 4 with the result of `QueryMinimum()` when we consider edge e_i to be off the path of all non-tree edges uv in the data structure of Lemma 8.
-

falls on or off the uv -path in T , since the data structure of Lemma 8 takes $O(\log n)$ time per operation. For a given non-tree edge uv , edge e_i falls on or off the uv -path in T a total of $O(\log n)$ times by Lemma 5; thus step 4 takes $O(m \log^2 n)$ time. The final `QueryMinimum()` call in step 5 takes $O(m \log n)$ time. The total time taken is $O(m \log^2 n)$. ◀

We make a few further remarks about Algorithm 4. To determine the edges of the minimum cut, the data structure of Lemma 8 can be augmented to return the index j of the edge that achieves the minimum given in operation `QueryMinimum()`. With e_i and e_j , we can determine the vertex partition in G of the minimum cut and, as stated in Section 4, and from this we can find which non-tree edges cross the minimum cut easily in $O(m \log n)$ time.

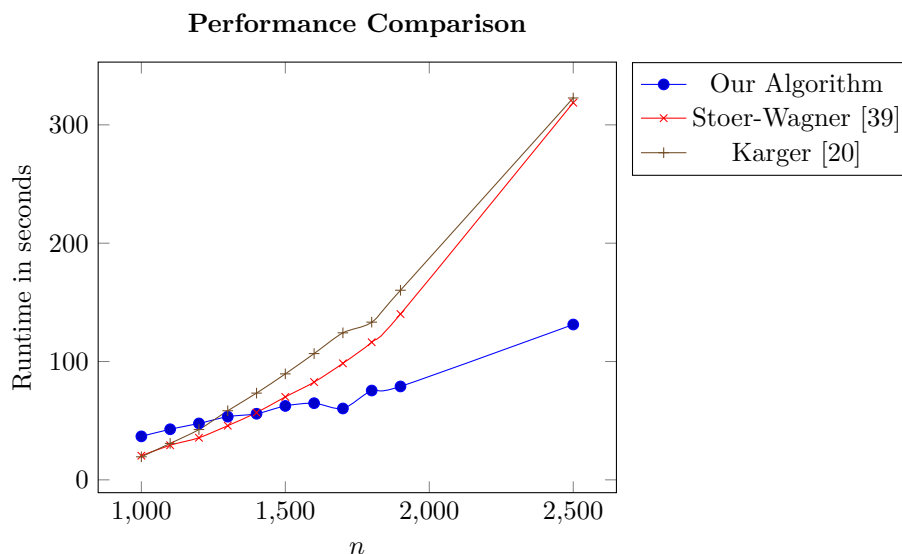
The space complexity of Algorithm 3 was easily linear. In Algorithm 4, we must know the identity of each non-tree edge uv in every transition point where edge e_i falls on or off the uv -path. Naively this costs $O(m \log n)$ space. This can be improved to $O(m)$ space by performing step 2 incrementally while executing step 4. That is, we only need to know the next transition point where the non-tree edge uv falls on or off the uv -path, and from the current transition point this can be determined in constant time via the heavy-light decomposition.

Recall that while Algorithm 3 helped demonstrate the approach of Algorithm 4, we need only implement Algorithm 4, since Algorithm 4 finds the minimum cut of G that cuts either 1 or 2 edges of T .

From this we get our final theorem, equivalent to the result of Karger [23].

► **Theorem 10.** *The minimum cut in a weighted undirected graph can be found in $O(m \log^3 n)$ time with high probability.*

Proof. We first find $\Theta(\log n)$ spanning trees by Algorithm 2. We then find the minimum cuts that 2-respect each of these trees by Algorithm 4. By Lemmas 4 and 9, this finds the minimum cut with high probability in $O(m \log^3 n)$ time. ◀



■ **Figure 1** Performance comparison of an $O(m \log^4 n)$ implementation of our algorithm with an $O(n^3)$ Stoer-Wagner [39] and $O(n^3 \log n)$ Karger [20].

6 Implementation

We have implemented an $O(m \log^4 n)$ version of our algorithm in C++³. Algorithm 1 together with an $O(m \log n)$ minimum spanning tree routine take about 100 lines of code, Algorithm 2 takes about 200 lines, Algorithm 4 takes about 200 lines, and using an augmented binary search tree as the data structure for Lemma 8 takes about 200 lines. To the best of our knowledge, our implementation is the first to achieve near-linear time complexity. We have tested it against an $O(n^3)$ implementation of the Stoer-Wagner algorithm [39] and an $O(n^3 \log n)$ implementation of Karger’s randomized contraction algorithm [20]. Under favorable inputs, the runtime compares as in Figure 1.

Figure 1 demonstrates the near-linear growth in the running time of our algorithm. Unfortunately, it does not appear our implementation is competitive compared to existing implementations [5]. The bottleneck is in obtaining the $O(\log n)$ spanning trees for Algorithm 4, even when Algorithm 2 runs in $O(m \log^3 n)$ time and Algorithm 4 runs in $O(m \log^4 n)$ time. The issue is the large constant factors due to the quadratic dependencies on epsilons, seen in Algorithms 5 and 6. We have calculated that the number of calls to the minimum spanning tree routine in our implementation can be as much as $8100 \ln n \ln m$, and that changing the choices of epsilons for Algorithms 1 and 2 does not yield significant improvement.

If we replace Algorithm 1 with the more-complicated Gabow’s algorithm [8], we can likely improve our implementation’s runtime. Further, a factor of about two can be saved by finding c' via an approximation algorithm [26]. However, a large constant factor will remain due to the sampling procedure in Lemma 14, discussed in Appendix A. All known algorithms to compute weighted tree packings have dependence on c , the value of the minimum cut, and Lemma 14 reduces the value of the minimum cut to at least $3(d+2)(\ln n)/\epsilon^2$, which in our algorithms manifests as a factor of $108(d+2) \ln n$. It appears that for Karger’s approach to be made practical, this large constant factor will likely need to be improved or heuristic approaches would need to be considered [5].

³ Our implementation is available at: <https://github.com/nalinbhardwaj/min-cut-paper>.

7 Conclusion

In this paper, we have discussed a simplification to Karger's original near-linear time minimum cut algorithm [23]. In contrast to Karger's original algorithm [23], finding spanning trees that have a constant probability of 2-respecting the minimum cut is now the more-complicated part of the algorithm and finding minimum cuts that 2-respect a tree is relatively simpler. In actuality, both were complicated in Karger's original algorithm, however the work to find the tree packing was largely abstracted to previous publications. The same can be said for many statements of Karger's near-linear time algorithm [9, 31]. Our version, on the other hand, is self-contained: the only procedures outside of Algorithms 1, 2, and 4 required to implement the full algorithm are a minimum spanning tree subroutine and (optionally) a top tree data structure.

The main contribution of our algorithm is a new, simple procedure to find a minimum cut that 2-respects a tree T in $O(m \log^2 n)$ time. Karger advertises that the complexity of his near-linear time algorithm is $O(m \log^3 n)$ and thus his routine to find a minimum cut that 2-respects a tree also takes $O(m \log^2 n)$ time. However, he gives two small improvements to the algorithm to reduce the overall runtime to $O(m \log^2 n \log(n^2/m) / \log \log n + n \log^6 n)$. The first uses the fact that finding a 1-respecting cut can be done in linear time, and the other is an improvement which reduces an $O(\log n)$ factor to an $O(\log(n^2/m))$ factor in the 2-respect routine. For our algorithm, the first improvement can be applied by substituting our 1-respect algorithm with his. The second improvement can not be applied. Thus, when $m = \Theta(n^2)$, his algorithm is faster by an $O(\log n)$ factor. However, for this case, Karger gives a different, simpler algorithm [23] which finds the global minimum cut in $O(n^2 \log n)$ time anyway.

There are three algorithms that are referred to as simple min-cut algorithms: the Stoer-Wagner algorithm [39] which runs in $O(nm \log n)$ time or $O(nm + n^2 \log n)$ time with a Fibonacci heap [7], Karger's randomized contraction algorithm [20] which runs in $O(n^2 m \log n)$ time, and the improvement to Karger's algorithm by Karger and Stein [25] which runs in $O(n^2 \log^3 n)$ time. In comparison to these, our approach is the least simple. However, our $O(m \log^3 n)$ runtime is significantly better. While the large constant factors in our approach make this only relevant at large values of n , we hope the procedure developed in this paper can be used in conjunction with an optimized version of Karger's sampling technique to produce an asymptotically fast, practical minimum cut algorithm.

References

- 1 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005.
- 2 David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.
- 3 Rodrigo A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 116–125. ACM, 1993.
- 4 Parinya Chalermsook, Jittat Fakcharoenphol, and Danupon Nanongkai. A deterministic near-linear time algorithm for finding minimum cuts in planar graphs. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 828–829, 2004.
- 5 Chandra S. Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Cliff Stein. Experimental study of minimum cut algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 324–333, 1997.

- 6 Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 343–354, 2019.
- 7 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
- 8 H.N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.
- 9 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in $O(m \log^2 n)$ time. *CoRR*, abs/1911.01145, 2019.
- 10 B. Geissmann and L. Gianinazzi. Cache oblivious minimum cut. In *International Conference on Algorithms and Complexity*, 2017.
- 11 Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures*, pages 1–11, 2018.
- 12 Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *International Symposium on Distributed Computing*, pages 1–15, 2013.
- 13 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, 2020.
- 14 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
- 15 Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in polylogarithmic amortized update. *ACM Transactions on Algorithms*, 14(2), 2018.
- 16 J.X. Hao and J.B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *Journal of Algorithms*, 17(3):424–446, 1994.
- 17 Monika Henzinger, Alexander Noe, Christian Schulz, and Darren Strash. Practical minimum cut algorithms. *J. Exp. Algorithmics*, 23:1.8:1–1.8:22, 2018.
- 18 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1919–1938, 2017.
- 19 M. Jünger, G. Rinaldi, and S. Thienel. Practical performance of efficient minimum cut algorithms. *Algorithmica*, 26:172, 2000.
- 20 David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30, 1993.
- 21 David R. Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, pages 11–17, 1995.
- 22 David R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2), 1999.
- 23 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000.
- 24 David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.
- 25 David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.
- 26 David Ron Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Stanford, CA, USA, 1995. UMI Order No. GAX95-16851.
- 27 Ken-Ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66:4:1–4:50, 2018.
- 28 Antonio Molina Lovett and Bryce Sandlund. A simple algorithm for minimum cuts in near-linear time. *CoRR*, abs/1908.11829, 2019.

- 29 David W. Matula. A linear time $2 + \epsilon$ approximation algorithm for edge connectivity. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 500–504, 1993.
- 30 Antonio J. Molina Lovett and Bryce Sandlund. personal communication.
- 31 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (to appear)*, 2020.
- 32 Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.
- 33 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k -connected spanning subgraph of a k -connected graph. *Algorithmica*, 7:583–596, 1992.
- 34 Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *International Symposium on Distributed Computing*, 2014.
- 35 C. St.J. A. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961.
- 36 S. A. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 495–504, 1991.
- 37 Aparna Ramanathan and Charles J. Colbourn. Counting almost minimum cutsets with reliability applications. *Mathematical Programming*, 39:253–261, 1987.
- 38 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.
- 39 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.
- 40 Robert E. Tarjan and Renato F. Werneck. Self-adjusting top trees. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 813–822, 2005.
- 41 Mikkel Thorup. Fully-dynamic min-cut*. *Combinatorica*, 27(1):91–127, 2007.
- 42 Mikkel Thorup and David R. Karger. Dynamic graph algorithms with applications. In *Proceedings of the 7th annual scandinavian symposium and workshops on algorithm theory*, 2000.
- 43 Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms.*, 1995.

A Karger’s Algorithm for Packing Spanning Trees

In this section we give the intuition and mathematics behind the spanning tree packing of Karger’s algorithm.

A.1 Tree Packing

The basic idea of Karger’s near-linear time algorithm [23] is to exploit the following combinatorial result. Recall that a tree packing of an undirected unweighted graph G is a set of spanning trees such that each edge of G is contained in at most one spanning tree. The weight of a tree packing is the number of trees in it.

► **Theorem 11** (Nash-Williams [35]). *Any undirected unweighted multigraph with minimum cut c contains a tree packing of weight at least $c/2$.*

Now consider a minimum cut and a tree packing given by Theorem 11. Each edge of the minimum cut can only be present in at most one spanning tree. As there are c edges of the minimum cut, this implies that the average spanning tree contains at most $c/(c/2) = 2$ edges of the minimum cut. In other words, a spanning tree chosen at random from a packing of Theorem 11 will 2-constrain the minimum cut with probability at least $1/2$.

Suppose we are given a spanning tree T of G with each edge of T marked if it crosses the minimum cut. The endpoints of any marked edge must fall on opposite sides of the cut. Conversely, the endpoints of any unmarked edge must be on the same side of the cut. It follows that if we know the edges of T that cross the minimum cut, we can determine the vertex partition of the minimum cut and its total weight in G .

This gives the intuition behind Karger's algorithm [23]. We sample spanning trees from a tree packing of G and for each tree T , we find the minimum cut that 2-respects T . Unfortunately, several obstacles need be overcome before this can be made into an efficient algorithm. For one, all currently known approaches of determining a tree packing of Theorem 11 have runtime $\Omega(cm)$, which for large values of c is far more than the runtime we seek. Further, Theorem 11 must be generalized to weighted graphs.

We first address the latter concern. Recall the definition of weighted tree packings given in Section 3.

► **Lemma 12** (Karger [23]). *Any undirected weighted graph with minimum cut c contains a weighted tree packing of weight at least $c/2$.*

Proof. For contradiction, suppose some graph G with minimum cut c and $\epsilon > 0$ exist such that G does not contain a weighted packing of weight $(1 - \epsilon)c/2$ or greater.

Take G and approximate each edge e_i of weight w_i by a rational number a_i/b_i such that $a_i/b_i < w_i$ and $w_i - a_i/b_i < \epsilon$. Multiply all edges by $d = \prod_i b_i$ and call the resulting graph G' . Then by Theorem 11, when viewed as an unweighted multigraph, G' has a tree packing of weight at least $(1 - \epsilon)dc/2$. If we weight each tree of the packing by $1/d$, the packing becomes a weighted packing of G of weight at least $(1 - \epsilon)c/2$, a contradiction. ◀

Note that for both Lemma 11 and Lemma 12, an upper bound of weight c also exists, because every spanning tree in the packing must cross the minimum cut at least once.

To effectively use Lemma 12, we formally state the relationship between weighted packings and trees that 2-constrain small cuts.

► **Lemma 13** (Karger [23]). *Consider a weighted graph G and a weighted tree packing of weight βc , where c is the weight of the minimum cut in G . Then given a cut of weight αc , a fraction at least $\frac{1}{2}(3 - \alpha/\beta)$ of the trees (by weight) 2-constrain the cut.*

Proof. Note that every spanning tree must cross every cut. Let x denote the total weight of trees with at least three edges crossing the cut and y the total weight of trees with one or two edges crossing the cut. Then $x + y = \beta c$ and $3x + y \leq \alpha c$. Rearranging, we get $y \geq \frac{1}{2}(3\beta c - \alpha c)$. ◀

A.2 Random Sampling

In order to avoid the $\Omega(cm)$ complexity of finding a packing of weight $c/2$, we first apply random sampling to G . Specifically, we use the following from Karger's earlier work.

► **Lemma 14** (Karger [22]). *Let $p = 3(d + 2)(\ln n)/(\epsilon^2 \gamma c) \leq 1$, where c is the weight of the minimum cut of an unweighted multigraph G and $\gamma \leq 1, \gamma = \Theta(1)$. Then if we sample each edge of G independently with probability p , the resulting graph H has the following properties with probability $1 - 1/n^d$.*

1. *The minimum cut in H is of size within a $(1 + \epsilon)$ factor of $cp = 3(d + 2)(\ln n)/(\gamma \epsilon^2)$, which is $O(\epsilon^{-2} \log n)$.*

12:14 Simple Min-Cut in Near-Linear Time

2. A cut in G takes value within a factor $(1 + \epsilon)$ of its expected value in H . In particular, the minimum cut in G corresponds (under the same vertex partition) to a $(1 + \epsilon)$ -times minimum cut of H .

By picking ϵ to be a constant such as $1/6$, Lemma 14 will allow us to reduce the size of the minimum cut in H to $O(\log n)$. We can then run existing algorithms [36, 8] to pack trees in H in $\tilde{O}(m)$ time. Further, since the minimum cut of G corresponds to a $(1 + \epsilon)$ -times minimum cut of H , we can still apply Lemma 13 on the sampled graph H so that a tree randomly sampled from the packing has a constant probability of 2-constraining the minimum cut in G .

There are still several issues to resolve. Lemma 14 applies to unweighted multigraphs G , but our graph G can have non-negative real weights. The other issue is that the value γ needs to be known ahead of time in order to apply the lemma. We first address the latter issue.

Lemma 14 requires knowing a constant-factor underestimate $c' = \gamma c$ for the minimum cut c . In particular, without $\gamma \leq 1$, property 2 of Lemma 14 is not guaranteed with high probability, and if $\gamma = o(1)$, the minimum cut of H will be of size $\omega(\epsilon^{-2} \log n)$ with high probability. We may run a linear-time 3-approximation algorithm [29], with modifications to work on weighted graphs [26], to find this approximation. This is simple to state, but more difficult to implement.

A different approach is to start with a known upper bound U for c' . Karger states that we can then halve this upper bound until “our algorithms succeed” [22]. This approach is taken by the implementation of Chekuri et al. [5]. Unfortunately, it is not rigorous as stated. Lemma 14 indicates that with a constant-factor underestimate $c' = \gamma c$ for c , our algorithm can proceed. However, it does not give a process for rejecting a guess c' that is not a constant-factor underestimate for c . We could try all powers of 2 for c' within a known lower and upper bound of the value of the minimum cut, and run our algorithms for all possibilities. This is rigorous, but introduces an extra $O(\log n)$ factor in our runtimes, assuming the range of c' we try is polynomial in n . We instead show the following.

► **Lemma 15.** *Let $p = 3(d + 2)(\ln n)/(\epsilon^2 \gamma c) \leq 1$ as in Lemma 14, but with $\gamma \geq 6$ and $\epsilon \leq 1/3$. Then if we sample each edge of the unweighted multigraph G uniformly at random with probability p , the resulting graph H has minimum cut of size less than $(d + 2)(\ln n)/\epsilon^2$ with probability at least $1 - 1/n^{d+2}$.*

Proof. Consider the size of a minimum cut of G as a cut in H . Let X be a random variable denoting this size. Then $\mathbb{E}[X] = cp$. By a Chernoff bound, $\Pr[X \geq (1 + \delta)cp] \leq e^{-\frac{1}{3}(cp\delta)}$ for $\delta \geq 1$. Let $(1 + \delta) = \frac{7}{3}$. Then

$$\begin{aligned} \Pr[X \geq (d + 2)(\ln n)/\epsilon^2] &\leq e^{-\frac{1}{3}(cp(\frac{7}{3}-1))} \\ &= e^{-(d+2)(\ln n)\gamma^{-1}\epsilon^{-2}(\frac{7}{3}-1)} \\ &= n^{-\frac{1}{3}(d+2)\epsilon^{-2}+(d+2)\gamma^{-1}\epsilon^{-2}} \\ &\leq n^{-\frac{1}{6}(d+2)\epsilon^{-2}} \\ &< n^{-(d+2)}. \end{aligned}$$

Therefore, the minimum cut in H has size less than $(d + 2)(\ln n)/\epsilon^2$ with probability at least $1 - 1/n^{d+2}$. ◀

Lemma 15 states that if our estimate $c' = \gamma c$ satisfies $\gamma \geq 6$, the minimum cut will be at least a factor 3 smaller than $3(d + 2)(\ln n)/\epsilon^2$ with high probability. Recall that with $\gamma = 1$ and therefore $c' = c$, we expect the minimum cut in H to be within a factor $(1 + \epsilon)$ from

$3(d+2)(\ln n)/\epsilon^2$ with high probability. Lemma 15 gives us the necessary tool to reject c' that are not a constant factor underestimate of c . We try a value for c' , and if the size of the minimum cut in H is greater than $(1+\epsilon)^{-1}3(d+2)(\ln n)/\epsilon^2$, we know $c' < 6c$. Therefore we can decrease c' by a factor of 6 and rerun the tree packing algorithm. The resulting graph H must satisfy the conditions of Lemma 14, therefore the algorithm may proceed. Since our tree packing algorithms determine the minimum cut up to constant factors, this approach avoids the need of a different (or recursive!) minimum cut algorithm to run on H .

We briefly remark on the choice of known upper bound U . If the edge weights are polynomially bounded by the number of vertices, n , a simple upper bound of the sum of weights of edges attached to any single vertex will do. If we do not consider this guarantee, Karger shows [22] that the minimum weight edge w in a maximum spanning tree has the property that the minimum cut must have weight between w and n^2w . Thus, setting $U = n^2w$ gives only $O(\log n)$ values of c' to try regardless of edge weights. The choice of an upper bound U is further discussed in [5].

We now return to the issue of real-value weights in Lemma 14. This was described as a complication in [5], to which they substituted a heuristic method in order to achieve practicality. The approach we have described thus far is amenable to small constant-factor approximations. Suppose we replace G with a graph G' such that each edge weight is first normalized so the smallest weight edge has weight 1, then all edge weights are multiplied by 100 and rounded to the nearest integer. Normalizing has no effect on the relative sizes of cuts in G' . Rounding to the nearest integer when the smallest weight edge has weight at least 100 has the effect that a cut of weight x will take on a new weight in range $[.995x, 1.005x]$. Then the original minimum cut of G corresponds to an at most 201/199-times minimum cut of G' . Now, G' can be represented as an unweighted multigraph and then sampled according to Lemma 14. In the resulting graph H , the minimum cut of G corresponds to an at most $201/199 \cdot 7/6$ -times minimum cut of H with the choice $\epsilon = 1/6$. By adjusting constants throughout the rest of our approach, this shows we can treat real weighted graphs G correctly. The other issue is how to do so efficiently.

If we consider G' as an unweighted multigraph, the number of edges of G' is proportional to the weight of edges of G , which may be quite large. However, we may also consider G' as an integer-weighted graph, in which case we can sample each edge of G' by drawing from the binomial distribution with probability p and number of trials the weight of the respective edge. There are many methods to sample from the binomial distribution. One simple method that can be made efficient for our purposes is inverse transform sampling. Let X denote a random variable sampled from the binomial distribution as described. In inverse transform sampling, we draw a number u uniformly at random between 0 and 1, and then choose our sample x to be the largest such that $P(X < x) \leq u$. Instead of having to sample a number of times equal to the weight of an edge, we must only compute the probabilities of the cumulative distribution function for the binomial distribution for all possible values that may result in H . We can make this efficient with the following observation. Say the weight of the minimum cut in H is \hat{c} . Then a tree packing of H has value at most \hat{c} , and in particular for a given edge, any weight beyond \hat{c} is excess capacity that cannot be used in the tree packing. It follows that capping the weight of any edge of H to the maximum size of the minimum cut in H , thus $O(\log n)$, will have no impact on the packing found. Thus, we must only compute $O(\log n)$ probabilities of the binomial distribution per edge, which can be done in total $O(\log n)$ time per edge.

The final choice is to pick a tree packing algorithm. Karger gives two options. The first is an algorithm by Gabow [8], which computes a $c/2$ packing. The second is a more general approach by Plotkin-Shmoys-Tardos [36], which can find a packing a factor $(1 + \epsilon')$ from

12:16 Simple Min-Cut in Near-Linear Time

the maximum packing, which has value in $[c/2, c]$. Karger describes the latter approach as simpler, using only minimum spanning tree computations. Although the paper [36] does not explicitly give a routine for packing spanning trees, such a procedure is explicitly given in Thorup and Karger [42], with credit given to Plotkin-Shmoys-Tardos [36] and Young [43]. This procedure also appears in Gawrychowski et al. [9]. We give the procedure in Algorithm 1 and state a version of Algorithm 1 with general epsilon in Algorithm 5.

■ **Algorithm 5** Obtain a Packing of Weight at least $(1 - \epsilon)c/2$ from a Graph G .

Let G be a graph with m edges and n vertices.

1. Initialize $\ell(e) \leftarrow 0$ for all edges e of G . Initialize multiset $P \leftarrow \emptyset$. Initialize $W \leftarrow 0$.
2. Repeat the following:
 - a. Find a minimum spanning tree T with respect to $\ell(\cdot)$.
 - b. Set $\ell(e) \leftarrow \ell(e) + \epsilon^2/(3 \ln m)$ for all $e \in T$. If $\ell(e) > 1$, return W, P .
 - c. Set $W \leftarrow W + \epsilon^2/(3 \ln m)$.
 - d. Add T to P .

We now give the general form of Lemma 3 with proof.

► **Lemma 16** ([36, 42, 43]). *Given $0 < \epsilon < 1$ and an undirected unweighted graph G with m edges, n vertices, and minimum cut c , Algorithm 5 returns a weighted packing of weight at least $(1 - \epsilon)c/2$ in $O(mc \log n)$ time.*

Proof. On each iteration, the weight of some tree is increased by $\epsilon^2/(3 \ln m)$. Since the weight of the resulting packing is bounded by c , there are at most $3c \ln m/\epsilon^2 = O(c \log n)$ iterations. The bottleneck in each iteration is the time to compute a minimum spanning tree in G . With an $O(m)$ time minimum spanning tree algorithm [24] our final time complexity is $O(mc \log n)$; an alternative way to achieve this runtime when Algorithm 5 is used in Algorithm 6 was shown in Section 3. Correctness is given via Thorup and Karger [42], Young [43], and Plotkin-Shmoys-Tardos [36]. ◀

Our full procedure for obtaining $\Theta(\log n)$ spanning trees for the rest of the algorithm is given in Algorithm 2. We give a version of Algorithm 2 with general epsilons in Algorithm 6.

We give the generalization of Lemma 4 for Algorithm 6 below.

► **Lemma 17.** *Algorithm 6 returns a collection of $\Theta(\log n)$ spanning trees of G in time $O(m \log^3 n)$ such that the minimum cut of G 2-respects at least one tree in the collection with high probability.*

Proof. We first prove correctness. Consider general epsilons $\epsilon_1, \epsilon_2, \epsilon_3 > 0$, where in Algorithm 2, $\epsilon_1 = 1/100$ is the real-weight approximation, $\epsilon_2 = 1/6$ is the approximation for Lemmas 14 and 15, and $\epsilon_3 = 1/5$ is the approximation for Algorithm 1 to return a packing of size $(1 - \epsilon_3)c/2$ or greater.

Suppose for a particular c' that $c' \geq 6c$, where c is the size of the minimum cut in G' . Then by Lemma 15, H will have minimum cut of size less than $b/3 = (d + 2) \ln n/\epsilon_2^2$ with high probability. A maximum tree packing of H will have weight at most \hat{c} , the weight of the minimum cut in H , and thus the weight of the tree packing found by Algorithm 5 will be at most $b/3 < \frac{1}{2}(1 - \epsilon_3)(1 + \epsilon_2)^{-1}b$ because $(1 + \epsilon_2)^{-1}(1 - \epsilon_3) > 2/3$. Therefore Algorithm 6 will proceed to the next iteration with $c' \leftarrow c'/2$. Note that the overall probability of failure from any of the $O(\log n)$ iterations of this step is at most $O(\log n \cdot n^{-(d+2)}) \leq n^{-d}$ for sufficiently large n .

■ **Algorithm 6** Obtain $\Theta(\log n)$ Spanning Trees for the 2-respect Algorithm.

Let d denote the exponent in the probability of success $1 - 1/n^d$. Let $\epsilon_1, \epsilon_2, \epsilon_3 > 0$ be constants of approximation such that $f = 3/2 - (\frac{2+\epsilon_1}{2-\epsilon_1})(1+\epsilon_2)(1-\epsilon_3)^{-1} > 0$ and $(1+\epsilon_2)^{-1}(1-\epsilon_3) > 2/3$. Let $b = 3(d+2) \ln n/\epsilon_2^2$.

1. Form graph G' from G by first normalizing the edge weights of G so the smallest non-zero edge weight has weight 1, then multiplying each edge weight by ϵ_1^{-1} and rounding to the nearest integer. Let U be an upper bound for the size of the minimum cut of G' .
 2. Initialize $c' \leftarrow U$. Repeat the following:
 - a. Construct H in the following way: for each edge e of G' , let e have weight in H drawn from the binomial distribution with probability $p = \min(b/c', 1)$ and number of trials the weight of e in G' . Cap the weight of any edge in H to at most $\lceil(1+\epsilon_2)12b\rceil$.
 - b. Run Algorithm 5 on H with approximation ϵ_3 , considering an edge of weight w as w parallel edges. There are three cases:
 - i. If $p = 1$, set P to the packing returned and skip to step 3.
 - ii. If the returned packing is of weight $\frac{1}{2}(1-\epsilon_3)(1+\epsilon_2)^{-1}b$ or greater, set $c' \leftarrow c'/6$ and repeat steps 2a and 2b, setting P to the packing returned and then proceeding to step 3.
 - iii. Otherwise, repeat steps 2a and 2b with $c' \leftarrow c'/2$.
 3. Return $\lceil -d \ln n / \ln(1-f) \rceil$ trees sampled uniformly at random proportional to their weights from P .
-

Now suppose Algorithm 5 returns a tree packing of weight $\frac{1}{2}(1-\epsilon_3)(1+\epsilon_2)^{-1}b$ or greater. By the above, $c' < 6c$ with high probability. If $c' \leq c$, Lemma 14 says that the weight of the minimum cut is at least $(1+\epsilon_2)^{-1}b$ with high probability, unless $p > 1$. In the latter case, this implies the weight of the minimum cut is $O(\log n)$ and there is no need to apply sampling to G' . Consider the former case. The tree packing is of weight at least $(1-\epsilon_3)$ times half the minimum cut. It follows that the tree packing will be of weight at least $\frac{1}{2}(1-\epsilon_3)(1+\epsilon_2)^{-1}b$. The consequence of this is that if a tree packing of this weight or greater is found in step 2b, in addition to the bound $c' < 6c$, we also know $c' > c/2$ with high probability, since whenever $c' \leq c$, Lemma 14 says the packing will have weight at least $\frac{1}{2}(1-\epsilon_3)(1+\epsilon_2)^{-1}b$, and we decrease c' by a factor of 2 in each iteration. Therefore, if we set $c' \leftarrow c'/6$, then in the next iteration we will have $c/12 < c' < c$.

Now consider the next iteration when the tree packing is returned. In sampling H , we only preserve weights in H up to $\lceil(1+\epsilon_2) \cdot 12b\rceil$. Since $c' > c/12$, the expected size of the minimum cut in H is at most $12b = 12 \cdot 3(d+2) \ln n/\epsilon_2^2$. Thus, with high probability, by Lemma 14, the size of the minimum cut in H is at most $(1+\epsilon_2)12b$, and as explained previously, we can afford to remove the capacity of any edge beyond $(1+\epsilon_2)12b$ without impacting the returned packing. Now by Lemma 13 with $\alpha \leq \frac{2+\epsilon_1}{2-\epsilon_1}(1+\epsilon_2)$ and $\beta \geq \frac{1}{2}(1-\epsilon_3)$, a fraction of at least $f = 3/2 - (\frac{2+\epsilon_1}{2-\epsilon_1})(1+\epsilon_2)(1-\epsilon_3)^{-1}$ of the trees in the packing found will 2-constrain the minimum cut of G . The probability that no tree in a sample of size t 2-constrains the minimum cut is $(1-f)^t$. Solving for t in $(1-f)^t = n^{-d}$ yields $t = -d \ln n / \ln(1-f)$. Therefore with probability at least $1 - 1/n^d$, at least one tree in the returned sample will 2-constrain the minimum cut.

Time complexity can be proven as follows. Sampling H can be done in $O(m \log n)$ time, as explained previously. Algorithm 5 runs in $O(m' \hat{c} \log^2 n)$ time using a textbook $O(m \log n)$ minimum spanning tree algorithm, where \hat{c} is the value of the minimum cut in H and m' is

12:18 Simple Min-Cut in Near-Linear Time

the number of edges in H , where weighted edges are considered parallel unit weight edges. Due to the sampling procedure, $m' = O(m \log n)$. To reduce this complexity, we can either use a linear time minimum spanning tree algorithm [24] or the implementation trick given in Section 3. If we use the latter, we reduce the effective m' needed in Algorithm 5 to $O(m)$. Further, in expectation, the value of the minimum cut \hat{c} of H doubles in each iteration of Algorithm 6. A high probability statement can be made via an argument similar to Lemma 15. Therefore the cost of running Algorithm 5 doubles in each iteration, with the final cost being $O(m \log^3 n)$, since $\hat{c} = O(\log n)$ by Lemma 14. This is a geometric series, so the entire cost is $O(m \log^3 n)$, and so Algorithm 6 runs in $O(m \log^3 n)$ time with high probability. ◀

Since Algorithm 5 returns $O(\log n)$ trees, we could avoid sampling trees from the weighted packing and instead return all of them. We keep the sampling in Algorithm 6 because, depending on the constants, sampling may require less trees. Further, the above version of Algorithm 5 is more versatile in that the packing algorithm can be changed. Observe that the entire algorithm is still only correct with high probability, since we required sampling G' to construct graph H . Finally, returning all trees from Algorithm 5 does not actually allow us to relax ϵ_1 , ϵ_2 , or ϵ_3 . The condition $f = 3/2 - (1 + \epsilon_1)(1 + \epsilon_2)(1 - \epsilon_3)^{-1} > 0$ is satisfied for all values of α and β that guarantee at least one tree in a weighted packing of weight βc 2-constrains a cut of weight αc given by Lemma 13.

Algorithm 6 is slightly different than the approach taken by Karger [23]. In particular, Karger sparsifies edges of H to have $m' = O(n \log n)$ and replaces an $O(m \log n)$ time minimum spanning tree computation in the tree packing algorithm with an $O(m)$ one, avoiding the implementation trick of Gawrychowski et al. [9]. This gives complexity $O(n \log^3 n)$ for finding the $\Theta(\log n)$ spanning trees. However, since the remaining part of the algorithm also takes $O(m \log^3 n)$ time, we avoid these optimizations to simplify our procedures.


Parameterized Study of Steiner Tree on Unit Disk Graphs

Sujoy Bhore 

Algorithms and Complexity Group, TU Wien, Austria
sujoy@ac.tuwien.ac.at

Paz Carmi 

Ben-Gurion University of the Negev, Beersheba, Israel
carmip@cs.bgu.ac.il

Sudeshna Kolay 

Indian Institute of Technology Kharagpur, India
skolay@cse.iitkgp.ac.in

Meirav Zehavi 

Ben-Gurion University of the Negev, Beersheba, Israel
meiravze@bgu.ac.il

Abstract

We study the STEINER TREE problem on unit disk graphs. Given a n vertex unit disk graph G , a subset $R \subseteq V(G)$ of t vertices and a positive integer k , the objective is to decide if there exists a tree T in G that spans over all vertices of R and uses at most k vertices from $V \setminus R$. The vertices of R are referred to as *terminals* and the vertices of $V(G) \setminus R$ as *Steiner* vertices. First, we show that the problem is NP-hard. Next, we prove that the STEINER TREE problem on unit disk graphs can be solved in $n^{O(\sqrt{t+k})}$ time. We also show that the STEINER TREE problem on unit disk graphs parameterized by k has an FPT algorithm with running time $2^{O(k)}n^{O(1)}$. In fact, the algorithms are designed for a more general class of graphs, called clique-grid graphs [16]. We mention that the algorithmic results can be made to work for STEINER TREE on disk graphs with bounded aspect ratio. Finally, we prove that STEINER TREE on disk graphs parameterized by k is W[1]-hard.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Unit Disk Graphs, FPT, Subexponential exact algorithms, NP-Hardness, W-Hardness

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.13

Funding *Sujoy Bhore*: Research of Sujoy Bhore was supported by the Austrian Science Fund (FWF), grant P 31119.

Paz Carmi: Research of Paz Carmi was partially supported by the Lynn and William Frankel Center for Computer Science and by Grant 2016116 from the United States-Israel Binational Science Foundation.

Meirav Zehavi: Research of Meirav Zehavi was supported by the Israel Science Foundation (ISF) grant no. 1176/18 and United States - Israel Binational Science Foundation (BSF) grant no. 2018302.

Acknowledgements We are grateful to the anonymous reviewers for their helpful comments.

1 Introduction

Given a graph G with a weight function $w : E(G) \rightarrow \mathbb{R}^+$ and a subset $R \subseteq V(G)$ of vertices, a Steiner tree is an acyclic subgraph of G spanning all vertices of R . The vertices of R are usually referred to as *terminals* and the vertices of $V(G) \setminus R$ as *Steiner* vertices. The MINIMUM STEINER TREE problem is to find a Steiner tree T such the total weight of $E(T)$ is minimized. The decision version of this is the STEINER TREE problem, where given a graph G , a subset $R \subseteq V(G)$ of vertices and a positive integer k , the objective is to determine if



© Sujoy Bhore, Paz Carmi, Sudeshna Kolay, and Meirav Zehavi;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 13; pp. 13:1–13:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

there exists a Steiner tree T in G for the *terminal* set R such that the number of *Steiner* vertices in T is at most k . The STEINER TREE problem is one of Karp's classic NP-complete problems [22]; moreover, that makes the optimization problem NP-hard.

A special case of the MINIMUM STEINER TREE problem is the METRIC STEINER TREE problem. Given a complete graph $G = (V, E)$, each vertex corresponds to a point in a metric space, and for each edge $e \in E$ the weight $w(e)$ corresponds to the distances in the space. In other words, the edge weights satisfy the triangle inequality. It is well known that, given an instance of the non-metric Steiner tree problem, it is possible to transform it in polynomial time into an equivalent instance of the METRIC STEINER TREE problem. Moreover, this transformation preserves the approximation factor [30]. The EUCLIDEAN STEINER TREE problem or GEOMETRIC STEINER TREE problem takes as input n points in the plane. The objective is to connect them by lines of minimum total length in such a way that any two points may be interconnected by line segments either directly or via other points and line segments. The MINIMUM STEINER TREE problem is NP-hard even in Euclidean or Rectilinear metrics [18].

Arora [2] showed that the EUCLIDEAN STEINER TREE and RECTILINEAR STEINER TREE problems can be efficiently approximated arbitrarily close to the optimal. Several approximation schemes have been proposed over the years on MINIMUM STEINER TREE for graphs with arbitrary weights [4, 7, 23, 27]. Although the Euclidean version admits a PTAS, it is known that the METRIC STEINER TREE problem is APX-complete. There is a polynomial-time algorithm that approximates the minimum Steiner tree to within a factor of $\ln(4) + \epsilon \approx 1.386$ [8]; however, approximating within a factor $\frac{96}{95} \approx 1.0105$ is NP-hard [3].

The decision version, STEINER TREE is well-studied in parameterized complexity. A well-studied parameter for the STEINER TREE is the number of terminals $t = |R|$. It is known that the STEINER TREE is FPT for this parameter due to the classical result of Dreyfus and Wagner [13]. Fuchs et al. [17] and Nederlof [26] gave alternative algorithms for STEINER TREE parameterized by t with running times that are not comparable with the Dreyfus and Wagner algorithm. On the other hand, STEINER TREE parameterized by the number of Steiner vertices k is W[2]-hard [12]. Hence, the focus has been on designing parameterized algorithms for graph subclasses like planar graphs [20], d -degenerate graphs [29], etc. In [15], Dcořák et al. designed an efficient parameterized approximation scheme (EPAS) for the STEINER TREE parameterized by k ¹.

In this paper, we study the STEINER TREE problem on unit disk graphs when the parameter is the number of Steiner vertices k . Unit disk graphs are the geometric intersection graphs of unit circles in the plane. That is, given n unit circles in the plane, we have a graph G where each vertex corresponds to a circle such that there is an edge between two vertices when the corresponding circles intersect. Unit disk graphs have been widely studied in computational geometry and graph algorithms due to their usefulness in many real-world problems, e.g., optimal facility location [31], wireless and sensor networks; see [19, 21]. These led to the study of many NP-complete problems on unit disk graphs; see [9, 14].

There are some works on variants of MINIMUM STEINER TREE on unit disk graphs in the approximation paradigm. Li et al. [24] studied node-weighted Steiner trees on unit disk graphs, and presented a PTAS when the given set of vertices is c -local. Moreover, they used this to solve the node-weighted connected dominating set problem in unit disk graphs and obtained a $(5 + \epsilon)$ -approximation algorithm. In [5], Biniat et al. studied the FULL STEINER

¹ For any $\epsilon > 0$ computes a $(1 + \epsilon)$ approximation in time $f(p, \epsilon) \times n^{O(1)}$ for a computable function f independent of n .

TREE² problem on unit disk graphs. They presented a 20-approximation algorithm for this problem, and for λ -precise graphs gave a $(10 + \frac{1}{\lambda})$ -approximation algorithm where λ is the length of the longest edge. Although there have been a plethora of work on variants of the MINIMUM STEINER TREE problem on unit disk graphs in approximation algorithms, hardly anything is known in parameterized complexity for the decision version. In this regard, we refer to the work of Marx et al. [25] who investigated the parameterized complexity of the MINIMUM STEINER TREE problem on planar graphs, where the number of terminals (k) is regarded as the parameter. They have designed an $n^{O(\sqrt{k})}$ -time exact algorithm, and showed that this problem on planar graphs cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$, assuming ETH. However, these results do not directly apply on unit disk graphs as unit disk graphs can contain very large cliques, but, then planar graphs contains arbitrarily large stars. Recently, Berg et al. [11] showed that the STEINER TREE problem can be solved in $2^{O(n^{1-\frac{1}{d}})}$ time on intersection graphs of d -dimensional similarly-sized fat objects, for some $d \in \mathbb{Z}_+$.

More often than not, the geometric intersection graph families such as unit disk graphs, unit square intersection graphs, rectangle intersection graphs, provide additional geometric structure that helps to generate algorithms. In this paper, our objective is to understand parameterized tractability landscape of the STEINER TREE problem on unit disk graphs.

Our Results

First in Section 3, we show that STEINER TREE on unit disk graphs is NP-hard. Then, in Section 4, we design a subexponential algorithm for the STEINER TREE problem on unit disk graphs parameterized by the number of terminals t and the number of Steiner vertices k .

► **Theorem 1.** *STEINER TREE on unit disk graphs can be solved in $n^{O(\sqrt{t+k})}$ time.*

The approach to design this subexponential algorithm is very similar to that used in [16]. First, we apply a Baker-like shifting strategy to create a family \mathcal{F} of instances (of EXACT STEINER TREE, which is a variant of STEINER TREE) such that if the input instance (G, R, t, k) is a yes-instance then there is at least one constructed instance in \mathcal{F} that is a yes-instance of EXACT STEINER TREE. On the other hand, if (G, R, t, k) is a no-instance of STEINER TREE, then no instance of \mathcal{F} is a yes-instance of EXACT STEINER TREE. With the knowledge that the answer is preserved in the family \mathcal{F} , we design a dynamic programming subroutine to solve EXACT STEINER TREE on each of the constructed instances of \mathcal{F} .

Next, in Section 5, we show that the STEINER TREE on unit disk graphs has an FPT algorithm when parameterized by k .

► **Theorem 2.** *STEINER TREE on unit disk graphs can be solved in $2^{O(k)}n^{O(1)}$ time.*

Here, we show that solving the STEINER TREE problem on an instance (G, R, t, k) is equivalent to solving the problem on an instance (G', R', t', k) where the graph G' is obtained by contracting all connected components of $G[R]$. Although G' loses all geometric properties, we show that the number of terminals in R' is only dependent on k . This essentially changes the problem to running the Dreyfus-Wagner algorithm on (G', R', t', k) .

Both the results in Theorem 1 and 2 are shown to work for a superclass of graphs, called clique-grid graphs. We would like to remark that the algorithms can also be made to work for disk graphs with constant aspect ratio.

² A full Steiner tree is a Steiner tree which has all the terminal vertices as its leaves

Finally, in contrast, in Section 6 we prove that the STEINER TREE problem for disk graphs is $W[1]$ -hard, parameterized by the number Steiner vertices k . The STEINER TREE problem is known to be $W[2]$ -hard on general graphs [12]. However, it is not clear how to use that reduction for disk graphs. We show a reduction of our problem from GRID TILING with $\geq [10]$, ruling out the possibility of a $f(k)n^{o(k)}$ time algorithm for any function f , assuming ETH.

► **Theorem 3.** *The STEINER TREE problem on disk graphs is $W[1]$ -hard, parameterized by the number of Steiner vertices k .*

2 Preliminaries

The set $\{1, 2, \dots, n\}$ is denoted as $[n]$. For a graph G , and a subset $V' \subseteq V(G)$, $G[V']$ denotes the subgraph induced on V' . The EXACT STEINER TREE problem takes as input a graph G , a terminal set R with t terminals and a positive integer k . The aim is to determine whether there is a Steiner tree T in G for R that has exactly k Steiner vertices. A Steiner tree with at most k Steiner vertices is called a k -Steiner tree while one with exactly k Steiner vertices is called an exact k -Steiner tree. Note that if T is an exact k -Steiner tree then $|V(T)| = t + k$. When the STEINER TREE or EXACT STEINER TREE problem is restricted to taking input graphs only from a graph class \mathcal{G} , then these variants are referred to as STEINER TREE on \mathcal{G} and EXACT STEINER TREE on \mathcal{G} , respectively.

► **Observation 4.** *A tree T is a k -Steiner tree for an instance (G, R, t, k) if and only if T is an exact k' -Steiner tree for the instance (G, R, t, k') of EXACT STEINER TREE for some $k' \leq k$.*

► **Definition 5.** [16] *A graph G is a clique-grid graph if there is a pair $p, p' \in \mathbb{N}$ and a function $f : V(G) \rightarrow [p] \times [p']$ such that the following conditions hold:*

1. *For all $(i, j) \in [p] \times [p']$, $f^{-1}(i, j)$ is a clique in G .*
 2. *For all $uv \in E(G)$, if $f(u) = (i, j)$ and $f(v) = (i', j')$ then $|i - i'| \leq 2$ and $|j - j'| \leq 2$.*
- Such a function f is called a representation of the graph G .*

Unit disk graphs are clique-grid graphs [16]. Next, we define a representation of a clique-grid graph called a cell graph.

► **Definition 6.** [16] *Given a clique-grid graph G with representation $f : V(G) \rightarrow [p] \times [p']$, the cell graph $\text{cell}(G)$ is defined as follows:*

- $V(\text{cell}(G)) = \{v_{ij} \mid i \in [p], j \in [p'], f^{-1}(i, j) \neq \emptyset\}$,
- $E(\text{cell}(G)) = \{v_{ij}v_{i'j'} \mid (i, j) \neq (i', j'), \exists u \in f^{-1}(i, j) \text{ and } \exists v \in f^{-1}(i', j') \text{ such that } uv \in E(G)\}$.

For each vertex $v_{ij} \in V(\text{cell}(G))$, the pair (i, j) is also called a cell of G and by definition corresponds to a non-empty clique of G . A vertex $v \in V(G)$ is said to be in the cell (i, j) if $f(v) = (i, j)$. The neighbour of a cell $\mathcal{C} = (i, j)$ in a cell $\mathcal{C}' = (i', j') \neq \mathcal{C}$ are $\{v \in V(G) \mid f(v) = (i', j'), \exists u \text{ such that } f(u) = (i, j) \text{ and } uv \in E(G)\}$.

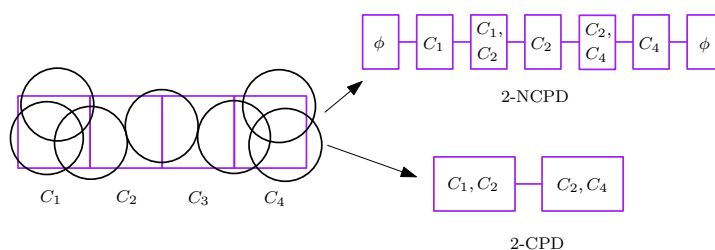
Let G be a graph. A *path decomposition* of a graph G is a pair $\mathcal{T} = (P, \beta : V(P) \rightarrow 2^{V(G)})$, where P is a path where every node $p \in V(P)$ is assigned a subset $\beta(p) \subseteq V(G)$, called a bag, such that the following conditions hold: (i) $\bigcup_{p \in V(P)} \beta(p) = V(G)$, (ii) for every edge $xy \in E(G)$ there is a $p \in V(P)$ such that $\{x, y\} \subseteq \beta(p)$, and (iii) for any $v \in V(G)$ the subgraph of P induced by the set $\{p \mid v \in \beta(p)\}$ is connected. A path decomposition will also be denoted as a sequence of bags $\{\beta(p_1), \beta(p_2), \dots, \beta(p_q)\}$ where $P = p_1p_2 \dots p_q$. The *width* of a path decomposition is $\max_{p \in V(P)} |\beta(p)| - 1$. The *pathwidth* of G is the minimum width

over all path decompositions of G and is denoted by $\text{pw}(G)$. Given a path decomposition of a graph G , we say it is rooted at exactly one of the two degree one vertices of the underlying path.

► **Definition 7.** [16] A path decomposition $\mathcal{T} = (P, \beta)$ of a clique-grid graph G with representation $f : V(G) \rightarrow [p] \times [p']$ is a nice ℓ -clique path decomposition (ℓ -NCPD) if for the root r of P , $\beta(r) = \emptyset$ and for each $v \in V(P)$ the following hold:

1. There are at most ℓ cells $\{(i_1, j_1), (i_2, j_2), \dots, (i_\ell, j_\ell)\}$ such that $\beta(v) = \bigcup_{p=1}^{\ell} f^{-1}(i_p, j_p)$,
2. The node v is one of the following types: (i) Leaf node where $\beta(v) = \emptyset$, (ii) Forget node where v has exactly one child u and there is a cell $(i, j) \in [p] \times [p']$ such that $f^{-1}(i, j) \subseteq \beta(u)$ and $\beta(v) = \beta(u) \setminus f^{-1}(i, j)$, (iii) Introduce node where v has exactly one child u and there is a cell $(i, j) \in [p] \times [p']$ such that $f^{-1}(i, j) \subseteq \beta(v)$ and $\beta(u) = \beta(v) \setminus f^{-1}(i, j)$,

See Figure 1 for an example of an NCPD. A path decomposition for a clique-grid graph G with representation f where only property 1 of Definition 7 is true for a positive number ℓ is referred to as an ℓ -CPD.



■ **Figure 1** An illustration of nice 2-clique path decomposition.

3 NP-Hardness of Steiner Tree on Unit Disk Graphs

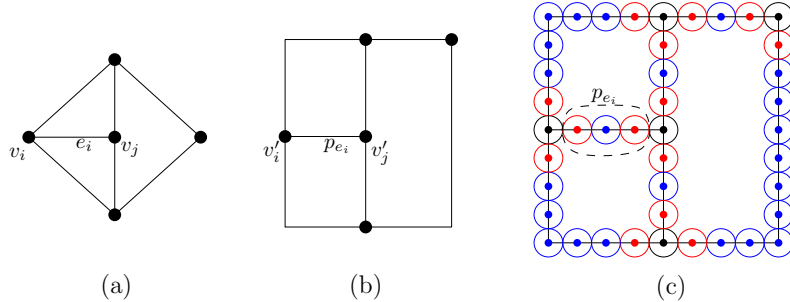
In this section, we consider the STEINER TREE problem on unit disk graphs and prove that this problem is NP-hard. We show a reduction from CONNECTED VERTEX COVER in planar graphs with maximum degree 4. The reduction is very similar to that in [1].

► **Theorem 8.** *The STEINER TREE problem on unit disk graphs is NP-hard.*

Proof. We show a reduction from the CONNECTED VERTEX COVER in planar graphs with maximum degree 4 problem, which is known to be NP-hard [18]. Given a planar graph G with maximum degree 4 and an integer k , the CONNECTED VERTEX COVER problem asks to find if there exists a vertex cover D for G such that the subgraph induced by D is connected and $|D| \leq k$. We adopt the proof of Abu-Affash [1], where it was shown that the k -BOTTLENECK FULL STEINER TREE problem is NP-hard. We make this reduction compatible for unit disk graphs. Given a planar graph G with maximum degree 4 and an integer k , we construct a unit disk graph $G_{\mathcal{C}}$ where $V(G_{\mathcal{C}}) = \mathcal{C}$ in polynomial time, where $V(G_{\mathcal{C}})$ is divided into two sets of unit disks R and S , denoted by Steiner and terminals, respectively. Let $V(G) = \{v_1, v_2, \dots, v_n\}$ and let $E(G) = \{e_1, e_2, \dots, e_m\}$. Then, we compute an integer k' such that G has a connected vertex cover D of size k if and only if there exists a STEINER TREE with at most k' Steiner vertices of $G_{\mathcal{C}}$.

As an intermediate step we build a rectangular grid graph G' . First, we embed G on a rectangular grid, with distance at least 8 between adjacent vertices. Each vertex $v_i \in V(G)$ corresponds to a grid vertex, and each edge $e = v_i v_j \in E(G)$ corresponds to a rectilinear

path comprised of some horizontal and vertical grid segments with endpoints corresponding to v_i and v_j . Let $V(G') = \{v'_1, \dots, v'_n\}$ be the grid points corresponding to the vertices of $V(G)$, and let $E(G') = \{p_{e_1}, \dots, p_{e_m}\}$ be the set of paths corresponding to the edges of $E(G)$. Moreover, these paths are pairwise disjoint; see Figure 2(b). This embedding can be done in $O(n)$ time and the size of the grid is at most $n - 2$ by $n - 2$; see [28]. Next, we construct an unit disk graph G_C from G' . First, we replace each grid vertex $v'_i \in V(G')$ by an unit disk. Let $C = \{c_1, \dots, c_n\}$ be the set of unit disks centered at the grid points corresponding to the vertices of $V(G')$. For the sake of explanation we call these disks grid point disks. At this point, the unit disk graph is not connected due to the edge length which we have taken between any two adjacent vertices in the grid graph. In fact this length ensures that there are no undesirable paths other than the ones in G . Next, we place two sets of disks on each path $p_{e_i} \in E(G')$. Let $|p_{e_i}|$ be the total length of the grid segments of p_{e_i} . We place two Steiner disks on p_{e_i} , such that each one of them is adjacent to a grid point disk corresponding to p_{e_i} and the distance between their centers is exactly 2. Next, we place $|p_{e_i}| - 6/2$ many terminals disks on p_{e_i} such that the distance between any two adjacent centers is exactly 2. See Figure 2(c) for detailed explanation. Let $s(e_i)$ be the set of Steiner disks and $t(e_i)$ be the set of terminal disks placed to p_{e_i} . The terminal set $R = \bigcup_{e_i \in E(G')} t(e_i)$; the Steiner set $S = C \cup \bigcup_{e_i \in E(G')} s(e_i)$. $V(G_C) = R \cup S$ and G_C is the intersection graph induced by $V(G_C)$. Finally, we set $k' = m + 2k - 1$. Observe that, for any path p_{e_i} , the terminal set $t(e_i)$ itself form a Steiner tree without any Steiner disks. However, in order to make that tree connected we need at least one of Steiner disks from $s(e_i)$. This completes the construction.



■ **Figure 2** (a) A planar graph G of maximum degree 4, (b) the intermediate rectilinear embedding G' of G , (c) the unit disk graph G_C ; the black disks are corresponding to the grid vertices of G' , the blue disks are Steiner disks and the red disks are the terminal disks.

In the forward direction, suppose G has a connected vertex cover D of size at most k . We construct a Steiner tree of R in the following manner. For each edge e_i , we simply take the terminal path induced by $t(e_i)$. Now, let T_S be any spanning tree of the subgraph of G induced by D , containing $|D| - 1$ edges. The existence of such a spanning tree is ensured since D is a connected vertex cover of G . For each edge $e = v_i v_j \in T_S$ we connect the corresponding disks c_i, c_j by two Steiner red disks adjacent to them. Then, for each edge $e = v_i v_j \in G \setminus T_S$ we select one endpoint that is in D (say v_i) and connect c_i to the tree by its adjacent disk. The constructed tree is a Steiner tree of R consisting $|D| + 2(|D| - 1) + (m - (|D| - 1))$ which is $m + 2k - 1$.

Conversely, let there exists a Steiner tree T of R with at most k' Steiner disks. Let $D \subseteq C$ be the set of vertices that appear in T , and let T' be the subtree of T spanning over D . For each subset $t(e_i) \subseteq R$, let T_{e_i} be the subtree of T_{e_i} spanning the vertices in $t(e_i)$. By the

above construction, T_{e_i} does not require any Steiner disk. Moreover, it is easy to see that in any valid solution T_{e_i} must be connected to at least one endpoint of D . This implies that the set of vertices in G corresponding to the vertices in D is a connected vertex cover of G . Moreover a tree T_{e_i} which also a subtree of T is connected to D via two Steiner disks of $s(e_i)$. Therefore, T_S contains $|D| + 2(|D| - 1) + (m - (|D| - 1))$ many Steiner disks. We started with the tree T with at most $k' = m + 2k - 1$ many Steiner disks. This completes the proof. ◀

4 Subexponential Exact Algorithm for Steiner Tree on Unit Disk Graphs

In this section, we prove Theorem 1 by designing a sub-exponential algorithm for the STEINER TREE problem on unit disk graphs parameterized by $t + k$, where t is the number of terminals and k is an upper bound on the number of Steiner vertices. In fact, our aim for this section is to design a subexponential algorithm for STEINER TREE on clique-grid graphs and as unit disk graphs are clique-grid graphs [16], this would imply the algorithm proposed in Theorem 1.

► **Lemma 9.** *The STEINER TREE problem on clique-grid graphs can be solved in $n^{O(\sqrt{t+k})}$ time.*

For the rest of the section, we concentrate on proving Lemma 9. Informally, we first apply a Baker-like shifting strategy to create a family \mathcal{F} of instances of EXACT STEINER TREE that preserves the answer for the input instance (G, R, t, k) of STEINER TREE: if (G, R, t, k) is a yes-instance then there is at least one constructed instance in \mathcal{F} that is a yes-instance of EXACT STEINER TREE; if (G, R, t, k) is a no-instance of STEINER TREE then all instances of \mathcal{F} are no-instances of EXACT STEINER TREE. As a second step, we design a dynamic programming subroutine to solve EXACT STEINER TREE on each of the constructed instances of \mathcal{F} , which is enough to solve the STEINER TREE problem on (G, R, t, k) .

Before we describe the subexponential algorithm, we state some properties of Steiner trees in clique-grid graphs.

► **Observation 10.** *Consider a k -Steiner tree T for a clique-grid graph G with representation f , such that the set $\{uv \in E(T) | f(u) \neq f(v)\}$ is minimised over all k -Steiner trees for G . Let $\mathcal{C} = (i, j)$ be a cell of G . Then there are at most 24 edges with one endpoint in \mathcal{C} and the other endpoint in another cell.*

Proof. We claim that in the k -Steiner tree where the set $\{uv \in E(T) | f(u) \neq f(v)\}$ is minimised, there can be at most one neighbour of \mathcal{C} in each cell $\mathcal{C}' \neq \mathcal{C}$. Suppose that \mathcal{C}' is a cell that contains at least two neighbours of \mathcal{C} . Let two such neighbours be u', v' . Note that $u'v'$ is an edge in $E(G)$. Let u, v (may be the same) be the neighbours of u, v , respectively in \mathcal{C} . Note that uv is an edge in $E(G)$. Thus adding the edge $u'v'$ and removing the edge uu' results in a connected graph containing all the terminals. The spanning tree of this connected graph has strictly less number of edges with endpoints in different cells, which is a contradiction to the choice of T .

By the definition of clique-grid graphs, $|i - i'|, |j - j'| \leq 2$. Thus, when we fix a cell \mathcal{C} there are at most 24 cells that can have neighbours of vertices in \mathcal{C} . Putting everything together, for the k -Steiner tree T where the set $\{uv \in E(T) | f(u) \neq f(v)\}$ is minimised, $|\{v | f(v) \neq (i, j), \exists u \text{ such that } f(u) = (i, j), uv \in E(G)\}| \leq 24$. ◀

► **Observation 11.** *Suppose there is a k -Steiner tree for a clique-grid graph G , and let T be a k -Steiner tree where the set $\{uv \in E(T) \mid f(u) \neq f(v)\}$ is minimised. Moreover, amongst k -Steiner trees where $\{uv \in E(T) \mid f(u) \neq f(v)\}$ is minimised, T has minimum number of Steiner points. Then, in T the number of Steiner vertices per cell is at most 24 .*

Proof. For the sake of contradiction, let $\mathcal{C} = (i, j)$ be a cell such that $|f^{-1}(i, j) \cap V(T)| \geq 24 + 1$. Then by Observation 10, there is at least one Steiner vertex $v \in f^{-1}(i, j) \cap V(T)$ such that it does not have any neighbours in $T \setminus f^{-1}(i, j)$. Consider the subgraph $T \setminus \{v\}$. Since the vertices of $f^{-1}(i, j)$ induce a clique, $T \setminus \{v\}$ is still a connected subgraph that contains all the terminals and strictly less number of Steiner vertices. Thus, a spanning tree of this connected subgraph contradicts the choice of T . ◀

Consider a k -Steiner tree T for an instance (G, R, t, k) of STEINER TREE where $\{uv \in E(T) \mid f(u) \neq f(v)\}$ is minimised and then the number of Steiner vertices is minimised. By Observation 4, T is an exact k' -Steiner tree for the instance (G, R, t, k') of EXACT STEINER TREE for some $k' \leq k$. Next, we define a *good family of instances* that preserve the answer for (G, R, t, k) of STEINER TREE.

► **Definition 12.** *For an instance (G, R, t, k) of STEINER TREE on clique-grid graphs where G has representation f , a good family of instances \mathcal{F} has the following properties:*

1. *For each instance (H, R, t, k') in the family, the input graph H is an induced subgraph of G that contains all vertices in R and $k' \leq k$. Note that H is also a clique-grid graph where $f|_{V(H)}$ is a representation.*
2. *(G, R, t, k) is a yes-instance of STEINER TREE if and only if there exists an instance $(H, R, t, k') \in \mathcal{F}$ which is a yes-instance of EXACT STEINER TREE.*
3. *For any instance $(H, R, t, k') \in \mathcal{F}$, H has a $7\sqrt{t+k}$ -NCPD.*

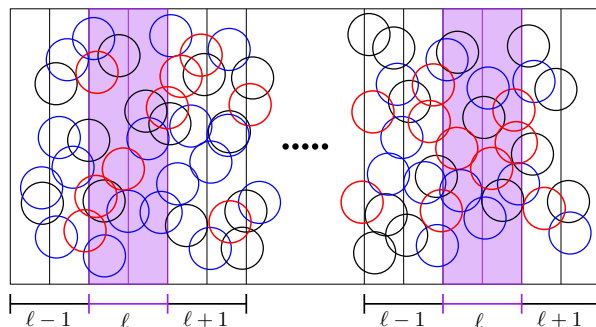
We show that given an instance (G, R, t, k) of STEINER TREE on clique-grid graphs, a good family of instances can be enumerated in subexponential time.

► **Lemma 13.** *Given an instance (G, R, t, k) for STEINER TREE on clique-grid graphs with G represented by f , a good family of instances \mathcal{F} can be computed in $n^{O(\sqrt{t+k})}$ time.*

Proof. Let T be a k -Steiner tree for G . In particular, T is an exact k' -Steiner tree for some $k' \leq k$ and $V(T) = t + k' \leq t + k$. First, we employ a Baker-like technique similar to [16] (please refer to Figure 3). Note that if G has n vertices and has representation $f : V(G) \rightarrow [p] \times [p']$, then $p, p' \leq n$. Thus, f represents G on the $n \times n$ grid. First we define a column of the $n \times n$ grid. For any $j \in [n]$ the set of cells $\{(i, j) \mid i \in [n]\}$ is called a column. There are n columns for the $n \times n$ grid. We partition the n columns of the $n \times n$ grid with $n/2$ blocks of two consecutive columns and label them from the set of labels $[\sqrt{t+k}]$. Formally, each set of consecutive columns $\{2i-1, 2i\}$, where $i \in [n/2]$ is labelled with $i \bmod \sqrt{t+k}$. Thus, all the two consecutive columns $\{2i-1, 2i\}$ are labelled with $i \bmod \sqrt{t+k}$.

Recall that an exact k' -Steiner tree T has at most $t+k$ vertices. Applying the pigeonhole principle, there is a label $\ell \in \{1, 2, \dots, \sqrt{t+k}\}$ such that the number of vertices from $V(T)$ which are in columns labelled ℓ is at most $\sqrt{t+k}$. As we do not know this k' -Steiner tree T , we guess the Steiner vertices of $V(T)$ which are in the columns labelled ℓ . The number of potential guesses is bounded by $n^{O(\sqrt{t+k})}$. Suppose Y' is the set of guessed Steiner vertices of $V(T)$ which are in the columns labelled by ℓ . Then we delete all the non-terminal vertices in columns labelled ℓ , except the vertices of Y' . Let S be the set of deleted non-terminal vertices. Let Y_R be the set of terminal vertices that are in columns labelled by ℓ . Let $Y = Y' \cup Y_R$. Notice that by choice of label ℓ , $|Y| \leq \sqrt{t+k}$. By Property 2 of clique-grid

graphs, $G \setminus (S \cup Y)$ is a disjoint union of clique-grid graphs each of which is represented by a function with at most $2\sqrt{t+k}$ columns. Formally, $G_1 = G[\bigcup_{j=1}^{2(\ell-1)} f^{-1}(*, j)]$ and $G_{i+1} = G[\bigcup_{j=i \cdot 2\ell+1}^{\min\{i \cdot 2\ell+2\sqrt{t+k}, n\}} f^{-1}(*, j)]$ for each $i \in \{1, \dots, n/\sqrt{t+k}\}$. Each G_i is a clique-grid graph with representation $f_i : V(G_i) \rightarrow [n] \times [2\sqrt{t+k}]$ defined as, $f_i(u) = (r, j)$, when $f(u) = (r, (i-1)2\ell+j)$. Thus, by Property 2 of Definition 5, $G \setminus (S \cup Y) = G_1 \uplus \dots \uplus G_{n/\sqrt{t+k}}$.



■ **Figure 3** An illustration of grid labelling. The blue disks are terminals, and the red and black disks are chosen Steiner vertices and not-chosen non-terminal vertices, respectively.

▷ **Claim 14.** The graph $G \setminus S$ has a $7\sqrt{t+k}$ -NCPD.

Proof. Suppose we are able to show that for each $i \in \{1, \dots, n/\sqrt{t+k}\}$ G_i has a $6\sqrt{t+k}$ -CPD. This results in a $6\sqrt{t+k}$ -CPD for $G \setminus (S \cup Y) = G_1 \uplus \dots \uplus G_{n/\sqrt{t+k}}$. Finally, note that $|Y| \leq \sqrt{t+k}$ and therefore the vertices of Y can belong to at most $\sqrt{t+k}$ cells. We add Y to all the bags in the $6\sqrt{t+k}$ -CPD for $G \setminus (S \cup Y)$ to obtain a $7\sqrt{t+k}$ -CPD for $G \setminus S$. We convert the $7\sqrt{t+k}$ -CPD of $G \setminus S$ into a NCPD using the known algorithm of [6]. Note that this results in a $7\sqrt{t+k}$ -NCPD.

What is left to show is that for each G_i there is a $6\sqrt{t+k}$ -CPD. First, for each G_i , we give a path decomposition with the following sequence of bags: $\{X_1, X_2, \dots, X_{n-2}\}$. This is done by defining each $X_i = f^{-1}(i, *) \cup f^{-1}(i+1, *) \cup f^{-1}(i+2, *)$. It is easy to check that this is a path decomposition of G_i . Note that since G_i has at most $2\sqrt{t+k}$ columns, the number of cells contained in each $X_j, j \in [n-1]$ is at most $6\sqrt{t+k}$. ◀

Finally, notice that from the definition of the constructed instances keeping in mind potential k -Steiner trees, (G, R, t, k) is a yes-instance of STEINER TREE if and only if there is an instance $(H, R, t, k') \in \mathcal{F}$ such that it is a yes-instance of EXACT STEINER TREE. Thus, accounting for guessing a label $\ell \in [\sqrt{t+k}]$ and the set Y of Steiner vertices and terminal vertices of a potential solution Steiner tree that belong to columns labelled ℓ , we obtain a good family of $n^{O(\sqrt{t+k})}$ instances for the given instance (G, R, t, k) . ◀

For the ease of our algorithm design, we make a slight modification of the NCPD for a constructed instance $(H, R, t, k') \in \mathcal{F}$: Upon fixing the label ℓ and a set Y of terminal vertices and potential Steiner vertices in the columns labelled by ℓ , we add the set Y in all the bags of the resulting NCPD for $G \setminus S$. Therefore, no bag is empty after this modification. In particular the first and the last bags of the modified path decomposition contain only the set Y . Also notice that as $|Y| \leq \sqrt{t+k}$, the new path decomposition of H is still an $O(\sqrt{t+k})$ -CPD. We call this new path decomposition of H a *modified NCPD*. Now, we are ready to prove Lemma 9.

13:10 Steiner Tree on Unit Disk Graphs

Proof of Lemma 9. As a first step of the algorithm, by Lemma 13 in $n^{O(\sqrt{t+k})}$ time we compute a good family of instances \mathcal{F} for the given instance (G, R, t, k) of STEINER TREE on clique-grid graphs. From Definition 12(2), (G, R, t, k) is a yes-instance of STEINER TREE if and only if there is an instance $(H, R, t, k') \in \mathcal{F}$ that is a yes-instance of EXACT STEINER TREE. Deriving from Definition 12(3), Lemma 13 and the construction of a modified NCPD, for each instance $(H, R, t, k') \in \mathcal{F}$, there is a modified $O(\sqrt{t+k})$ -NCPD for H , due to a guessed label ℓ and a guessed set Y of non-terminal vertices from columns labelled by ℓ such that the following hold: (i) $|Y| \leq \sqrt{t+k}$, (ii) if (H, R, t, k') is a yes-instance then there is an exact k' -Steiner tree T such that all vertices of Y are Steiner vertices in T . Let the modified NCPD using the set Y have the sequence of bags $\{X_1, X_2, \dots, X_q\}$. Recall that the definition of the modified NCPD ensures that $X_1 = X_q = Y$.

In the next step, our algorithm for STEINER TREE considers every instance $(H, R, t, k') \in \mathcal{F}$ and checks if it is a yes-instance of EXACT STEINER TREE. By Definition 12(2), this is sufficient to determine if (G, R, t, k) is a yes-instance of STEINER TREE.

For the rest of the proof we design a dynamic programming subroutine algorithm \mathcal{A} for EXACT STEINER TREE that takes as input an instance $(H, R, t, k') \in \mathcal{F}$ and uses its modified $O(\sqrt{t+k})$ -NCPD to determine whether it is a yes-instance of EXACT STEINER TREE. Suppose (G, R, t, k) is a yes-instance and consider a k -Steiner tree T for (G, R, t, k) where $\{uv \in E(T) | f(u) \neq f(v)\}$ is minimised and then the number of Steiner vertices in T is minimised. Using Observation 4, this is an exact k' -Steiner tree of G for some $k' \leq k$. By the construction in Lemma 13 note that there is an instance $(H, R, t, k') \in \mathcal{F}$ such that T is an exact k' -Steiner tree for (H, R, t, k') . The aim of the dynamic programming algorithm is to correctly determine that this particular instance (H, R, t, k') is a yes-instance. The algorithm \mathcal{A} is designed in such a manner that for such a yes-instance (H, R, t, k') the tree T will be the potential solution Steiner tree that behaves as a certificate of correctness.

The states of the dynamic programming algorithm store information required to represent the partial solution Steiner tree, which is the potential solution Steiner tree restricted to the graph seen so far. The states are of the form $\mathcal{A}[\ell, Q, \mathcal{Q} = Q_1 \uplus Q_2 \dots \uplus Q_b, \mathcal{P} = P_1 \uplus \dots \uplus P_b, k'']$ where:

- $\ell \in [q]$ denotes the index of the bag X_ℓ of the modified NCPD of H .
- $Q \subseteq X_\ell \setminus R$ is a set of at most $24 \cdot 7$ non-terminal vertices. For each cell $\mathcal{C} = (i, j)$ that belongs to X_ℓ , $|Q \cap f^{-1}(i, j)| \leq 24$.
- $\mathcal{Q} = Q_1 \uplus Q_2 \dots \uplus Q_b$ is a partition of Q with the property that for each cell $\mathcal{C} = (i, j)$, $Q \cap f^{-1}(i, j)$ is contained completely in exactly one part of \mathcal{Q} .
- The partition \mathcal{P} is over the vertex set $Q \cup (R \cap X_\ell)$. $Q \cap P_i = Q_i$. Also for each cell \mathcal{C} in X_ℓ , $\mathcal{C} \cap (Q \cup R)$ is completely contained in exactly one part of \mathcal{P} .
- The value k'' represents the total number of Steiner vertices used so far in this partial solution Steiner tree. $|Q| \leq k''$ holds.

Essentially, let T be an exact k' -Steiner tree for (H, R, t, k') if it is a yes-instance. For $\ell \in [q]$, let T_{ptl}^ℓ represent the partial solution Steiner tree when T is restricted to $H[\bigcup_{j=1}^\ell X_j]$. The partition \mathcal{P} represents the intersection of a component of T_{ptl}^ℓ with X_ℓ . The set Q is the set of Steiner vertices of T_{ptl}^ℓ in the bag X_ℓ and \mathcal{Q} is the partition of Q with respect to the components of T_{ptl}^ℓ . The number k'' denotes the total number of Steiner vertices in T_{ptl}^ℓ .

In order to show the correctness of \mathcal{A} we need to maintain the following invariant throughout the algorithm: (LHS) $\mathcal{A}[\ell, Q, \mathcal{Q} = Q_1 \uplus Q_2, \dots, Q_b, \mathcal{P} = P_1 \uplus P_2 \uplus P_b, k''] = 1$ if and only if (RHS) there is a forest T' as a subgraph of $H[\bigcup_{j=1}^\ell X_j]$ with b connected components D_1, \dots, D_b : $D_i \cap X_\ell = P_i$, $(D_i \setminus R) \cap X_\ell = Q_i$, the total number of non-terminal points in T' is k'' , for each cell \mathcal{C} the number of nonterminal vertices in $\mathcal{C} \cap T'$ is at most 24, and $R \cap (\bigcup_{j=1}^\ell X_j) \subseteq V(T')$.

Suppose the algorithm invariant is true. This means that if $\mathcal{A}[q, Y, Y, Y, k'] = 1$ then there is an exact k' -Steiner tree for (H, R, t, k') . On the other hand, suppose (G, R, t, k) is a yes-instance and has a k -Steiner tree T where $\{uv \in E(T) | f(u) \neq f(v)\}$ is minimised and then the number of Steiner vertices in T is minimised. By Observation 11, the number of Steiner vertices of T in each cell of G is bounded by 24. By Observation 4 and the construction in Lemma 13 note that there is a subset Y and an instance $(H, R, t, k') \in \mathcal{F}$ such that T is an exact k' -Steiner tree for (H, R, t, k') and $Y \subseteq V(T)$. Suppose the invariant of the algorithm is true. This means that if (G, R, t, k) is a yes-instance of STEINER TREE then there is a (H, R, t, k') for which $\mathcal{A}[q, Y, Y, Y, k'] = 1$.

Thus, proving the correctness of the algorithm \mathcal{A} amounts to proving the correctness of the invariant of \mathcal{A} . We prove the correctness of the invariant by induction on ℓ . If $\ell = 1$ then X_ℓ must be a **leaf bag**. By definition of the modified NCPD, the bag contains Y .

$\mathcal{A}[1, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$ if $Q = Y$, \mathcal{Q} is the partition of Y into the connected components in $H[Y]$, $\mathcal{P} = \mathcal{Q}$, $k'' = |Y|$. In all other cases, $\mathcal{A}[1, Q, \mathcal{Q}, \mathcal{P}, k''] = 0$.

First, suppose $\mathcal{A}[1, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. Then as X_1 does not contain any terminal vertices, (RHS) trivially is true for the cases when $\mathcal{A}[1, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. On the other hand, suppose (RHS) is true for $\ell = 1$. Again considering the cases when $\mathcal{A}[1, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$, (LHS) holds. So the invariant holds when $\ell = 1$.

Now, we assume that $\ell > 1$. Our induction hypothesis is that the invariant of the algorithm is true for all $1 \leq \ell' < \ell$. We show that the invariant is true for ℓ . There can be two cases:

Case 1: X_ℓ is a **forget bag** with exactly one child $X_{\ell-1}$: Let \mathcal{C} be the cell being forgotten in X_ℓ . Consider $\mathcal{A}[\ell, Q, \mathcal{Q} = Q_1, \dots, Q_b, \mathcal{P} = P_1 \dots P_b, k'']$.

Let $Q' \subseteq X_{\ell-1} \setminus R$ such that $Q \subseteq Q'$ and $Q' \setminus Q$ consists of a set of at most 24 non-terminal vertices from \mathcal{C} . Let $\mathcal{P}' = P'_1 \dots P'_b$ be a partition of $(Q' \cup R) \cap X_{\ell-1}$ such for each cell \mathcal{C}' in $X_{\ell-1}$, $\mathcal{C}' \cap (Q' \cup R)$ is completely contained in exactly one part. Also, $P_i = P'_i \setminus \mathcal{C}$. Moreover, consider the part P'_i such that $\mathcal{C} \cap (Q' \cup R) \subseteq P'_i$: $P'_i \setminus (\mathcal{C} \cap (Q' \cup R)) \neq \emptyset$. Let \mathcal{Q}' be the partition of Q' such that $Q' \cap P'_i = Q'_i$. If $\mathcal{A}[\ell-1, Q', \mathcal{Q}', \mathcal{P}', k''] = 1$ then $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. Otherwise, $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 0$.

Suppose (LHS) of the invariant is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$: $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. By definition, there is a $\mathcal{A}[\ell-1, Q', \mathcal{Q}', \mathcal{P}', k''] = 1$ for a $Q', \mathcal{Q}', \mathcal{P}'$ as described above. By induction hypothesis, (RHS) corresponding to $\mathcal{A}[\ell-1, Q', \mathcal{Q}', \mathcal{P}', k''] = 1$ holds. Thus, there is a witness forest T' in $H[\bigcup_{j=1}^{\ell-1} X_j] = H[\bigcup_{j=1}^{\ell} X_j]$ (By definition of a forget bag). By definition of $Q, \mathcal{Q}, \mathcal{P}$, T' is also a witness forest in $H[\bigcup_{j=1}^{\ell} X_j]$ and therefore (RHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$.

On the other hand, suppose (RHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$. Then there is a witness forest T' in $H[\bigcup_{j=1}^{\ell} X_j] = H[\bigcup_{j=1}^{\ell-1} X_j]$. Moreover, T' has b connected components D_1, \dots, D_b : $D_i \cap X_\ell = P_i$, $(D_i \setminus R) \cap X_\ell = Q_i$, the total number of non-terminal points in T' is k'' and $R \cap (\bigcup_{j=1}^{\ell} X_j) \subseteq V(T')$. Let $D_i \cap X_{\ell-1} = P'_i$, $(D_i \setminus R) \cap X_{\ell-1} = Q'_i$, $Q' = \bigcup_{j=1}^b Q'_j$. Note that the total number of non-terminal points in T' is k'' and by definition of a forget node it is still true that $R \cap (\bigcup_{j=1}^{\ell-1} X_j) \subseteq V(T')$. By induction hypothesis, (LHS) is true for $\mathcal{A}[\ell-1, Q', \mathcal{Q}', \mathcal{P}', k'']$ and $\mathcal{A}[\ell-1, Q', \mathcal{Q}', \mathcal{P}', k''] = 1$. By the description above, this implies that $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. Therefore, (LHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$.

Case 2: X_ℓ is an **introduce bag** with exactly one child $X_{\ell-1}$. Let \mathcal{C} be the cell being introduced in X_ℓ . Consider $\mathcal{A}[\ell, Q, \mathcal{Q} = Q_1, \dots, Q_b, \mathcal{P} = P_1 \dots P_b, k'']$. Without loss of generality, let P_b contain all the vertices in $\mathcal{C} \cap (Q \cup R)$.

13:12 Steiner Tree on Unit Disk Graphs

By definition of a state, $|\mathcal{C} \cap Q| \leq 24$. Let $\text{St} = \mathcal{C} \cap Q$ and $Q' = Q \setminus \text{St}$. Let $\mathcal{P}' = P'_1 \uplus P'_2 \dots \uplus P'_b \uplus \dots \uplus P'_d$ be a partition of $Q' \cup (R \cap X_{\ell-1})$ such that for $j < b$, $P_j = P'_j$, and $P_b = \mathcal{C} \cap (Q \cup R) \cup \bigcup_{j=b}^d P'_j$. Moreover, $\mathcal{C} \cap (Q \cup R)$ has a neighbour in each P'_j , $b \leq j \leq d$. Let \mathcal{Q}' be the partition of Q' such that $Q' \cap P'_i = Q'_i$. Let $k^* = k'' - |\text{St}|$. If $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*] = 1$ then $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. Otherwise, $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 0$. Suppose (LHS) of the invariant is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$: $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. By definition, there is a $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*] = 1$ for a $Q', \mathcal{Q}', \mathcal{P}'$ as described above. By induction hypothesis, (RHS) corresponding to $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*] = 1$ holds. Thus, there is a witness forest T' in $H[\bigcup_{j=1}^{\ell-1} X_j]$. By definition of $Q, \mathcal{Q}, \mathcal{P}$, $H[V(T') \cup (\mathcal{C} \cap (Q \cup R))]$ is a connected graph. Consider a spanning tree of this connected graph. By definition of k^* , this spanning tree has all vertices of R and exactly k'' non-terminal vertices. Therefore, this spanning tree is a witness forest in $H[\bigcup_{j=1}^{\ell} X_j]$ and therefore (RHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$.

On the other hand, suppose (RHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$. Then there is a witness forest T' in $H[\bigcup_{j=1}^{\ell} X_j]$. Moreover, T' has b connected components D_1, \dots, D_b : $D_i \cap X_{\ell} = P_i$, $(D_i \setminus R) \cap X_{\ell} = Q_i$, the total number of non-terminal points in T' is k'' and $R \cap (\bigcup_{j=1}^{\ell} X_j) \subseteq V(T')$. Without loss of generality, let D_b contain $T' \cap \mathcal{C}$. Let $D'_1, D'_2, \dots, D'_b, \dots, D'_d$ be the connected components of T' restricted to $H[\bigcup_{j=1}^{\ell-1} X_j]$. Let $D'_i \cap X_{\ell-1} = P'_i$, $(D'_i \setminus R) \cap X_{\ell-1} = Q'_i$, $Q' = \bigcup_{j=1}^d Q'_j$. Note that the total number of non-terminal points in T' is $k^* = k'' - |\text{St}|$ and by definition of an introduce node it is true that $R \cap (\bigcup_{j=1}^{\ell-1} X_j) \subseteq V(T') \cap (\bigcup_{j=1}^{\ell-1} X_j)$. By induction hypothesis, (LHS) is true for $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*]$ and $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*] = 1$. By the description above, this implies that $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k''] = 1$. Therefore, (LHS) is true for $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$.

Finally, we analyse the time complexity of the algorithm. First, the good family \mathcal{F} is computed in $n^{O(\sqrt{t+k})}$ time as per Lemma 13, and the number of instances in the good family \mathcal{F} is $n^{O(\sqrt{t+k})}$. For one such instance (H, R, t, k') the possible states for the algorithm \mathcal{A} are of the form $[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$. By definition, $\ell \leq n$, $k'' \leq k'$ and $Q = O(\sqrt{t+k})$. Again, by definition \mathcal{P} is upper bounded by the number of partitions of cells contained in a bag of the modified NCPD of (H, R, t, k') . Thus, the number of possibilities of \mathcal{P} is $\sqrt{t+k}^{O(\sqrt{t+k})}$. Also by definition, \mathcal{Q} is fixed once Q and \mathcal{P} are fixed. Therefore, the number of possible states is $n^{O(\sqrt{t+k})}$. From the description of \mathcal{A} , the computation of $\mathcal{A}[\ell, Q, \mathcal{Q}, \mathcal{P}, k'']$ may look up the solution for $n^{O(\sqrt{t+k})}$ instances of the form $\mathcal{A}[\ell - 1, Q', \mathcal{Q}', \mathcal{P}', k^*]$ and therefore takes $n^{O(\sqrt{t+k})}$ time. Thus, the total time for the dynamic programming is $O(n^{\sqrt{t+k}})$. ◀

5 FPT Algorithm for Steiner Tree on Unit Disk Graphs

In this section, we prove Theorem 2. We consider the STEINER TREE problem on unit disk graphs and design an FPT algorithm parameterized by k , which is an upper bound on the number of Steiner vertices in the solution Steiner tree. Our algorithm is based on the idea that for an instance (G, R, t, k) , in order to determine the existence of a Steiner tree we can first find spanning trees for all components of $G[R]$ and extend these spanning trees to a required k -Steiner tree.

In fact, we prove our results for the superclass of clique-grid graphs. For an instance (G, R, t, k) of STEINER TREE on clique-grid graphs, where G has n vertices and $R \subseteq V(G)$ is the set of terminals we prove the following result in this rest of this section.

► **Lemma 15.** *STEINER TREE on clique-grid graphs has an FPT algorithm with running time $2^{O(k)} n^{O(1)}$.*

First, we prove some properties of Steiner trees for unit disk graphs. Consider the induced subgraph $G[R]$. Let C_1, C_2, \dots, C_q be the connected components in $G[R]$. For each C_i , $i \in [q]$, let T_i be a spanning tree of C_i .

► **Observation 16.** *Let G be a clique-grid graph with the terminal set R . Let C_1, C_2, \dots, C_q be the connected components of $G[R]$, and for each $i \in [q]$ let T_i be a spanning tree for each C_i . For any k , let T' be a k -Steiner tree for G . Then there is a k -Steiner tree T such that for each $i \in [q]$ T_i is a subtree of T . Moreover, $q \leq 24k$.*

Proof. Consider the k -Steiner tree T and let $S = V(T) \setminus R$ be the set of Steiner vertices of T . Note that in $G[R \cup S]$, T is a spanning tree and therefore $G[R \cup S]$ is a connected graph. Similarly, for each $i \in [q]$, T_i is a subgraph of $G[R \cup S]$. Consider the subgraph $H = T' \cup \bigcup_{i \in [q]} T_i$. As T' is a spanning tree, $T' \cup \bigcup_{i \in [q]} T_i$ is a connected graph. We consider an arbitrary ordering \mathcal{O} of the edges in $E(H) \setminus (\bigcup_{i \in [q]} E(T_i))$. In this order we iteratively throw away an edge $e_j \in E(H) \setminus (\bigcup_{i \in [q]} E(T_i))$ if the resulting graph remains connected upon throwing e_j away. Let H' be the graph at the end of considering all the edges in the order \mathcal{O} . We prove that H' must be a tree. Suppose for the sake of contradiction, there is a cycle C as a subgraph of H' . As for each $i \in [q]$, T_i is a tree and for each $i \neq i' \in [q]$, $V(T_i) \cap V(T_{i'}) = \emptyset$, there must be an edge from $E(H) \setminus (\bigcup_{i \in [q]} E(T_i))$ in $E(C)$. Consider the edge $e \in (E(H) \setminus (\bigcup_{i \in [q]} E(T_i))) \cap E(C)$ with the largest index according to \mathcal{O} . This edge was throwable as $C \setminus \{e\}$ ensured any connectivity due to e . Thus, there can be no cycle in H' and it is a spanning tree of $V(H)$. This implies that $T = H'$ is a k -Steiner tree for G , S being the set of at most k Steiner vertices, such that for each $i \in [q]$, T_i is a subtree of T .

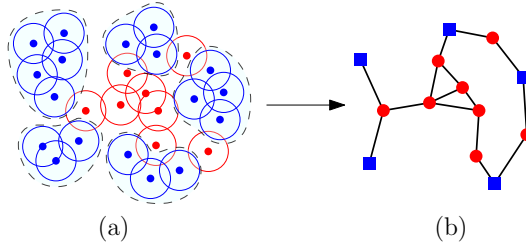
Finally, we show that if a k -Steiner tree T exists then $q \leq 24k$. Let f be a representation of the clique-grid graph G . Note that for any cell (a, b) $f^{-1}(a, b)$ is a clique. Therefore, there can be at most one component C_i intersecting with a cell (a, b) . By property (2) of Definition 5, there are at most 24 cells that can have neighbours of any vertex in (a, b) . Thus, for any Steiner vertex, there can be at most 24 components of $G[R]$ it can have neighbours in. Putting everything together, if there are at most k Steiner vertices that are used to connect the q connected components of $G[R]$ and each Steiner vertex can have neighbours in at most 24 components, then it must be that $q \leq 24k$. ◀

Henceforth, we wish to find a solution k -Steiner tree T such that for each $i \in [q]$, T_i is a subtree of T .

► **Definition 17.** *Let G be a clique-grid graph with the terminal set R . Let C_1, C_2, \dots, C_q be the connected components of $G[R]$, and for each $i \in [q]$ let T_i be a spanning tree for each C_i . Let G^* be the following graph: $V(G^*) = V(G \setminus R) \cup R^*$ where $R^* = \{c_i | i \in [q]\}$, $E(G^*) = \{v_1 v_2 | v_1, v_2 \in V(G) \setminus R\} \cup \{v c_i | v \in V(G) \setminus R, \exists u \in C_i \text{ s.t. } vu \in E(G)\}$. G^* is called the component contracted graph of G and $\{c_i | i \in [q]\}$ is the set of terminals for G^* (See Figure 4).*

Note that G^* may no longer be a clique-grid graph. From the definition of a component contracted graph and Observation 16, we have the following observation.

► **Observation 18.** *Let G be a clique-grid graph with the terminal set R . Let C_1, C_2, \dots, C_q be the connected components of $G[R]$, and for each $i \in [q]$ let T_i be a spanning tree for each C_i . Let G^* be the component contracted graph of G using the T_i 's. Then (G, R, t, k) is a yes-instance of STEINER TREE if and only if $q \leq 24k$ and (G^*, R^*, q, k) is a yes-instance of STEINER TREE.*



■ **Figure 4** An illustration of the component contraction; (a) red disks are Steiners and blue disks are terminals; (b) red vertices are Steiner vertices and blue vertices are contracted terminal components.

Now we are ready to design our FPT algorithm for STEINER TREE on clique-grid graphs parameterized by k and complete the proof of Lemma 15.

Proof of Lemma 15. Let (G, R, t, k) be an input instance of n -vertex clique-grid graphs. Let C_1, C_2, \dots, C_q be the connected components of $G[R]$, and for each $i \in [q]$ let T_i be a spanning tree for each C_i . Let G^* be the component contracted graph of G using the T_i 's. Let $R^* = \{c^i | i \in [q]\}$ be the terminal set of G^* . By, Observation 16, if G is a yes-instance then it must be that $q \leq 24k$. If this is not the case, then we immediately output no.

From now on, we are in the case $q \leq 24k$. By Observation 18, it is enough to determine whether (G^*, R^*, q, k) is a yes-instance of STEINER TREE. As noted earlier, G^* may no longer be a clique-grid graph.

We run the Dreyfus-Wagner algorithm [13] which returns a minimum edge-weighted Steiner tree connecting R^* in G^* . Since G^* is unweighted, the returned solution Steiner tree T has the minimum number of edges. Note that since G^* is unweighted, a Steiner tree for R^* minimizes the number of Steiner vertices if and only if it has minimum number of edges. The total number of Steiner vertices in T is $|V(T)| - |R^*|$. If $|V(T)| - |R^*| \leq k$, then our algorithm returns that (G^*, R^*, q, k) is a yes-instance of STEINER TREE, and otherwise it returns no.

The construction of G^* is done in polynomial time. Since $q \leq 24k$, the Dreyfus-Wagner algorithm runs in $2^{O(k)}n^{O(1)}$. Thus, our algorithm also has running time $2^{O(k)}n^{O(1)}$. ◀

6 W[1]-Hardness for Steiner Tree on Disk Graphs

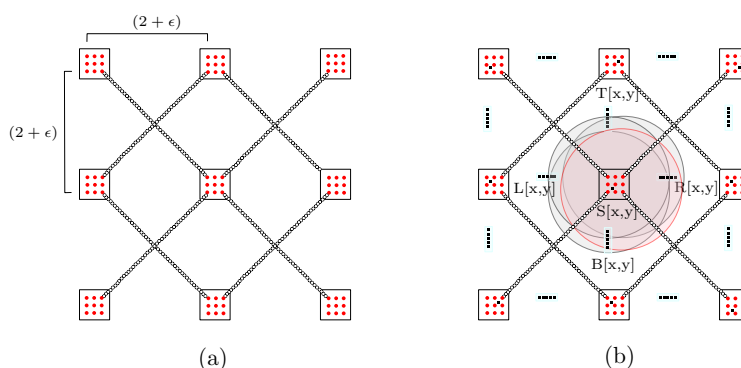
In this section, we consider the STEINER TREE problem on disk graphs and prove that this problem is W[1]-hard parameterized by the number Steiner vertices k .

► **Theorem 3.** *The STEINER TREE problem on disk graphs is W[1]-hard, parameterized by the number of Steiner vertices k .*

Proof. We prove Theorem 3 by giving a parameterized reduction from the GRID TILING WITH \geq problem which is known to be W[1]-hard³ [10]. In the GRID TILING WITH \geq problem, we are given an integer n , a $k \times k$ matrix for an integer k and a set of pairs $S_{ij} \subseteq [n] \times [n]$ of each cell. The objective is to find, for each $1 \leq i, j \leq k$, a value $s_{ij} \in S_{ij}$ such that if $s_{ij} = (a, b)$ and $s_{i+1,j} = (a', b')$ then $a \geq a'$; if $s_{ij} = (a, b)$ and $s_{i,j+1} = (a', b')$ then $b \geq b'$.

³ $k \times k$ GRID TILING WITH \geq problem is W[1]-hard, assuming ETH, cannot be solved in $f(k)n^{o(k)}$ for any function f

Let $I = (n, k, \mathcal{S})$ be an instance of the GRID TILING WITH \geq . We construct a set of unit disks D , that is divided into three sets of unit disks D_1, D_2, D_3 ; $D = D_1 \uplus D_2 \uplus D_3$. Each disk in D_1, D_2, D_3 is of radius 1, δ and κ , respectively. We will define the value of δ and κ shortly. The construction of the set $D = D_1 \uplus D_2 \uplus D_3$ will ensure that D contains a STEINER TREE with k^2 Steiner vertices if and only if I is a yes instance of GRID TILING WITH \geq . Let $\epsilon = 1/n^{10}$, and $\delta = \epsilon/4$. Here, we point out that the value of κ, ϵ are independent of each other. First, we move the cells away from each other, such that the horizontal (resp. vertical) distance between the left columns (resp. top rows) any two consecutive cell is $2 + \epsilon$. Let 100δ be the side of length of each cell. Then, we introduce diagonal chains of terminal disks into D_3 of radius $\kappa = \sqrt{2}(2 + \epsilon - 100\delta)/1000$ to connect the cells diagonally; see Figure 5(a). For every $1 \leq x, y \leq k$, and every $(a, b) \in \mathcal{S}(x, y) \subseteq [n] \times [n]$, we introduce into D_1 a disk of radius 1 centered at $(2x + \epsilon x + \epsilon a, 2y + \epsilon y + \epsilon b)$. Let $D[x, y] \subseteq D_1$ be the set of disks introduced for a fixed x and y , and notice that they mutually intersect each other. Next, for $1 \leq x, y \leq k$, we introduce into D_2 , disks of radius δ between consecutive cells of coordinate $(2x + 1 + \epsilon x + \epsilon a, 2y + \epsilon y)$ (placed horizontally); and $(2x + \epsilon x, 2y + 1 + \epsilon y + \epsilon b)$ (placed vertically). For every cell $S[x, y]$, we denote the top, bottom, left, right cluster of terminal disks of radius δ from D_2 by $L[x, y], R[x, y], T[x, y], B[x, y]$, respectively. Moreover, for each cell $S[x, y]$, we introduce a disk of radius δ at a coordinate that is completely inside the rectangle bounding the centres of disks in $D[x, y]$. This is to enforce that at least one disk is chosen from each $D[x, y]$. See Figure 5(b) for an illustration.



■ **Figure 5** (a) The schematic diagram of the cells, after adjusting the distance between adjacent cells which is $2 + \epsilon$. The red disks inside each cells, are the coordinates where the center of the Steiner disk of radius 1 will be placed. The diagonal chains consisting of terminal disks of radius κ , are connecting the cells diagonally. (b) The small black dots inside each cell are extra terminals of radius δ . Consider a cell $S[x, y]$. The shaded grey disks are the potential disks and the shaded red disk is chosen in the solution from $D[x, y]$.

We proceed with the following observation. Consider a disk p that is centered at $(2x + \epsilon x + \epsilon a, 2y + \epsilon y + \epsilon b)$ for some $(a, b) \in [n] \times [n]$. Now, consider a disk q from $R[x, y]$ centered at $(2x + 1 + \epsilon x + \epsilon a, 2y + \epsilon y)$. The distance between their centers are $\sqrt{1 + \epsilon^2 b^2}$. We need to show that this is less than $(1 + \epsilon/4)$. This is true because $1 + \epsilon^2 b^2$ is less than $(1 + \epsilon/4)^2$ as the value of b goes to n , $\epsilon = 1/n^{10}$ and the value of n is large. Hence, q is covered by the disk p from $S[x, y]$ centered at (a, b) . Next, consider a disk q' from $R[x, y]$ centered at $(2x + 1 + \epsilon x + \epsilon(a + 1), 2y + \epsilon y)$. The distance between their centers are $\sqrt{(1 + \epsilon)^2 + \epsilon^2 b^2}$. We show that this value is bigger than $(1 + \epsilon/4)$. This means $(1 + \epsilon)^2 + \epsilon^2 b^2$ is bigger than $(1 + \epsilon/4)^2$. As the value of b goes to n , it is not hard to see the left side is bigger since $\epsilon = 1/n^{10}$ and the value of n is large. Therefore, q' is not covered by the disk p from $S[x, y]$ centered at (a, b) . The same calculation holds for $L[x, y], T[x, y]$ and $B[x, y]$.

In the forward direction, let the pairs $s[x, y] \in S[x, y]$ form a solution for instance I , and let $s[x, y] = (a[x, y], b[x, y])$. For every $1 \leq x, y \leq k$, we select the disk $d[x, y]$ from D_1 of radius 1 centered at $(2x + \epsilon x + \epsilon a[x, y], 2y + \epsilon y + \epsilon b[x, y])$. We have seen in the previous paragraph that this disk covers any disk from $R[x, y]$ of center with $(2x + 1 + \epsilon x + \epsilon a[x, y], 2y + \epsilon y)$ but does not cover disks with coordinate $(2x + 1 + \epsilon x + \epsilon(a[x, y] + 1), 2y + \epsilon y)$. Similarly, this holds for $L[x, y], T[x, y], B[x, y]$. $s[x, y]$'s forms a solution of I , then we have $a[x, y] \geq a[x + 1, y]$. Therefore, the disks $d[x, y]$ and $d[x + 1, y]$ will cover all disks from $R[x, y]$. Similarly, we have $b[x, y] \geq b[x, y + 1]$ which implies that $d[x, y]$ and $d[x, y + 1]$ will cover $T[x, y]$ and form a component them. Now, the diagonals chains consisting of terminal disks of radius κ , we have taken to join the cells (see Figure 5(a)) ensures that all cells are connected. Moreover, we have shown that if $s[x, y]$'s form a solution of instance I , then all terminals in $L[x, y], R[x, y], T[x, y], B[x, y]$ (for any $1 \leq x, y \leq k$) are covered. Therefore, this will form a connected Steiner tree with k^2 many Steiner disks.

In the reverse direction, let $D' \subseteq D_1$ be a set of k^2 Steiner disks that spans over all terminals in $D_2 \cup D_3$. This is true when for every $1 \leq x, y \leq k$, the set D' contains a disk $d[x, y] \in D[x, y]$ that is centered at $(2x + \epsilon x + \epsilon a[x, y], 2y + \epsilon y + \epsilon b[x, y])$ for some $(a[x, y], b[x, y]) \in [n] \times [n]$. Indeed, we are required to choose one disk from $D[x, y]$ due to the reason that there is a terminal disk lying inside the rectangle bounding the centres of disks in $D[x, y]$. The claim is that $s[x, y] = (a[x, y], b[x, y])$'s form a solution of I . First of all, $d[x, y] \in D[x, y]$ implies that $s[x, y] \in S[x, y]$. Consider a cell $S[x, y]$. We have observed that it covers disk q from $R[x, y]$ centered at $(2x + 1 + \epsilon x + \epsilon a, 2y + \epsilon y)$, but a disk q' from $R[x, y]$ centered at $(2x + 1 + \epsilon x + \epsilon(a + 1), 2y + \epsilon y)$ is not covered. This is true for $L[x, y], T[x, y], B[x, y]$. Hence, if all terminals points from inside $S[x, y]$'s and $L[x, y], R[x, y], T[x, y], B[x, y]$ are covered by k^2 many Steiner disks, it would imply that $a[x, y] \geq a[x + 1, y]$ and $b[x, y] \geq b[x, y + 1]$. Therefore, $s[x, y]$'s form the solution for GRID TILING WITH \geq instance I . This completes the proof. ◀

Conclusion

In this paper we studied the parameterized complexity of STEINER TREE on unit disk graphs and disk graphs under the parameterizations of k and $t + k$. In future, we wish to explore tight bounds for the algorithms we have obtained and to probe into kernelization questions under these parameters. It would also be interesting to consider the minimum weight of a solution k -Steiner tree as a parameter. A variant of STEINER TREE that usually is easier to study is FULL STEINER TREE. However, in the case of unit disk graphs this problem proved to be very resilient to all our algorithmic strategies. We wish to explore FULL STEINER TREE on unit disk graphs under natural and structural parameters in future works.

References

- 1 A Karim Abu-Affash. The euclidean bottleneck full steiner tree problem. *Algorithmica*, 71(1):139–151, 2015.
- 2 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.
- 3 Piotr Berman, Marek Karpinski, and Alexander Zelikovsky. 1.25-approximation algorithm for steiner tree problem with distances 1 and 2. In *Workshop on Algorithms and Data Structures*, pages 86–97. Springer, 2009.
- 4 Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the steiner tree problem. *Journal of Algorithms*, 17(3):381–408, 1994.

- 5 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. On full steiner trees in unit disk graphs. *Computational Geometry*, 48(6):453–458, 2015.
- 6 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth, 1996.
- 7 Al Borchers and Ding-Zhu Du. Thek-steiner ratio in graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997.
- 8 Janka Chlebikova and M Chlebík. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008.
- 9 Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. In *Annals of Discrete Mathematics*, volume 48, pages 165–177. Elsevier, 1991.
- 10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.
- 11 Mark de Berg, Hans L Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C van der Zanden. A framework for eth-tight algorithms and lower bounds in geometric intersection graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 574–586, 2018.
- 12 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
- 13 Stuart E Dreyfus and Robert A Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.
- 14 Adrian Dumitrescu and János Pach. Minimum clique partition in unit disk graphs. In *Graphs and Combinatorics*, volume 27, pages 399–411. Springer, 2011.
- 15 Pavel Dvořák, Andreas Emil Feldmann, Dušan Knop, Tomáš Masařík, Tomáš Toufar, and Pavel Veselý. Parameterized approximation schemes for steiner trees with small number of steiner vertices. *arXiv preprint arXiv:1710.00668*, 2017.
- 16 Fedor V Fomin, Daniel Lokshantov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete & Computational Geometry*, 62(4):879–911, 2019.
- 17 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing System*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.
- 18 Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.
- 19 William K Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.
- 20 Mark Jones, Daniel Lokshantov, MS Ramanujan, Saket Saurabh, and Ondřej Suchý. Parameterized complexity of directed steiner tree on sparse graphs. In *European Symposium on Algorithms*, pages 671–682. Springer, 2013.
- 21 Karl Kammerlander. C 900-an advanced mobile radio telephone system with optimum frequency utilization. *IEEE journal on selected areas in communications*, 2(4):589–597, 1984.
- 22 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 23 Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.
- 24 Xianyu Li, Xiao-Hua Xu, Feng Zou, Hongwei Du, Pengjun Wan, Yuexuan Wang, and Weili Wu. A ptas for node-weighted steiner tree in unit disk graphs. In *International Conference on Combinatorial Optimization and Applications*, pages 36–48. Springer, 2009.
- 25 Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. On subexponential parameterized algorithms for steiner tree and directed subset tsp on planar graphs. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 474–484. IEEE, 2018.

13:18 Steiner Tree on Unit Disk Graphs

- 26 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/s00453-012-9630-x.
- 27 Hans Jürgen Prömel and Angelika Steger. A new approximation algorithm for the steiner tree problem with performance ratio $5/3$. *Journal of Algorithms*, 36(1):89–101, 2000.
- 28 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, 1990.
- 29 Ondřej Suchý. Extending the kernel for planar steiner tree to the number of steiner vertices. *Algorithmica*, 79(1):189–210, 2017.
- 30 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.
- 31 DW Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information processing letters*, 28(6):281–286, 1988.

Bounded-Angle Minimum Spanning Trees

Ahmad Biniiaz

School of Computer Science, University of Windsor, Canada
ahmad.biniiaz@gmail.com

Prosenjit Bose

School of Computer Science, Carleton University, Ottawa, Canada
jit@scs.carleton.ca

Anna Lubiw

Cheriton School of Computer Science, University of Waterloo, Canada
alubiw@uwaterloo.ca

Anil Maheshwari

School of Computer Science, Carleton University, Ottawa, Canada
anil@scs.carleton.ca

Abstract

Motivated by the connectivity problem in wireless networks with directional antennas, we study bounded-angle spanning trees. Let P be a set of points in the plane and let α be an angle. An α -ST of P is a spanning tree of the complete Euclidean graph on P with the property that all edges incident to each point $p \in P$ lie in a wedge of angle α centered at p . We study the following closely related problems for $\alpha = 120^\circ$ (however, our approximation ratios hold for any $\alpha \geq 120^\circ$).

1. The α -minimum spanning tree problem asks for an α -ST of minimum sum of edge lengths. Among many interesting results, Aschner and Katz (ICALP 2014) proved the NP-hardness of this problem and presented a 6-approximation algorithm. Their algorithm finds an α -ST of length at most 6 times the length of the minimum spanning tree (MST). By adopting a somewhat similar approach and using different proof techniques we improve this ratio to $16/3$.
2. To examine what is possible with non-uniform wedge angles, we define an $\bar{\alpha}$ -ST to be a spanning tree with the property that incident edges to all points lie in wedges of average angle α . We present an algorithm to find an $\bar{\alpha}$ -ST whose largest edge-length and sum of edge lengths are at most 2 and 1.5 times (respectively) those of the MST. These ratios are better than any achievable when all wedges have angle α . Our algorithm runs in linear time after computing the MST.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases bounded-angle MST, directional antenna, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.14

Funding Supported by NSERC.

Acknowledgements Matthew Katz's invited talk on bounded-angle spanning tree problems at CCCG 2018 was the inspiration for this work. We thank Therese Biedl for helpful discussions.

1 Introduction

A wireless network can be represented by disks in the plane, where a wireless node at point p with transmission range r is represented by a disk of radius r centered at point p . An edge of the network connects two nodes if each one is inside the disk centered at the other one. The question of assigning transmission ranges to the nodes to ensure a well-connected network of low interference has been widely studied [5, 6, 10, 13, 15, 16, 20, 23]. If different nodes may have different transmission ranges then we obtain “power assignment” problems,



© Ahmad Biniiaz, Prosenjit Bose, Anna Lubiw, and Anil Maheshwari;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 14; pp. 14:1–14:22

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which have also been heavily studied. The minimum transmission range to ensure network connectivity is the *bottleneck* of the minimum Bottleneck Spanning Tree (BST) – equivalently, the maximum edge-length in a Minimum Spanning Tree (MST).

In recent years, the idea of replacing omni-directional antennas with *directional antennas* has received considerable attention (see, e.g., [1, 4, 8, 11, 12, 14, 15, 24]). In this model, the full disk at each point p is restricted to a circular wedge with apex p that has some angle α and is oriented in some direction. Directional antennas are desirable in many ways, for example, they require lower transmission power, cause less interference, and provide more secure communication (see [5, 24] and references therein). The *symmetric communication network* [5] has an edge between two nodes if each one is inside the other’s wedge.

When every node has the same transmission range r and angle α , there is still freedom to orient the directional antennas. The question of whether r and α permit a connected network is NP-hard (see Further Background below for details). Most previous work has concentrated on the case where α is some fixed value. Aschner and Katz [4] formulated this in terms of an α -Spanning Tree (α -ST): a spanning tree of the complete Euclidean graph on a point set P in the plane such that for each point $p \in P$ all the edges incident to p lie in a wedge of angle α centered at p (see Figure 1-left). For any $\alpha < \pi/3$, an α -ST may not exist, for example, if P is the set of vertices of an equilateral triangle. However, for any $\alpha \geq \pi/3$, an α -ST always exists (see [1], [2], and [12] for three different and somewhat involved proofs). There is a relationship between α -STs and the well-studied concept of restricted degree spanning trees, since d edges at a vertex always lie in some wedge of angle at most $2\pi(1 - 1/d)$.



■ **Figure 1** Left: A 120° -ST ($\frac{2\pi}{3}$ -ST). Right: a degree-5 minimum spanning tree which is a $\frac{8\pi}{5}$ -ST.

To evaluate edge lengths of α -STs two concepts are useful. An α -Bottleneck Spanning Tree (α -BST) is an α -ST that minimizes the maximum edge length, and an α -Minimum Spanning Tree (α -MST) is an α -ST that minimizes the sum of the edge lengths. Both are NP-hard to compute. Although any MST is also a BST [10], this is not necessarily true for α -MST and α -BST – we give an example later on. For $\alpha \geq 120^\circ$ Aschner and Katz [4, 22] gave a simple polynomial time 5-approximation algorithm for the α -BST.

The main result of Aschner and Katz [4] is an involved 6-approximation algorithm for the α -MST for $\alpha \geq 120^\circ$. In particular, they construct an α -ST of length at most 6 times the length of an MST of the points. Our main technical result is to improve this approximation factor to $16/3$. An interesting aspect of their proof is that they orient the wedges at small subsets of the points so that the network on these points is connected *and* the wedges cover the whole plane. The problem of orienting wedges at fixed points to cover (or “light up”) the plane is called the “Floodlight Problem”. Bose et al. [9] showed that for any n points and any set $\{\alpha_1, \dots, \alpha_n\}$ of angles that are at most π and sum to 2π , there is a way to assign angles to points and orient the wedges to light up the plane. To obtain our main result, we prove a strengthened version of the Floodlight result for $n = 3$ where we obtain a connected symmetric communication network even if the angles are pre-assigned to the points.

The new problem we study in this paper is the directional antenna problem when different nodes are allowed to have different wedge angles. The goal is to minimize the sum of the angles. For consistency with previous notation, we define an $\bar{\alpha}$ -ST to be a spanning tree of the point set P such that for each point $p \in P$ all the edges incident to p lie in a wedge of angle α_p centered at p and the average of the α_p 's is α , or in other words, $\sum_p \alpha_p = n\alpha$. One might hope that, as with the Floodlight Problem, a constant angle sum would suffice to construct a good network. Indeed, taking a star as a spanning tree achieves constant angle sum (with π at an extreme point taken as the center, and 0 at the leaves of the star) – but this uses a large transmission range. We show that it is not possible in general to achieve constant angle sum and transmission ranges bounded by a constant times the BST bottleneck.

In the positive direction, we show that allocating an angle sum of $n \cdot 120^\circ$ non-uniformly does help, in particular, we can achieve smaller maximum edge-length and sum of edge lengths (compared to the MST) than is possible with uniform angles. Now we give more details about our results and techniques, and more details on background.

1.1 Our Results

We obtain the following results for angle $\alpha = 120^\circ$, however, our approximation ratios hold for any angle $\alpha \geq 120^\circ$.

(1) Aschner and Katz [4] gave an elegant algorithm that finds a 120° -ST of length at most 6 times the MST length, thus providing a 6-approximation algorithm for the 120° -MST. Our main technical result (Theorem 5 in Section 3) is to improve the approximation ratio to $16/3$.

Aschner and Katz prove their result using (in their words) a “surprising theorem”, proved via a long case-analysis, that there is a way to assign 120° wedges to any 3 points (a “triplet”) such that: the union of the 3 wedges covers the plane; the resulting graph on the triplet is connected; and the resulting graph on any pair of triplets (whose wedges are assigned independently) is also connected. After that, their approach is to take an MST of the points, double its edges and take short-cuts to obtain a Hamiltonian path of length at most 2 times the MST length (as in the standard TSP approximation), and then apply their result to successive triplets of points on the path. Their theorem guarantees that successive pairs of triplets are connected. Finally, they get an approximation factor of 6 using the triangle inequality, and a judicious choice of whether to start the partition into triplets at the first, second, or third point of the path. One limitation of their approach is that the two edges used in each triplet may be the longest and second longest edges of the triplet.

We follow the same approach. We take a non-crossing Hamiltonian path and consider successive triplets of points along the path. The fact that edges of the Hamiltonian path are non-crossing ensures that the triplets are also “non-crossing” in some sense. This allows us to assign wedges to the triplets in a different way which leads to a shorter proof of the “surprising theorem” for non-crossing triplets, as well as a better length guarantee within each triplet.

Our algorithm is slower than that of Aschner and Katz because of the extra work of uncrossing the edges of the Hamiltonian path.

(2) We give an algorithm to find a $\overline{120^\circ}$ -ST whose largest edge-length is at most 2 times that of the MST, and whose sum of edge lengths is at most 1.5 times that of the MST (Theorem 11 in Section 4). The idea of our algorithm is to start with an MST that has maximum degree 5 (which is known to exist) and then re-assign parts of the wedge angles from leaves to inner vertices. Our algorithm runs in linear time after computation of the MST. The ratios 2 and 1.5 improve the best known ratios for 120° -BST and 120° -MST (5 and $16/3$, respectively). In

fact, our ratios for non-uniform angles are better than any possible with uniform angles, as we prove by designing an infinite class of point sets such that every 120° -ST has maximum edge length at least 3 times that of the MST, and sum of edge lengths at least 2 times that of the MST.

(3) We present the following lower bounds for approximating the above problems with respect to the MST. These lower bounds are proved in Section 2. Although the lower bounds 2 and 3 (in Proposition 1) for the 120° -MST and the 120° -BST seem to be common knowledge [4, 22], for the sake of completeness we provide a proof of them.

► **Proposition 1.** *The 120° -MST, the $\overline{120^\circ}$ -MST, and the 120° -BST problems cannot be approximated by ratios smaller than 2, $4/3$, and 3, respectively, given the MST length and the MST largest edge-length as lower bounds.*

► **Proposition 2.** *Let $A(n)$ be the smallest value that suffices to construct, for any set of n points in the plane, a connected symmetric network with angle sum $A(n)$ and with transmission ranges bounded by a constant times the BST largest edge-length. Then $A(n) = \Omega(\sqrt{n})$.*

► **Proposition 3.** *For any $\alpha < \pi$ there exists a point set for which no α -MST is an α -BST.*

1.2 Further Background

As mentioned above, there is a connection between α -STs and restricted degree spanning trees, due to the fact that d edges at a vertex always lie in some wedge of angle at most $2\pi(1 - 1/d)$. The Minimum degree- k spanning tree (degree- k MST) problem has been well-studied (see, e.g., [3, 13, 21, 23]). It is easy to compute a degree-2 spanning tree (a Hamiltonian path) of length at most twice the length of the Euclidean MST by doubling the MST edges, taking an Euler tour, short-cutting repeated vertices, and then removing an edge. It is also possible to compute in polynomial time degree-3, degree-4, and degree-5 spanning trees of lengths at most 1.402 [13], 1.1381 [13, 21], and 1 [25] times the MST length, respectively. This immediately implies the existence of 180° -ST, 240° -ST, 270° -ST, and 288° -ST ($\frac{8\pi}{5}$ -ST) of lengths at most 2, 1.402, 1.1381, and 1 times the MST length, respectively. See Figure 1-right for an illustration of a degree-5 MST which is a 288° -ST.

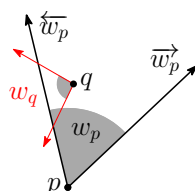
The α -MST problem is also related to the problem of computing angle-restricted Hamiltonian paths and cycles on points in the plane. One can compute a Hamiltonian path with angles of at most 90° by starting from an arbitrary point and iteratively connecting the current point to its farthest among the remaining points (see [19]); the angle 90° is tight in the sense that there are point sets for which every Hamiltonian path has an angle larger than $90^\circ - \varepsilon$ for any $\varepsilon > 0$ (see [12, 17]). Fekete and Woeginger [19] conjectured that for any even-size point set of at least 8 elements there exists a Hamiltonian cycle with angles at most 90° . Dumitrescu et al. [17] gave a partial solution by constructing a cycle with angles of at most 120° . The conjecture remains open.

Aschner and Katz [4] studied the α -MST problem for $\alpha \in \{90^\circ, 120^\circ, 180^\circ\}$. They proved the NP-hardness of this problem for $\alpha = 120^\circ$ and $\alpha = 180^\circ$ by reductions from the problem of finding a Hamiltonian path in hexagonal grid graphs and in square grid graphs of degree at most three, respectively. In addition to the result mentioned above for a 120° -ST of length at most 6 times the MST length, they also presented an algorithm for computing a 90° -ST of length at most 16 times the MST length.

The problem of constructing bounded-angle networks with no long edges (the α -BST problem) has been studied extensively. The NP-hardness reduction of Aschner and Katz for the 120° -MST problem also implies the NP-hardness of the 120° -BST problem and its

inapproximability by a factor smaller than $\sqrt{3}$. Carmi et al. [12] construct 90° -Hamiltonian paths with edges that are shorter than that of the construction by Fekete and Woeginger [19]. Aschner and Katz [4] construct 120° -hop-spanners with hop-ratio 6 and edge lengths at most 7 for unit disk graphs. Dobrev et al. [15, 16] and Caragiannis et al. [11] construct strongly connected directed networks with short edges and nodes of bounded out-degree.

1.3 Notation and Preliminaries



■ **Figure 2** The point p sees q but q does not see p .

Let w_p be a wedge in the plane with apex p . We denote by $\overrightarrow{w_p}$ the clockwise (right) boundary ray of w_p , and by $\overleftarrow{w_p}$ the counterclockwise (left) boundary ray of w_p . Let w_q be another convex wedge in the plane with apex q . If q lies in w_p then we say that p sees q and denote this by $p \rightarrow q$. We use $p \leftrightarrow q$ to denote that p and q are *mutually visible*, that is, p and q see each other. In other words, $p \leftrightarrow q$ denotes $p \rightarrow q$ and $q \rightarrow p$. In Figure 2 the point p sees q but q does not see p , and thus they are not mutually visible. Let P be a set of points in the plane and assume that wedges, possibly of different angles, are placed at every point of P . Then the *induced mutual-visibility graph* on P is a geometric graph with vertex set P that has a straight-line edge between two points p and q if and only if p and q are mutually visible. The underlying non-geometric graph is the *symmetric communication network* on P .

We denote the sum of edge lengths of a geometric graph G by $w(G)$. We need the following basic fact about triangles in the plane.

► **Lemma 4.** *Let a , b , and c be three points in the plane, and let $E = \{ab, ac, bc\}$ be the set of edges between them. Then the total length of the shortest and longest edges in E is at most 1.5 times the total length of any two edges of E .*

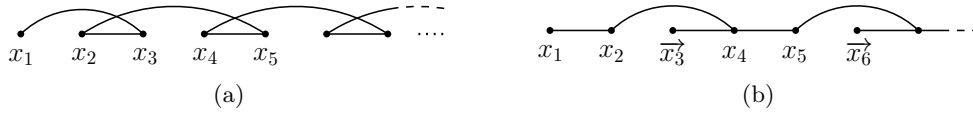
Proof. After a suitable relabeling we may assume that $|ab| \leq |ac| \leq |bc|$. To prove the lemma it suffices to show that (i) $|ab| + |bc| \leq 1.5(|ac| + |bc|)$ and (ii) $|ab| + |bc| \leq 1.5(|ab| + |ac|)$. Statement (i) holds because $|ab| \leq |ac|$. To verify statement (ii) observe that by the triangle inequality we have $|bc| \leq (|ab| + |ac|)$, and by our assumption that $|ab|$ is not larger than $|ac|$ we have $|ab| \leq 0.5(|ab| + |ac|)$. The sum of the two inequalities implies statement (ii). ◀

2 Lower bounds

In this section we prove Propositions 1, 2, and 3. First we prove Proposition 1 that is: *The 120° -MST, the 120° -MST, and the 120° -BST problems cannot be approximated by ratios smaller than 2, $4/3$, and 3, respectively, given the MST length and the MST largest edge-length as lower bounds.*

Proof. Consider a sequence $X = (x_1, x_2, \dots, x_n)$ of n points on the x -axis with coordinates $1, 2, \dots, n$, respectively, as in Figure 3. We show that X satisfies the statement of the proposition for the three problems. The MST of X is a path with edges of length 1 and total length $n - 1$.

14:6 Bounded-Angle Minimum Spanning Trees



■ **Figure 3** (a) A 120° -ST of length $2n - 3$, and (b) a $\overline{120^\circ}$ -ST of length $4n/3 - 3$.

First we prove the lower bound 2 for the 120° -MST problem (this lower bound is also mentioned in [4]). We show by induction that any 120° -minimum spanning tree T on X has length at least $2n - 3$. This is trivial if $n \in \{1, 2\}$, thus assume that $n \geq 3$. There are $n - 1$ intervals on the x -axis between consecutive points of X . If every interval is covered by two edges of T then $w(T) \geq 2n - 2$. Assume that some interval $[i, i + 1]$ is covered by only one edge of T , say edge e . By removing e we obtain two subtrees T_1 and T_2 where T_1 spans the points x_1, \dots, x_i while T_2 spans x_{i+1}, \dots, x_n . Let n_1 and n_2 be the number vertices of T_1 and T_2 , respectively. Assume that both n_1 and n_2 are at least 2. Since T_1 spans x_1, \dots, x_i , the vertex x_i is oriented to the left (and does not see any point to its right). Analogously, x_{i+1} is oriented to the right. Therefore e is not incident to x_i or x_{i+1} , and thus its length is at least 3. By the induction hypothesis we get $w(T) = w(T_1) + w(T_2) + w(e) \geq (2n_1 - 3) + (2n_2 - 3) + 3 = 2(n_1 + n_2) - 3 = 2n - 3$. Now assume that $n_1 = 1$, and thus $n_2 \geq 2$. Then T_1 has only vertex $x_i = x_1$ which is an endpoint of e . Since T_2 spans x_2, \dots, x_n , the vertex x_2 is oriented to the right. Therefore e is not incident to x_2 , and thus its length is at least 2. Thus $w(T) = w(T_1) + w(T_2) + w(e) \geq 0 + (2n_2 - 3) + 2 = 2n - 3$. The case where $n_2 = 1$ is handled similarly.

To verify the lower bound 3 for the 120° -BST problem, assume that $n = 5$. Consider the edge-maximal graph on X that has edges of length at most 2. In any spanning tree in this graph, at least one of x_2, x_3, x_4 has incident edges in both directions (left and right). Thus, no matter how we place wedges of angle 120° on vertices of X , we cannot get a 120° -ST of edge lengths at most 2. Therefore, any 120° -ST on X has an edge of length at least 3, as in Figure 3(a). This argument can be generalized for any n larger than 5.

Now consider any $\overline{120^\circ}$ -minimum spanning tree \overline{T} on X . We show that $w(\overline{T}) \geq 4n/3 - 3$. Partition the vertices of \overline{T} into X_1 and X_2 where X_1 is the set of vertices with wedges of angle strictly less than 180° and X_2 is the set of vertices with wedges of angle at least 180° . Since the total available angle is $120n$ degrees, $|X_2| \leq 120n/180 = 2n/3$. Thus $|X_1| = n - |X_2| \geq n/3$. Observe that every interval (between consecutive vertices of X) is covered by an edge of \overline{T} . Every vertex $x_i \in X_1$ sees the vertices that are either to its left or to its right. We denote x_i by \overleftarrow{x}_i if it sees the vertices to its left, and by \overrightarrow{x}_i otherwise (see Figure 3(b)). For every \overleftarrow{x}_i the interval $[i - 1, i]$ is covered by at least two edges otherwise connectivity is lost: one edge is incident to \overleftarrow{x}_i and another edge connects a point to the left of \overleftarrow{x}_i with a point to the right (assuming $i \neq n$). Similarly for every \overrightarrow{x}_i the interval $[i, i + 1]$ is covered by an edge that is incident to \overrightarrow{x}_i and by an edge that connects a point to the right of \overrightarrow{x}_i with a point to the left (assuming $i \neq 1$), as in Figure 3(b). Thus, for every \overleftarrow{x}_i (except possibly \overleftarrow{x}_n) there exists a unique interval that is covered by two edges of \overline{T} . Similarly, for every \overrightarrow{x}_i (except possibly \overrightarrow{x}_1) there exists a unique interval that is covered by two edges of \overline{T} . (If x_i is oriented to the left and x_{i+1} is oriented to the right then – by the minimality of the tree – (x_i, x_{i+1}) is an edge of \overline{T} and the interval $[i, i + 1]$ is covered by three edges.) Therefore the length of \overline{T} is at least $(n - 1) + (|X_1| - 2) \geq 4n/3 - 3$. ◀

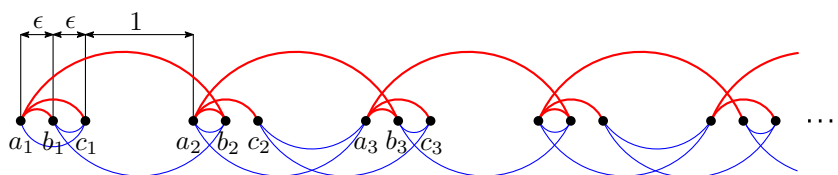
Now we prove Proposition 2 that is: *Let $A(n)$ be the smallest value suffices to construct, for any set of n points in the plane, a connected symmetric network with angle sum $A(n)$ and with transmission ranges bounded by a constant times the BST largest edge-length. Then $A(n) = \Omega(\sqrt{n})$.*

Proof. Consider a set of n points on the vertices of a regular $\sqrt{n} \times \sqrt{n}$ grid of side length $\sqrt{n}-1$. The largest edge-length of any BST for this point set is 1. Now consider any connected symmetric network T for this point set that satisfies the statement of the proposition, and let the constant c be the largest transmission range (edge-length) in T . Let L denote the set of supporting lines for all edges of T . Since any line in the plane contains at most \sqrt{n} points of the grid, L has at least \sqrt{n} lines. Therefore, T has at least one vertex, say v , where the neighbors of v lie on more than one line of L . Let V be the set of all such vertices of T . Observe that every line in L passes through a vertex in V . For every $v \in V$ the neighbors of v lie in a square of side-length $2c$ that is centered at v . Thus v has at most $(2c)^2$ neighbors (that can lie on at most $(2c)^2$ lines in L). Therefore $|V| \geq |L|/(2c)^2 \geq \sqrt{n}/(2c)^2$.

Now consider any $v \in V$. Let v_1 and v_2 be two neighbors of v such that edges $(v, v_1) \in T$ and $(v, v_2) \in T$ lie on two different lines of L . Since v, v_1 , and v_2 are grid points and lie on a square of constant side-length $2c$, the angle between v_1 and v_2 at v is a constant number. Let α be the smallest such constant over all vertices in V . Then, the total angle at vertices in V is at least $\alpha \cdot |V| \geq \alpha\sqrt{n}/(2c)^2 = \Omega(\sqrt{n})$. ◀

The following is a proof of Proposition 3 that is: *For any $\alpha < \pi$ there exists a point set for which no α -MST is an α -BST.*

Proof. Consider the point set P in Figure 4 consisting of $n \geq 9$ points partitioned into $t = n/3$ triplets (a_i, b_i, c_i) where $|c_i a_{i+1}| = 1$ and $|a_i b_i| = |b_i c_i| = \epsilon$ for some $\epsilon < 1/(2n-3)$. We refer to an edge of length at least 1 as a *long edge*. The red bold tree is an α -ST of total length $(t-1)((1+3\epsilon) + 2\epsilon + \epsilon) + 2\epsilon + \epsilon = (t-1) + (6t-3)\epsilon < t$, where the last inequality holds by our choice of ϵ . Therefore any α -MST for P has at most $t-1$ long edges because otherwise it would have a length larger than t . This implies that each interval $[c_i, a_{i+1}]$ is covered by at most 1 edge of any α -MST. On the other hand, any spanning tree for P has at least $t-1$ long edges as all triplet should be connected to the rest of the tree by a long tree edge. It turns out that any α -MST T of P has exactly $t-1$ long edges each connecting a point in triplet (a_i, b_i, c_i) to a point in triplet $(a_{i+1}, b_{i+1}, c_{i+1})$. In the rest of the proof we show that T has an edge of length at least $1+3\epsilon$. This immediately proves our claim because P admits an α -ST of edge lengths at most $1+2\epsilon$, see for example the thin blue tree in Figure 4.



■ **Figure 4** The red bold spanning tree has a total length of less than t . The blue thin spanning tree has edges of lengths at most $1 + 2\epsilon$.

Consider any $i \in \{1, \dots, t-1\}$. Let e_i be the long edge of T between triplets (a_i, b_i, c_i) and $(a_{i+1}, b_{i+1}, c_{i+1})$. The point c_i cannot be an endpoint of e_i because otherwise there must be a long edge e'_i in T that connects a point to the left of c_i to a point to the right of c_i ; this contradicts the fact that the interval $[c_i, a_{i+1}]$ is covered by exactly one edge of T (by the degree constraint c_i cannot be connected to any point to the left). By symmetry, a_{i+1} cannot be an endpoint of e_i either. If a_i or c_{i+1} is an endpoint of e_i then $|e_i| \geq 1 + 3\epsilon$ and we are done. Assume that for every $i \in \{1, \dots, t-1\}$ we have $e_i = (b_i, b_{i+1})$. Then (b_1, b_2) and (b_2, b_3) are edge of T . Thus the angle at b_2 is larger than α which contradicts T being an α -ST. ◀

3 Approximating the 120° -MST

Let $\alpha = 120^\circ$ in this section. Let P be a set of points in the plane. Aschner and Katz [4] showed a construction of an α -ST on P of length at most 6 times the MST length. In this section we present an alternate construction that achieves an α -ST of length at most $16/3$ times the MST length, thereby proving the following theorem.

► **Theorem 5.** *Given a set of points in the plane and an angle $\alpha \geq 120^\circ$, there is an α -spanning tree of length at most $16/3$ times the length of the MST. Furthermore, there is a polynomial time algorithm to find such an α -ST, thus providing a $16/3$ -approximation algorithm for the α -MST problem.*

3.1 The construction of Aschner and Katz

To facilitate comparisons, we briefly describe the algorithm of Aschner and Katz [4]. They use the following theorems to compute an α -ST.

► **Theorem 6** (Aschner and Katz, 2014). *Given a set P of three points in the plane, one can place at each point of P a wedge of angle 120° such that the three wedges cover the plane and the induced mutual-visibility graph on P is connected, and hence it contains an 120° -ST.*

A placement that satisfies the conditions of Theorem 6 is given in the proof of Claim 2.1 [4]. This placement has an interesting property which is given in the following theorem whose proof is very involved.

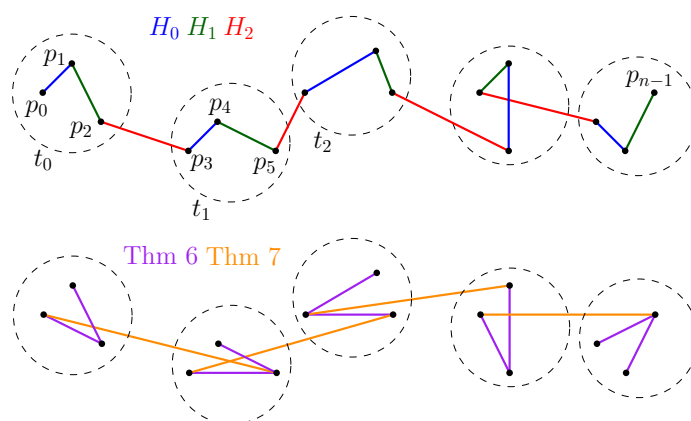
► **Theorem 7** (Aschner and Katz, 2014). *Let P_1 and P_2 be two disjoint sets each containing three points in the plane. Assume that a wedge of angle 120° is placed at each point of P_1 and at each point of P_2 according to the placement of Theorem 6. Then, the induced mutual-visibility graph on $P_1 \cup P_2$ is connected, and hence it contains a 120° -ST.*

Let H be a Hamiltonian path on P of length at most 2 times the MST length. The constant 2 is tight, as Fekete et al. [18] showed that for any fixed $\varepsilon > 0$ there exists a point set for which any Hamiltonian path has length at least $2 - \varepsilon$ times the MST length.

Let (p_0, \dots, p_{n-1}) be the sequence of points of P from one endpoint of H to the other. Let $(h_0, h_1, \dots, h_{n-2})$ be the sequence of the edges of H where $h_i = (p_i, p_{i+1})$. Partition the edges of H into three sets $H_0 = \{h_0, h_3, h_6, \dots\}$, $H_1 = \{h_1, h_4, h_7, \dots\}$, and $H_2 = \{h_2, h_5, h_8, \dots\}$, as in Figure 5. The length of one of these sets, say H_2 , is at least $w(H)/3$. Therefore $w(H_0) + w(H_1) \leq \frac{2}{3}w(H)$. Partition P into a sequence of triplets $(p_0, p_1, p_2), (p_3, p_4, p_5), \dots$ such that the edges of H , that lie between consecutive triplets, are in H_2 . Then place three wedges on the points of each triplet according to Theorem 6, and let G_α be the induced mutual-visibility graph. The points in each triplet are connected (by Theorem 6) and any pair of triplets are connected (by Theorem 7), and thus G_α is connected.

Now let T be a spanning tree of G_α computed as follows (see Figure 5): between the points in each triplet take two edges that are obtained from Theorem 6 (these edges are called *inner edges*), and between every two consecutive triplets take an edge that is obtained by Theorem 7 (these edges are called *connecting edges*). To bound the length of T , we charge edges of H for the edges of T . Every edge of H that belongs to H_2 lies between two consecutive triplets, and thus is charged only once for the connecting edge between the two triplets. Every edge of H that belongs to $H_0 \cup H_1$ lies inside a triplet, say t . Such an edge is charged 4 times: twice for the two inner edges of t and twice for the two edges that connect t to adjacent triplets. Therefore,

$$w(T) \leq w(H_2) + 4(w(H_0) + w(H_1)) = w(H) + 3(w(H_0) + w(H_1)) \leq 3w(H) \leq 6w(\text{MST}).$$



■ **Figure 5** Top: path H where H_0, H_1, H_2 are colored blue, green, and red, respectively. Bottom: spanning tree T where edges obtained by Theorems 6 and 7 are colored purple and orange, respectively.

3.2 The new construction

Our approach, for the construction of an α -ST of length at most $16/3$ times the MST length, is similar to that of Aschner and Katz, however we use a different placement of wedges on points of triplets which in turn requires different proof techniques.

Let $P = \{p_1, p_2, p_3\}$ be any set of three points in the plane. The complete graph on P has edge set $E = \{p_1p_2, p_1p_3, p_2p_3\}$ and contains exactly three spanning trees. Let T_1 be the tree that has the two shortest edges of E , T_3 be the tree that has the two longest edges of E , and T_2 be the tree that has the shortest and the longest edges of E . Observe that $w(T_1) \leq w(T_2) \leq w(T_3)$. We refer to T_1, T_2 , and T_3 as the *shortest, intermediate, and longest* trees on P , respectively.

The α -ST obtained by the wedge placement in (the proof of) Theorem 6 contains the two longest edges of E . In other words, this placement gives the spanning tree T_3 . Due to the objective of minimizing the sum of edge lengths, one may ask for a wedge placement that covers the plane using T_1 or T_2 . We answer this question in the affirmative in Theorem 8. This theorem improves Theorem 6 in the sense that it uses T_1 or T_2 and also allows more flexibility on the angles of the wedges.

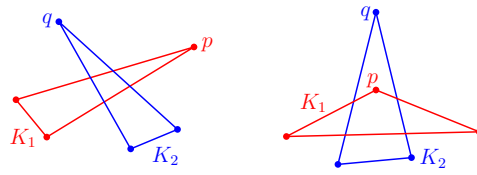
Consider three points in the plane, and three angles $\alpha_1, \alpha_2, \alpha_3$ where each α_i is at most π and $\alpha_1 + \alpha_2 + \alpha_3 = 2\pi$. The Floodlight result of Bose et al. [9] implies that one can cover the plane by placing three wedges of angles $\alpha_1, \alpha_2, \alpha_3$ at the three points. Our Theorem 8 also improves this result in two ways: (i) one can cover the plane with three wedges even if the assignment of angles to points is specified in advance, and (ii) the induced mutual-visibility graph is connected, in particular it contains the shortest or the intermediate tree on points.

► **Theorem 8** (proved in Section 3.3). *Given a set $P = \{p_1, p_2, p_3\}$ of three points in the plane and three angles $\alpha_1, \alpha_2, \alpha_3 \leq \pi$ where $\alpha_1 + \alpha_2 + \alpha_3 = 2\pi$, one can place at each p_i a wedge of angle α_i such that the three wedges cover the plane and the induced mutual-visibility graph contains the shortest or the intermediate tree on P .*

If we set each α_i equal to 120° in Theorem 8, then we get the following corollary.

► **Corollary 9.** *Given a set P of three points in the plane, one can place wedges of angle 120° at points of P such that the three wedges cover the plane and the induced mutual-visibility graph contains the shortest or the intermediate tree on P .*

14:10 Bounded-Angle Minimum Spanning Trees



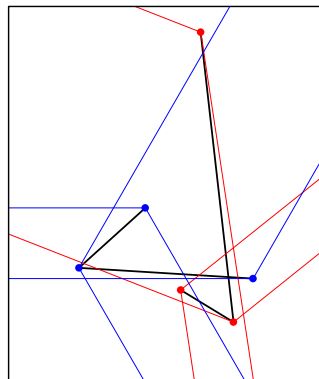
■ **Figure 6** Two configurations of a forbidden pair $\{P_1, P_2\}$ forming triangles K_1 and K_2 .

Corollary 9 ensures covering of the plane and also uses T_1 or T_2 . Unfortunately, we cannot directly use this corollary in the approach of Aschner and Katz because their Theorem 7 ensures a connection between two triplets t_1 and t_2 only if they are oriented by Theorem 6; such a connection may not exist if the triplets are oriented by Corollary 9. However, we show in Theorem 10 that if the relative positions of the points of t_1 and t_2 is not among the two configurations shown in Figure 6, then the orientation of Corollary 9 ensures a connection between them. Our final proof relies on the fact that the configurations in Figure 6 never arise from triplets of a non-crossing Hamiltonian path.

To be more precise, consider disjoint point sets P_1 and P_2 , each of size three, in the plane. Let K_1 and K_2 be the complete graphs (triangles) on points of P_1 and P_2 , respectively. We say that the (unordered) pair $\{P_1, P_2\}$ is *forbidden* if there exist vertices $p \in K_1$ and $q \in K_2$ such that (i) both incident edges of p intersect both incident edges of q , or (ii) p lies inside the triangle K_2 and the non-incident edge of p intersects both incident edges of q . See Figure 6.

► **Theorem 10** (proved in Appendix A). *Let P_1 and P_2 be two disjoint sets, each containing three points in the plane, such that $\{P_1, P_2\}$ is not forbidden. Assume that a wedge of angle 120° is placed at each point of P_1 and at each point of P_2 according to the placement algorithm of Corollary 9. Then, the induced mutual-visibility graph on $P_1 \cup P_2$ is connected, and hence it contains a 120° -ST.*

► **Remark 1.** Intuitively, it would seem that our proof of Theorem 10 should follow the same approach as that of Aschner and Katz’s Theorem 7. We note, however, that their proof of Theorem 7 is highly involved and uses a combination of nontrivial ideas. In order to cope with the large number of cases, they classify the number of edges in the induced mutual-visibility graph of each triplet. Their proof ensures the existence of an edge between P_1 and P_2 only if the wedges are oriented according to the placement in the proof of Theorem 6. Such an edge may not exist for other wedge placements with similar properties, i.e., coverage of the plane



■ **Figure 7** There is no connection between the red and blue triplets.

and connectivity of each set P_1 and P_2 ; for example see Figure 7 which is borrowed from [4] (notice that the pair of triplets in this figure is forbidden). Thus, there is no straightforward way of adjusting their proof to work for our Theorem 10 where the wedges in P_1 and P_2 are oriented according to Corollary 9 instead. We give a relatively short proof of Theorem 10 in Appendix 3.3. Instead of classifying the number of edges of visibility graphs, we introduce a “representative” point for each triplet; this facilitates a shorter presentation of our proof.

The remaining ingredient we need in order to apply Theorem 10 is the following observation that forbidden triplets do not arise if we begin with a Hamiltonian path H that is non-crossing.

► **Observation 1.** *Let H be a non-crossing Hamiltonian path and let t_1 and t_2 be disjoint triplets, each of which is obtained by consecutive vertices of H . Then, $\{t_1, t_2\}$ is not forbidden.*

Proof. Each of t_1 and t_2 must contain two edges of H (because they come from consecutive vertices of H), and the union of these edges is non-crossing because H is non-crossing. But if $\{t_1, t_2\}$ is forbidden (see Figure 6), then there is no way to choose two edges in each of t_1 and t_2 such that their union is non-crossing. ◀

To use this observation, we need a non-crossing Hamiltonian path. Such a path can be obtained by iteratively flipping crossing edges of H . It is known that this iterative process terminates after $O(n^3)$ edge flips [26], where n is the number of path vertices. (See [7] for some recent results on obtaining non-crossing configurations by edge flips.) Since the edge flip operation does not increase the total edge length (this can be verified by the triangle inequality), the length of the resulting non-crossing path is not more than that of the original path. Therefore, we assume from now on that H is non-crossing and $w(H) \leq 2w(MST)$.

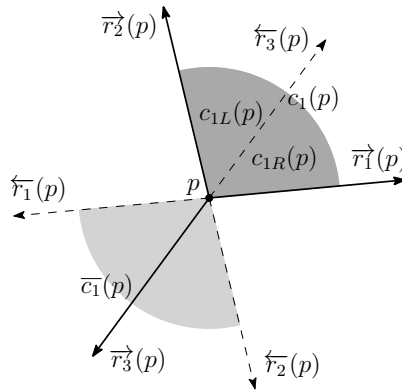
Now we have the necessary tools (a non-crossing path H and Theorem 10) to use the orientation of the wedges as in Corollary 9. Let (p_0, \dots, p_{n-1}) be the sequence of points from one endpoint of H to the other. Construct edge sets H_0, H_1, H_2 , and as before assume that $w(H_2) \geq w(H)/3$, which implies $w(H_0) + w(H_1) \leq \frac{2}{3}w(H)$. Construct triplets $t_0 = (p_0, p_1, p_2), t_1 = (p_3, p_4, p_5), \dots$, and orient the wedges according to Corollary 9. Since H is non-crossing, no pair of triplets is forbidden (by Observation 1). The induced mutual-visibility graph, G_α , is connected because the points of each triplet are connected (by Corollary 9) and every two triplets are connected (by Theorem 10). We obtain a spanning tree T from G_α as before. Every edge of H_2 is charged only once for the connecting edge between two consecutive triplets. Every pair (h_i, h_{i+1}) of edges in each triplet t (where $h_i \in H_0$ and $h_{i+1} \in H_1$) is charged 3.5 times: twice for connecting edges between t and its adjacent triplets, and 1.5 times for inner edges of t (by Lemma 4 the length of the tree obtained by Corollary 9 is at most $1.5(w(h_i) + w(h_{i+1}))$). Therefore

$$\begin{aligned} w(T) &\leq w(H_2) + 3.5(w(H_0) + w(H_1)) = w(H) + 2.5(w(H_0) + w(H_1)) \\ &\leq w(H) + \frac{5}{3}w(H) = \frac{8}{3}w(H) \leq \frac{16}{3}w(MST). \end{aligned}$$

This finishes the proof of Theorem 5.

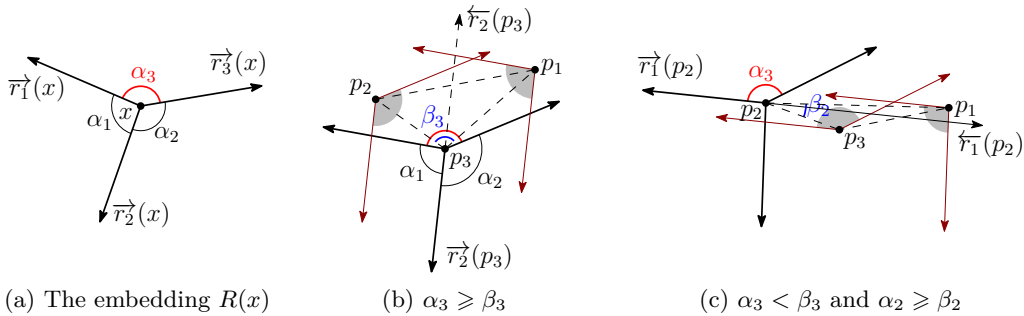
3.3 Proof of Theorem 8

In this section we prove Theorem 8: *Given a set $P = \{p_1, p_2, p_3\}$ of three points in the plane and three angles $\alpha_1, \alpha_2, \alpha_3 \leq \pi$ where $\alpha_1 + \alpha_2 + \alpha_3 = 2\pi$, one can place at each p_i a wedge of angle α_i such that the three wedges cover the plane and the induced mutual-visibility graph contains the shortest or the intermediate tree on P .*



■ **Figure 8** Notation for proofs.

First we provide some preliminaries for the proof; see Figure 8. We use similar notation also for the proof of Theorem 10. Let p be a point in the plane and let $[\vec{r}_1^>(p), \vec{r}_2^>(p), \vec{r}_3^>(p)]$ be the cyclic counterclockwise permutation of three rays emanating from p where the angle between any two consecutive rays is less than π . See the figure to the right for an illustration. These rays partition the plane into three cones with apices at p . For an index $i \in \{1, 2, 3\}$ we denote by $\vec{r}_{i+1}^>(p)$ the ray after $\vec{r}_i^>(p)$ in the cyclic permutation, by $\vec{r}_i^<(p)$ the ray emanating from p in the opposite direction of $\vec{r}_i^>(p)$, and by $l_i(p)$ the line through $\vec{r}_i^>(p)$. We denote by $c_i(p)$ the convex cone with boundary rays $\vec{r}_i^>(p)$ and $\vec{r}_{i+1}^>(p)$, and by $\bar{c}_i(p)$ the reflection of $c_i(p)$ with respect to p . Moreover, we denote by $c_{iR}(p)$ the portion of $c_i(p)$ that is between $\vec{r}_i^>(p)$ and $\vec{r}_{i+2}^>(p)$, and by $c_{iL}(p)$ the portion of $c_i(p)$ that is between $\vec{r}_{i+2}^>(p)$ and $\vec{r}_{i+1}^>(p)$.



■ **Figure 9** Illustration of the proof of Theorem 8.

Now we proceed with the proof. Consider the triangle with vertices $p_1, p_2,$ and p_3 . Let $\beta_1, \beta_2, \beta_3$ denote the interior angles of this triangle at p_1, p_2, p_3 respectively, and note that $\beta_1 + \beta_2 + \beta_3 = \pi$. Without loss of generality assume that $\beta_1 \leq \beta_2 \leq \beta_3$, and thus $|p_2p_3| \leq |p_1p_3| \leq |p_1p_2|$. After a suitable reflection assume that p_3 appears to the left of the ray from p_1 towards p_2 . We consider two cases where $\alpha_3 \geq \beta_3$ and $\alpha_3 < \beta_3$. For each case we show how to place three wedges w_1, w_2, w_3 (with angles $\alpha_1, \alpha_2, \alpha_3$) at points p_1, p_2, p_3 , respectively, to satisfy the conditions of the theorem.

Let $[\vec{r}_1^>(x), \vec{r}_2^>(x), \vec{r}_3^>(x)]$ be the cyclic counterclockwise permutation of three rays emanating from some point x in the plane such that the angle at each cone $c_i(x)$ is α_i for $i \in \{1, 2, 3\}$. Let $R(x)$ denote a fixed embedding of these rays in the plane, as in Figure 9(a).

- $\alpha_3 \geq \beta_3$. Translate $R(x)$ so that x lies on p_3 , i.e., $x = p_3$. Rotate $R(x)$ so that p_1 and p_2 lie in $c_3(p_3)$ and $\vec{r}_2^<(p_3)$ intersects the segment p_1p_2 (observe the existence of such rotation). Place $w_1, w_2,$ and w_3 at cones $c_1(p_3), c_2(p_3),$ and $c_3(p_3)$, respectively.

The union of these wedges covers the entire plane. Now translate w_1 and w_2 so that their apices lie on p_1 and p_2 , respectively, as in Figure 9(b). The union of the three wedges still covers the plane. In this setting, p_3 and p_1 are mutually visible and so are p_3 and p_2 . Thus, the mutual-visibility graph contains the shortest tree on P with edges $T_1 = \{p_3 \leftrightarrow p_1, p_3 \leftrightarrow p_2\}$.

- $\alpha_3 < \beta_3$. We claim that $\alpha_2 \geq \beta_2$. For the sake of contradiction assume that $\alpha_2 < \beta_2$. Then $\alpha_1 + \beta_2 + \beta_3 > \alpha_1 + \alpha_2 + \alpha_3 = 2\pi$. Since β_2 and β_3 are interior angles of a triangle, $\beta_2 + \beta_3 \leq \pi$. By combining these two inequalities we get $\alpha_1 > \pi$, which contradicts an assumption in the statement of the theorem. Thus our claim follows.

Translate $R(x)$ so that x lies on p_2 . Rotate $R(x)$ so that p_1 and p_3 lie in $c_2(p_2)$ and $\overleftarrow{r}_1(p_2)$ intersects the segment p_1p_3 . Place w_1, w_2, w_3 at cones $c_1(p_2), c_2(p_2), c_3(p_2)$. Translate w_1 and w_3 so that their apices lie on p_1 and p_3 , respectively, as in Figure 9(c). Again the three wedges cover the entire plane, and the mutual-visibility graph contains the intermediate tree on P with edges $T_2 = \{p_2 \leftrightarrow p_1, p_2 \leftrightarrow p_3\}$.

4 Approximating the $\overline{120^\circ}$ -MST

In this section we prove the following theorem.

► **Theorem 11.** *Given a set of points in the plane and an angle $\alpha \geq 120^\circ$, there is an $\overline{\alpha}$ -spanning tree of length at most 1.5 times the length of the MST and with edges of length at most 2 times the BST largest edge-length. Furthermore, there is an algorithm to find such an $\overline{\alpha}$ -ST that runs in linear time after computing the MST.*

Let P be a set of points in the plane. As in previous sections we assume that $\alpha = 120^\circ$. Let T be a degree-5 MST of P . We assign to each vertex of T an initial angle α which we refer to by “charge”. The initial charge of a vertex may not cover all its incident edges. The idea is to modify the tree locally and transfer charges between nodes to make sure that all vertices have enough charges to cover their incident edges in the new tree. Mainly we transfer charges from leaves to internal vertices, because edges incident to leaves can be covered by 0° wedges (however at the end of this section we assign to all leaves positive wedges).

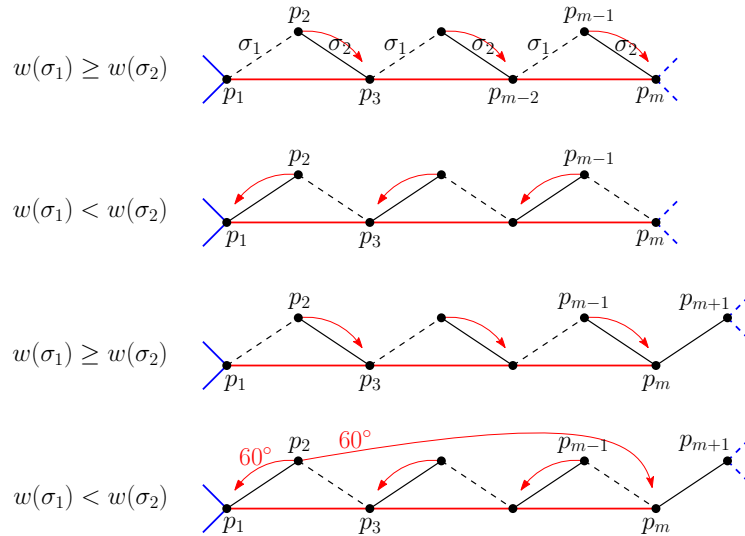
Assume that T is not a path and thus has a vertex of degree at least 3 (we describe the case where T is a path at the end of this section). A *maximal path* in T is a path with at least two edges where its internal-node degrees are 2 and its end-node degrees are not 2.

Our algorithm has two phases. Phase 1 works as follows. Contract every maximal path of T to an edge (this is done by removing the internal nodes of the path and connecting its endpoints by an edge). This results in a tree T' that has no vertex of degree 2, and moreover, the degree of each vertex in T' is the same as its degree in T . Let ℓ denote the number of leaves of T' and let $n_3, n_4,$ and n_5 denote the number of vertices of degree three, four, and five in T' respectively. Since each vertex of degree three, four, and five introduces 1, 2, and 3 new leaves respectively, we have $\ell = 2 + n_3 + 2n_4 + 3n_5$. We consider the $\ell \cdot \alpha$ charges of all leaves as a “pool of charges” that are available to be distributed among other vertices. From this pool, we give the charge $\alpha, 2\alpha,$ and 3α to each vertex of degree three, four, and five respectively. After this redistribution, every vertex of degree three, four, and five holds the charge $240^\circ, 360^\circ,$ and 480° respectively (including its initial 120° charge), which is sufficient to cover all its incident edges. Moreover, the pool is left with 2α charges.

Based on the above discussion if T has no degree-2 vertices, then it is an $\overline{\alpha}$ -ST and we are done. Now we describe Phase 2 which takes care of contracted paths; here is the place where our tree gains an extra $0.5w(T)$ length. Consider every contracted path and denote

it by (p_1, p_2, \dots, p_m) if it has an even number of edges and by $(p_1, p_2, \dots, p_m, p_{m+1})$ if it has an odd number of edges. One endpoint of this path, say p_1 , has degree at least 3 in T and the other endpoint either is a leaf or it has degree at least 3 (see Figure 10). The charges of the two endpoints of the path have already been considered in Phase 1, but the charges of its internal nodes are untouched. Let σ denote the path (p_1, p_2, \dots, p_m) ; σ does not contain p_{m+1} if this point exists. Observe that $m \geq 3$. Partition the edges of σ into matchings $\sigma_1 = \{(p_1, p_2), (p_3, p_4), \dots\}$ and $\sigma_2 = \{(p_2, p_3), (p_4, p_5), \dots\}$, as in Figure 10. Let σ_{\max} denote the heavier of σ_1 and σ_2 (i.e., the one with larger total length) and σ_{\min} denote the lighter one. Thus $w(\sigma_{\min}) \leq w(\sigma_{\max})$ and hence $w(\sigma_{\min}) \leq 0.5w(\sigma)$. We replace the edge set σ_{\max} in T by the edge set $\delta = \{(p_1, p_3), (p_3, p_5), \dots, (p_{m-2}, p_m)\}$, as in Figure 10. Let T'' be the tree obtained after performing this replacement for all contracted paths.

We claim that T'' is a desired $\bar{\alpha}$ -ST. First we show that $w(T'') \leq 1.5w(T)$. To do so, it suffices to show that, for each contracted path, the length of edges after replacement (i.e., edges of σ_{\min} and δ) is not more than 1.5 times the length of original edges (i.e., edges of σ). By the triangle inequality, $w(\delta) \leq w(\sigma)$. Therefore $w(\delta) + w(\sigma_{\min}) \leq w(\sigma) + 0.5w(\sigma) \leq 1.5w(\sigma)$; this proves the total-length constraint of Theorem 11. Moreover, the length of every edge of δ is at most twice the largest edge-length of σ (again by the triangle inequality); this proves the edge-length constraint of Theorem 11. It remains to ensure the coverage of incident edges for all vertices of T'' . To that end we distribute the charges of p_2, p_4, \dots, p_{m-1} (which are new leaves) among other vertices. We consider two cases depending on the existence of p_{m+1} . First assume that p_{m+1} does not exist. Now we consider two sub-cases depending on which of σ_1 and σ_2 is heavier (i.e., is σ_{\max}).



■ **Figure 10** The contracted path is shown by black segments. The dashed-black edges belong to σ_{\max} and the red edges belong to δ .

- $\sigma_{\max} = \sigma_1$. In this case σ_1 has been replaced by δ . This replacement has not changed the degree of p_1 and thus it holds enough charge (from Phase 1) to cover its incident edges. We move the charges of leaves p_2, p_4, \dots, p_{m-1} to vertices p_3, p_5, \dots, p_m respectively (see Figure 10). Each of p_3, p_5, \dots, p_{m-2} has degree three and now holds 240° charge (including its own 120° charge) which is sufficient to cover its incident edges. Now consider p_m and notice that its degree has been increased by one. If p_m is of degree at least 3 in T ,

then it now holds at least 360° charge (120° from p_{m-1} and at least 240° from Phase 1) which covers all its incident edges. Assume that p_m is a leaf in T , and thus it has degree 2 in T'' . The original charge of p_m has been distributed among other vertices in Phase 1, but it now has 120° charge coming from p_{m-1} . Consider the triangle $\triangle p_m p_{m-1} p_{m-2}$. The segment $p_m p_{m-2}$ is the largest side of this triangle because otherwise we could add this edge to the MST and remove the larger of the other two sides to obtain a smaller tree. Thus the angle at p_{m-1} is the largest angle of this triangle, and hence the other two angles (including the one at p_m) are at most 90° . Therefore the two edges incident to p_m can be covered by its 120° charge.

- $\sigma_{\max} = \sigma_2$. Then σ_2 has been replaced by δ . We move the charges of p_2, p_4, \dots, p_{m-1} to p_1, p_3, \dots, p_{m-2} respectively (see Figure 10). The replacement does not change the degree of p_m and thus it holds enough charge from Phase 1 to cover its incident edges. Each of p_3, p_5, \dots, p_{m-2} has degree three and now holds 240° charge which covers its incident edges. The vertex p_1 now has at least 360° charge (120° from p_2 and at least 240° from Phase 1) which covers all its incident edges.

Now assume that p_{m+1} exists. Again we consider two sub-cases.

- $\sigma_{\max} = \sigma_1$. Move the charges of p_2, p_4, \dots, p_{m-1} to p_3, p_5, \dots, p_m (see Figure 10). The degrees of p_1 and p_{m+1} have not changed in Phase 2, and thus they hold enough charge from Phase 1 to cover their incident edges. Each of p_3, p_5, \dots, p_m has degree three and now holds 240° charge which covers its incident edges.
- $\sigma_{\max} = \sigma_2$. Move the charges of p_4, p_6, \dots, p_{m-1} to p_3, p_5, \dots, p_{m-2} . Split the 120° charge of p_2 evenly between p_1 and p_m as in Figure 10. The degree of p_{m+1} has not changed and thus it holds enough charge from Phase 1. Each of p_3, p_5, \dots, p_{m-2} has degree three and now holds enough charge 240° . The vertex p_m has degree two and now holds 180° charge (including its original 120° charge) which is sufficient to cover its two incident edges. Now consider p_1 . If it has degree 4 or 5 in T then after Phase 1 it holds at least 360° charge which covers all its incident edges. Assume that it has degree 3 in T , and now it has degree 4 in T'' . Then it holds 300° charge (60° from p_2 and 240° from Phase 1) which is enough to cover its four incident edges (in fact 270° is enough). It might be the case that p_1 was also incident to another contracted path and hence has received another incident edge which increases p_1 's degree to five. In this case p_1 also receives 60° charge from the other path. This would increase p_1 's charge to 360° which covers all its incident edges.

► **Remark 2.** If T is a path then we run only Phase 2 of the above algorithm on this path. Since there is no Phase 1, the charges of path endpoints are still available. The charge redistribution of Phase 2 guarantees the coverage of incident edges of all vertices. See Figure 10. If p_{m+1} does not exist then in the first sub-case (resp. the second sub-case) the charge of p_1 (resp. p_m) remains unused. If p_{m+1} exists, then its charge remains unused.

► **Remark 3.** At the end of the algorithm we have at least 120° unused charge (from the pool or from a path endpoint). We split the unused charge evenly between all vertices such that every vertex has a positive charge and its incident edges lie strictly inside the assigned wedge.

5 Conclusions

The obvious open problem is to improve our $16/3$ approximation ratio for the 120° -MST problem further by designing better algorithms. Our proof rely on the fact that the orientation of the wedges of every triplet covers the entire plane. Such orientations are a bottleneck for

our ratio. It might be possible to get better ratio with orientations that do not necessarily cover the entire plane. Another bottleneck is the use of a Hamiltonian path which forces a factor of 2 in the ratio. It might be possible to get better ratios by using the original MST instead of the path. However, this has to be done in a clever way as the number of connecting edges incident to a triplet may increase.

References

- 1 Eyal Ackerman, Tsachik Glander, and Rom Pinchasi. Ice-creams and wedge graphs. *Computational Geometry: Theory and Applications*, 46(3):213–218, 2013.
- 2 Oswin Aichholzer, Thomas Hackl, Michael Hoffmann, Clemens Huemer, Attila Pór, Francisco Santos, Bettina Speckmann, and Birgit Vogtenhuber. Maximizing maximal angles for plane straight-line graphs. *Computational Geometry: Theory and Applications*, 46(1):17–28, 2013. Also in *WADS'07*.
- 3 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
- 4 Rom Aschner and Matthew J. Katz. Bounded-angle spanning tree: Modeling networks with angular constraints. *Algorithmica*, 77(2):349–373, 2017. Also in *ICALP'14*.
- 5 Rom Aschner, Matthew J. Katz, and Gila Morgenstern. Do directional antennas facilitate in reducing interferences? In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 201–212, 2012.
- 6 Ahmad Biniiaz. Euclidean bottleneck bounded-degree spanning tree ratios. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.
- 7 Ahmad Biniiaz, Anil Maheshwari, and Michiel Smid. Flip distance to some plane configurations. *Computational Geometry: Theory and Applications*, 81:12–21, 2019. Also in *SWAT'18*.
- 8 Prosenjit Bose, Paz Carmi, Mirela Damian, Robin Y. Flatland, Matthew J. Katz, and Anil Maheshwari. Switching to directional antennas with constant increase in radius and hop distance. *Algorithmica*, 69(2):397–409, 2014. Also in *WADS'11*.
- 9 Prosenjit Bose, Leonidas J. Guibas, Anna Lubiw, Mark H. Overmars, Diane L. Souvaine, and Jorge Urrutia. The floodlight problem. *International Journal of Computational Geometry & Applications*, 7(1/2):153–163, 1997. Also in *CCCG'93*.
- 10 Paolo M. Camerini. The min-max spanning tree problem and some extensions. *Information Processing Letters*, 7(1):10–14, 1978.
- 11 Ioannis Caragiannis, Christos Kaklamanis, Evangelos Kranakis, Danny Krizanc, and Andreas Wiese. Communication in wireless networks with directional antennas. In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 344–351, 2008.
- 12 Paz Carmi, Matthew J. Katz, Zvi Lotker, and Adi Rosén. Connectivity guarantees for wireless networks with directional antennas. *Computational Geometry: Theory and Applications*, 44(9):477–485, 2011.
- 13 Timothy M. Chan. Euclidean bounded-degree spanning tree ratios. *Discrete & Computational Geometry*, 32(2):177–194, 2004. Also in *SoCG'03*.
- 14 Mirela Damian and Robin Y. Flatland. Spanning properties of graphs induced by directional antennas. *Discrete Mathematics, Algorithms and Applications*, 5(3), 2013.
- 15 Stefan Dobrev, Evangelos Kranakis, Danny Krizanc, Jaroslav Opatrny, Oscar Morales Ponce, and Ladislav Stacho. Strong connectivity in sensor networks with given number of directional antennae of bounded angle. *Discrete Mathematics, Algorithms and Applications*, 4(3), 2012. Also in *COCOA'10*.
- 16 Stefan Dobrev, Evangelos Kranakis, Oscar Morales Ponce, and Milan Plžík. Robust sensor range for constructing strongly connected spanning digraphs in UDGs. In *Proceedings of the 7th International Computer Science Symposium in Russia (CSR)*, pages 112–124, 2012.

- 17 Adrian Dumitrescu, János Pach, and Géza Tóth. Drawing Hamiltonian cycles with no large angles. *Electronic Journal of Combinatorics*, 19(2):P31, 2012. Also in *GD'94*.
- 18 Sándor P. Fekete, Samir Khuller, Monika Klemmstein, Balaji Raghavachari, and Neal E. Young. A network-flow technique for finding low-weight bounded-degree spanning trees. *Journal of Algorithms*, 24(2):310–324, 1997. Also in *IPCO 1996*.
- 19 Sándor P. Fekete and Gerhard J. Woeginger. Angle-restricted tours in the plane. *Computational Geometry: Theory and Applications*, 8:195–218, 1997.
- 20 Magnús M. Halldórsson and Takeshi Tokuyama. Minimizing interference of a wireless ad-hoc network in a plane. *Theoretical Computer Science*, 402(1):29–42, 2008.
- 21 Raja Jothi and Balaji Raghavachari. Degree-bounded minimum spanning trees. *Discrete Applied Mathematics*, 157(5):960–970, 2009.
- 22 Matthew J. Katz. Personal communication.
- 23 Samir Khuller, Balaji Raghavachari, and Neal E. Young. Low-degree spanning trees of small weight. *SIAM Journal on Computing*, 25(2):355–368, 1996. Also in *STOC'94*.
- 24 Evangelos Kranakis, Fraser MacQuarrie, and Oscar Morales Ponce. Connectivity and stretch factor trade-offs in wireless sensor networks with directional antennae. *Theoretical Computer Science*, 590:55–72, 2015.
- 25 Clyde L. Monma and Subhash Suri. Transitions in geometric minimum spanning trees. *Discrete & Computational Geometry*, 8:265–293, 1992. Also in *SoCG'91*.
- 26 Jan van Leeuwen and Anneke A. Schoone. Untangling a travelling salesman tour in the plane. In *Proceedings of the 7th Conference Graphtheoretic Concepts in Computer Science (WG)*, pages 87–98, 1981.

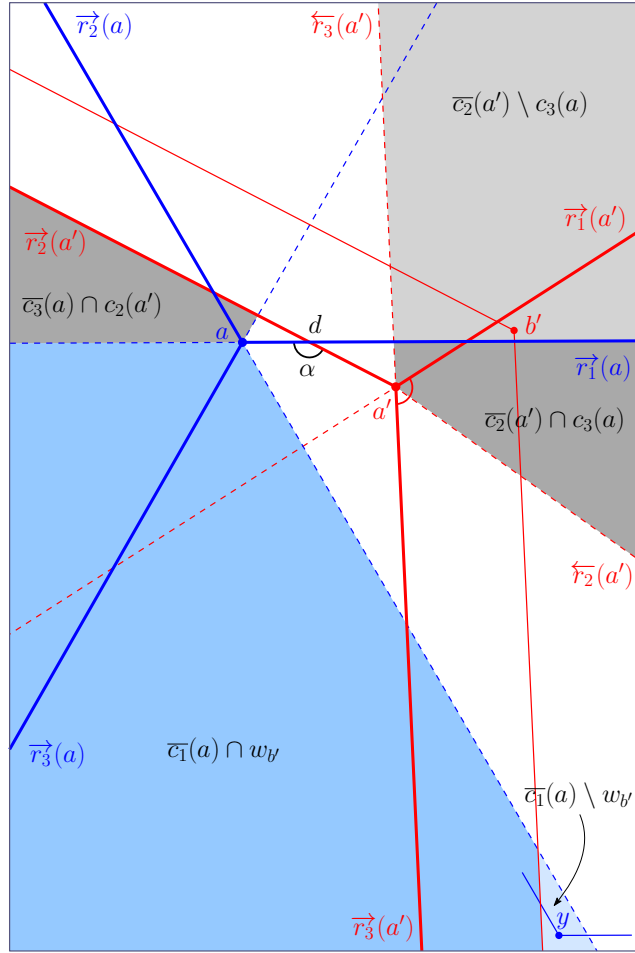
A Proof of Theorem 10

In this section we prove Theorem 10: *Let P_1 and P_2 be two disjoint sets, each containing three points in the plane, such that $\{P_1, P_2\}$ is not forbidden. Assume that a wedge of angle 120° is placed at each point of P_1 and at each point of P_2 according to the placement algorithm of Corollary 9. Then, the induced mutual-visibility graph on $P_1 \cup P_2$ is connected, and hence it contains a 120° -ST.*

We use the notation introduced in Section 3.3. Let $P_1 = \{a, b, c\}$ and $P_2 = \{a', b', c'\}$. Orient P_1 and P_2 according to Corollary 9 (in fact according to the proof of Theorem 8 with angles 120°). Let $w_a, w_b, w_c, w_{a'}, w_{b'}$ and $w_{c'}$ be the wedges of angle 120° that are placed at these points, respectively. Recall, from the proof of Theorem 8, three rays $[\vec{r}_1(x), \vec{r}_2(x), \vec{r}_3(x)]$ with cones of angles 120° that are placed at a point x . In the orientation of P_1 we may assume that these rays are placed at a , i.e., $x = a$. Thus b and c lie in the same cone $c_i(a)$ for some $i \in \{1, 2, 3\}$; in particular one of them lies in $c_{iL}(a)$ and the other lies in $c_{iR}(a)$. Moreover w_a covers $c_i(a)$ and we have $a \leftrightarrow b$ and $a \leftrightarrow c$. In the orientation of P_2 assume that these rays are placed at a' . Thus one of b' and c' lies in $c_{iL}(a')$ and the other lies in $c_{iR}(a')$ for some $i \in \{1, 2, 3\}$, $w_{a'}$ covers $c_i(a')$, and we have $a' \leftrightarrow b'$ and $a' \leftrightarrow c'$. Also recall that each cone $\bar{c}_i(a)$ contains a point of P_1 whose wedge covers $c_i(a)$, and similarly each cone $\bar{c}_i(a')$ contains a point of P_2 whose wedge covers $c_i(a')$.

Since the three cones $c_1(a), c_2(a),$ and $c_3(a)$ cover the plane, a' lies in one of them, say $c_3(a)$. Similarly, a lies in one of the three cones at a' , say $c_2(a')$. In this setting one of the following three configurations holds:

- (A) $\vec{r}_1(a)$ and $\vec{r}_2(a')$ intersect, but $\vec{r}_3(a)$ and $\vec{r}_3(a')$ do not intersect. See Figures 11 and 12.
- (B) $\vec{r}_1(a)$ and $\vec{r}_2(a')$ do not intersect, but $\vec{r}_3(a)$ and $\vec{r}_3(a')$ intersect.
- (C) $\vec{r}_1(a)$ and $\vec{r}_2(a')$ intersect, and $\vec{r}_3(a)$ and $\vec{r}_3(a')$ intersect. See Figures 13 and 14.



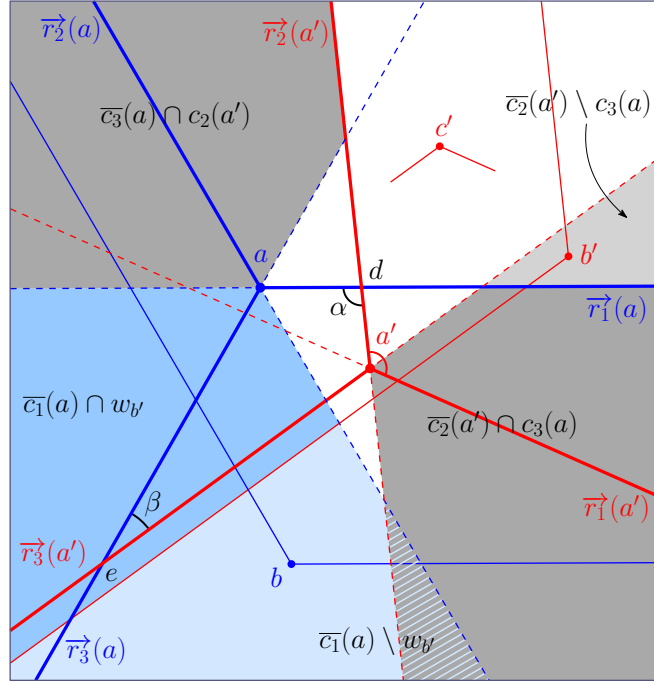
■ **Figure 11** $\vec{r}_1(a)$ and $\vec{r}_2(a')$ intersect, $\vec{r}_3(a)$ and $\vec{r}_3(a')$ do not intersect, and $b' \in c_{3L}(a') \cap c_1(a)$.

We consider each configuration separately. Since configurations (A) and (B) are symmetric, we describe only (A) and (C).

Configuration (A). Let d be the intersection point of $\vec{r}_1(a)$ with $\vec{r}_2(a')$, and let α denote the convex angle $\angle ada'$, as in Figure 11. Since $\vec{r}_3(a)$ and $\vec{r}_3(a')$ do not intersect, $\alpha \geq 2\pi/3$, and consequently a' lies in $c_{3L}(a)$ and a lies in $c_{2R}(a')$. Recall that each cone at a (resp. a') is covered by a point of P_1 (resp. P_2). Let $x \in P_1$ be the point in $\bar{c}_3(a)$ whose wedge w_x covers $c_3(a)$, and let $x' \in P_2$ be the point in $\bar{c}_2(a')$ whose wedge $w_{x'}$ covers $c_2(a')$. If $x \in \bar{c}_3(a) \cap c_2(a')$ and $x' \in \bar{c}_2(a') \cap c_3(a)$ (the dark-gray regions in Figure 11) then $x \leftrightarrow x'$ and we are done. Therefore we may assume that $x \in \bar{c}_3(a) \setminus c_2(a')$ or $x' \in \bar{c}_2(a') \setminus c_3(a)$. By symmetry we assume that $x' \in \bar{c}_2(a') \setminus c_3(a)$ (the light-gray region in Figure 11). This and the fact that $a' \in c_3(a)$ imply that $x' \neq a'$, and thus $x' \in \{b', c'\}$. After a suitable relabeling we assume that $x' = b'$. Therefore, $w_{b'}$ covers $c_2(a')$.

Since $\alpha \geq 2\pi/3$ and $a' \in c_{3L}(a)$, the rays $\overleftarrow{r}_3(a')$ and $\overleftarrow{r}_2(a')$ do not intersect the line $l_2(a)$, and consequently the cone $\bar{c}_2(a')$ does not intersect $l_2(a)$ and hence this cone is disjoint from $c_2(a)$. Therefore, b' – which is in $\bar{c}_2(a') \setminus c_3(a)$ – lies in $c_1(a)$. Let $y \in P_1$ be the point in $\bar{c}_1(a)$ whose wedge w_y covers $c_1(a)$. Since $b' \in c_1(a)$, the point y sees b' , as in Figure 11. If $y \in \bar{c}_1(a) \cap w_{b'}$ (the dark-blue region) then b' sees y , and hence $y \leftrightarrow b'$ and we

which is parallel to $\vec{r}_2(a)$; in particular this region is a subset of $c_{2R}(a)$ which is covered by w_c . Thus c sees c' . We are going to show that c' also sees c . Notice that $c_{2R}(a)$ (which contains c') is the reflection of $c_{3L}(a)$ (which contains c) with respect to a . This and the fact that $\overleftarrow{w}_{c'}$ is parallel to $\vec{r}_1(a')$ which intersects $\vec{r}_1(a)$ (because $\alpha \geq 2\pi/3$) imply that $\overleftarrow{w}_{c'}$ does not intersect $\vec{r}_1(a)$. Moreover, since $\overrightarrow{w}_{c'}$ is parallel to $\vec{r}_3(a')$ which intersects $\vec{r}_2(a)$ (because $\alpha < \pi$), $\overrightarrow{w}_{c'}$ does not intersect $\vec{r}_2(a)$. Thus $\overrightarrow{w}_{c'}$ and $\overleftarrow{w}_{c'}$ do not intersect the boundary rays $\vec{r}_1(a)$ and $\vec{r}_2(a)$ of $c_{3L}(a)$, and hence $w_{c'}$ covers $c_{3L}(a)$ which contains c . Therefore c' sees c and hence $c' \leftrightarrow c$.

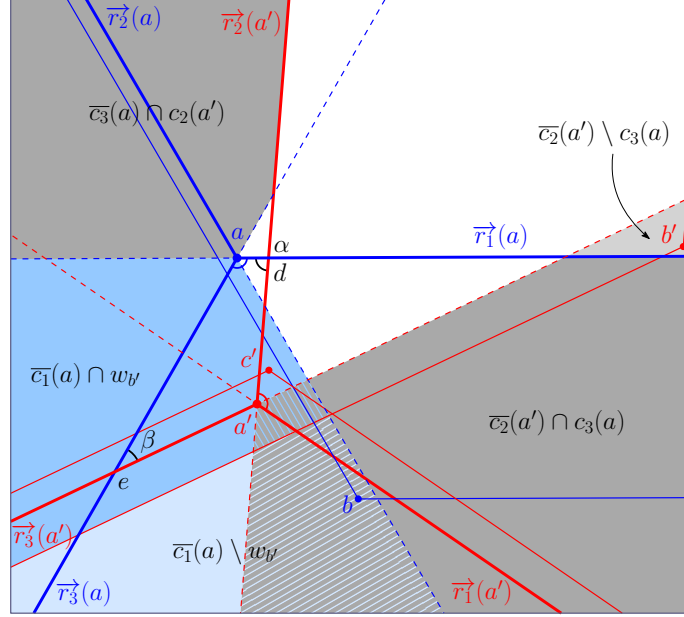


■ **Figure 13** $\vec{r}_1(a)$ and $\vec{r}_2(a')$ intersect, $\vec{r}_3(a)$ and $\vec{r}_3(a')$ intersect, and $a' \in c_{3L}(a)$.

Configuration (C). In this configuration $\vec{r}_1(a')$ does not intersect any of $\vec{r}_1(a)$, $\vec{r}_2(a)$, $\vec{r}_3(a)$, and $\vec{r}_2(a)$ does not intersect any of $\vec{r}_1(a')$, $\vec{r}_2(a')$, $\vec{r}_3(a')$. Let d be the intersection point of $\vec{r}_1(a)$ with $\vec{r}_2(a')$, and e be the intersection point of $\vec{r}_3(a)$ with $\vec{r}_3(a')$, as in Figures 13 and 14. Let α denote the convex angle $\angle ada'$, and β denote the convex angle $\angle aea'$. After a suitable relabeling we assume that $\alpha \geq \beta$, and thus $\alpha \geq \pi/3$ (notice that the sum of the interior angles of the convex quadrilateral with vertices a, d, a', e is 2π). This and the fact that $\angle ead = 2\pi/3$ imply that $\vec{r}_3(a)$ and $\vec{r}_2(a')$ do not intersect. Since $\vec{r}_3(a)$ and $\vec{r}_3(a')$ intersect, their opposite rays $\overleftarrow{r}_3(a)$ and $\overleftarrow{r}_3(a')$ do not intersect.

Let $x \in P_1$ be the point in $\overline{c_3}(a)$ whose wedge w_x covers $c_3(a)$, and let $x' \in P_2$ be the point in $\overline{c_2}(a')$ whose wedge $w_{x'}$ covers $c_2(a')$. As in configuration (A) if $x \in \overline{c_3}(a) \cap c_2(a')$ and $x' \in \overline{c_2}(a') \cap c_3(a)$ (the dark-gray regions in Figures 13 and 14) then $x \leftrightarrow x'$. Therefore we may assume by symmetry that $x' \in \overline{c_2}(a') \setminus c_3(a)$. This and the fact that $a' \in c_3(a)$ imply that $x' \neq a'$ and thus $x' \in \{b', c'\}$. After a suitable relabeling we assume that $x' = b'$, and thus $w_{b'}$ covers $c_2(a')$, as in Figures 13 and 14. The region $\overline{c_2}(a') \setminus c_3(a)$ (shown by light-gray color in Figures 13 and 14) which contains b' is in fact equal to $c_{1R}(a') \cap c_{1R}(a)$.

The wedge $w_{a'}$ covers $c_1(a')$ which contains b' . Since b' lies in $c_{1R}(a')$, the point c' lies in $c_{1L}(a')$ and $w_{c'}$ covers $c_3(a')$. Let $y \in P_1$ be the point in $\overline{c_1}(a)$ whose wedge w_y covers $c_1(a)$. Since $b' \in c_1(a)$, y sees b' . If $y \in \overline{c_1}(a) \cap w_{b'}$ (the dark-blue regions in Figures 13 and 14) then b' sees y and thus $y \leftrightarrow b'$ and we are done. Assume that $y \in \overline{c_1}(a) \setminus w_{b'}$ (the light-blue regions). This and the fact that a lies in $c_2(a')$ (which is covered by $w_{b'}$) imply that $y \neq a$ and thus $y \in \{b, c\}$. After a suitable relabeling we assume that $y = b$. Our assumption that b is not in $w_{b'}$ implies that $b \notin c_2(a')$. Recall that a' is in $c_3(a)$ and thus it lies either in $c_{3L}(a)$ or in $c_{3R}(a)$. We describe each case separately.



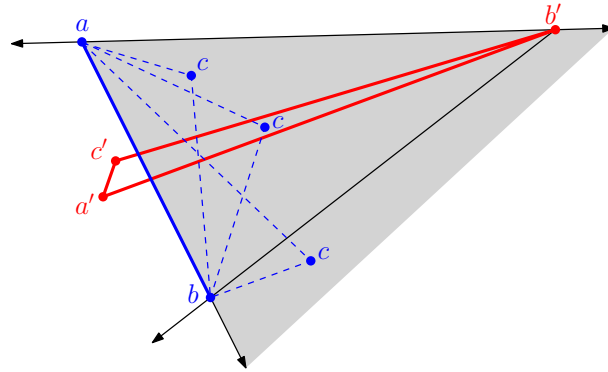
■ **Figure 14** $\overrightarrow{r_1}(a)$ and $\overrightarrow{r_2}(a')$ intersect, $\overrightarrow{r_3}(a)$ and $\overrightarrow{r_3}(a')$ intersect, and $a' \in c_{3R}(a)$.

- $a' \in c_{3L}(a)$. This case is depicted in Figure 13. In this case $\overrightarrow{r_1}(a')$ and $\overleftarrow{r_2}(a)$ do not intersect because $\alpha \geq \pi/3$. Thus $\overline{c_1}(a)$ and $c_1(a')$ are disjoint, and hence $b \notin c_1(a')$. Therefore $b \in c_3(a')$ and thus c' sees b . The boundary rays $\overleftarrow{w_b}$ and $\overrightarrow{w_b}$ of w_b do not intersect the boundary rays $\overrightarrow{r_2}(a')$ and $\overleftarrow{r_3}(a')$ of cone $c_{1L}(a')$ which contains c' . Therefore w_b covers $c_{1L}(a')$. This implies that b sees c' , and hence $b \leftrightarrow c'$.
- $a' \in c_{3R}(a)$. See Figure 14. If c' is to the left side of $\overleftarrow{r_2}(a)$ then $b \leftrightarrow c'$ and we are done. Assume that c' is to the right side of $\overleftarrow{r_2}(a)$, and thus $c' \in c_{3R}(a)$. If c' is to the right side of $\overleftarrow{w_b}$ then again $b \leftrightarrow c'$ and we are done. Assume that c' is to the left side of $\overleftarrow{w_b}$, which implies that $b \in c_{3R}(a)$ (we already knew that b is in $\overline{c_1}(a) = c_{2L}(a) \cup c_{3R}(a)$). Therefore w_a covers $c_3(a)$ which contains b , and thus c lies in $c_{3L}(a)$. This setting is depicted in Figure 14. We show that this is an invalid setting for points in P_1 and P_2 .

Consider the line through a and b . In the current setting a' and c' lie on the same side of this line, but b' lies on the other side. The points a' and c' are to the left side of the ray $\overrightarrow{b'a}$ (because $\overrightarrow{b'a}$ does not intersect the interior of $c_3(a)$ which contains a' and c') and to the right side of the ray $\overrightarrow{b'b}$ (because b is to the left side of $\overleftarrow{w_{b'}}$ while a' and c' are to its right side); see also Figure 15. Therefore a' and c' lie in the convex cone with boundary rays $\overrightarrow{b'a}$ and $\overrightarrow{b'b}$. Since c is in $c_{3L}(a)$, it is also in the convex cone with boundary rays $\overrightarrow{ab'}$ and \overrightarrow{ab} . As depicted in Figure 15, no matter where – in this cone – c lies, the two

14:22 Bounded-Angle Minimum Spanning Trees

triangles defined by the vertices of P_1 and P_2 form one of the forbidden configurations of Figure 6. This is impossible because $\{P_1, P_2\}$ is not forbidden by the statement of the theorem.



■ **Figure 15** The point c lies in the shaded cone with boundary rays $\vec{ab'}$ and \vec{ab} .

Low-Stretch Spanning Trees of Graphs with Bounded Width

Glencora Borradaile

Oregon State University, Corvallis, OR, USA
glencora@eecs.oregonstate.edu

Erin Wolf Chambers

Saint Louis University, MO, USA
erin.chambers@slu.edu

David Eppstein

University of California, Irvine, CA, USA
eppstein@uci.edu

William Maxwell

Oregon State University, Corvallis, OR, USA
maxwellw@oregonstate.edu

Amir Nayyeri

Oregon State University, Corvallis, OR, USA
nayyeria@oregonstate.edu

Abstract

We study the problem of low-stretch spanning trees in graphs of bounded width: bandwidth, cutwidth, and treewidth. We show that any simple connected graph G with a linear arrangement of bandwidth b can be embedded into a distribution \mathcal{T} of spanning trees such that the expected stretch of each edge of G is $O(b^2)$. Our proof implies a linear time algorithm for sampling from \mathcal{T} . Therefore, we have a linear time algorithm that finds a spanning tree of G with average stretch $O(b^2)$ with high probability. We also describe a deterministic linear-time algorithm for computing a spanning tree of G with average stretch $O(b^3)$. For graphs of cutwidth c , we construct a spanning tree with stretch $O(c^2)$ in linear time. Finally, when G has treewidth k we provide a dynamic programming algorithm computing a minimum stretch spanning tree of G that runs in polynomial time with respect to the number of vertices of G .

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treewidth, low-stretch spanning tree, fundamental cycle basis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.15

Related Version Full version hosted on arXiv: <https://arxiv.org/abs/2004.08375>.

Funding This material is based upon work supported by the National Science Foundation under Grant Nos. NSF CCF-1614562, DBI-1759807, CCF-1907612, CCF-1616248, CCF-1617951, CCF-1618301, and CCF-1816442.

1 Introduction

Let $G = (V, E)$ be an unweighted, connected graph with m edges and n vertices, and T be any spanning tree of G . For any $(u, v) \in E$, the stretch of (u, v) with respect to T is $stretch_T(u, v) = d_T(u, v)$, where $d_T(u, v)$ denotes the length of the unique u -to- v path in T . The stretch of T is then defined to be $stretch(T) = \frac{1}{m} \sum_{(u,v) \in E} stretch_T(u, v)$.

As minimal distance preserving structures, low-stretch spanning trees are a fundamental concept that have been studied extensively; they have also found applications in computer science in problems such as the k -server problem [3], minimum cost communication trees [26],



© Glencora Borradaile, Erin Wolf Chambers, David Eppstein, William Maxwell, and Amir Nayyeri; licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 15; pp. 15:1–15:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and solving diagonally dominant linear systems [21]. Perhaps the first notable structural result is the paper by Alon et al. [3], where they show that any general graph has a spanning tree of stretch $O(\exp(\sqrt{\log n \log \log n}))$ and that there exist graphs with minimum stretch $\Omega(\log n)$. A series of papers [11, 1, 22, 2] followed the result of Alon et al., culminating in the recent construction of Abraham and Neiman of an $O(\log n \log \log n)$ stretch spanning tree for general graphs, which is almost tight considering the $\Omega(\log n)$ lower bound. The existence of spanning trees with bounded average distortion is often implied by a stronger statement that the graph can be embedded into a distribution of spanning trees such that the expected stretch of any edge is bounded.

Given these results for general graphs, a natural question is to consider restricted classes of graphs, both in terms of finding better bounds than general graphs for some classes of graphs, as well as finding lower bounds that match the general case in others. For example, we know that constant factor stretch spanning trees exist for k -outerplanar graphs: they have stretch c^k for a constant c [17, 12]. On the lower bound side, we also know that grid graphs, which are planar, have a lower bound of $\Omega(\log n)$ on their stretch, so we cannot hope to get constant factor for this class. Additionally, Gupta et al. [17] found a family of bounded treewidth graphs (in fact, series parallel graphs) whose minimum stretch spanning trees have stretch $\Omega(\log n)$.

In light of these bounds, the search for families of graphs that might have smaller stretch must be limited to classes of graphs that exclude these examples. In this regard, a natural and still-open question is whether bounded pathwidth graphs admit a spanning tree of sublogarithmic stretch. In fact, we conjecture that bounded pathwidth graphs admit constant stretch spanning trees. In this paper, we make progress towards this conjecture by showing this is true for bounded bandwidth (Theorem 3) and bounded cutwidth graphs (Theorem 4); both classes are contained within the family of bounded pathwidth graphs. More precisely, we prove:

- For every n -vertex graph of bandwidth b there exists a random distribution over spanning trees of the graph, such that the expected stretch of any individual edge of the graph is $O(b^2)$. The random distribution can be sampled in linear time given a bandwidth- b linear arrangement of the graph, or constructed explicitly in quadratic time.
- Under the same assumptions, a spanning tree T of average stretch $O(b^3)$ can be constructed deterministically in linear time.
- Every n -vertex graph of cutwidth c has a spanning tree T of average stretch $O(c^2)$. T can be constructed from a cutwidth- c linear arrangement of the graph in linear expected time.
- We provide a dynamic programming algorithm computing the minimum stretch spanning tree of an unweighted graph with treewidth k . Our algorithm runs in $O(2^{3k} k^{2k} n^{k+1})$ time.

It is important to note that our algorithms require either a linear arrangement or a tree decomposition realizing the width as input, and computing such structures is NP-hard [25, 4, 15]. Due to space constraints the deterministic algorithm and some proofs have been placed in the appendix.

Lee and Sidiropoulos [23] show that a bounded pathwidth graph admits an embedding into a distribution of trees with constant distortion. In this paper, we conjecture that a similar result holds for embedding into a distribution of *spanning* trees. For embedding of bounded bandwidth graphs into normed spaces see Carrol et al. [8] and Bartal et al. [5].

The key insight by which we obtain these results lies in the connection between spanning trees of low-stretch and fundamental cycle bases of low weight. Any spanning tree T of G naturally gives a fundamental cycle basis for G : for each $e = (u, v) \in E \setminus T$, the basis contains

the unique cycle in $T \cup \{e\}$. The weight of this basis is defined to be the sum of the lengths of its cycles. A graph G has a spanning tree of average stretch $O(\log n)$ if and only if it has a fundamental cycle basis of weight $O(m \log n)$. Similarly, a cycle basis of length $O(m)$ is equivalent to a spanning tree of stretch $O(1)$. (The relationship between T 's stretch and fundamental cycle basis will be discussed in more detail in the next section.)

Shortest fundamental cycle bases have been studied as a basic structure of graphs and for their different applications in graph drawing [14], electrical engineering [7], chemistry [16], traffic light planning [20], periodic railway time tabling, [24, 27], and kinematic analysis of mechanical structures [9].

2 Preliminaries

2.1 Cycle bases

Given a simple, connected, unweighted graph G with n vertices and m edges the *cycle space* of G is an $m - n + 1$ dimensional vector space over \mathbb{Z}_2 that spans the cycles in G . In this context a cycle in G is any subgraph of G with even degree. We call a basis of this vector space a *cycle basis*, and the weight of a cycle basis is the sum of the lengths of the cycles in the basis. Given a spanning tree T of G we call a cycle formed by adding a non-tree edge to T a *fundamental cycle* with respect to T . Every spanning tree T of G yields a basis of the cycle space using the fundamental cycles induced by the $m - n + 1$ edges in $G \setminus T$. We call a basis of this form a *fundamental cycle basis*. Each cycle in the fundamental cycle basis created by T corresponds to exactly one edge in $G \setminus T$. We call this edge the *fundamental edge* of the cycle.

2.2 Fundamental cycle bases and low-stretch spanning trees

The weight of a fundamental cycle basis with respect to a tree T is closely related to the stretch of T . The *stretch* of an edge $e = (u, v)$ in G with respect to T , denoted $stretch_T(e)$, is defined as the length of the unique u -to- v path in T . The stretch of T is defined as the mean stretch of the edges,

$$stretch(T) = \frac{1}{m} \sum_{e \in E(G)} stretch_T(e).$$

Let $FCB(T)$ denote the weight of the fundamental cycle basis corresponding to T . By observing that the length of a fundamental cycle induced by an edge e is $stretch_T(e) + 1$ we see that the fundamental cycle basis with respect to T is related to the stretch of T by

$$FCB(T) = m \cdot stretch(T) + m - 2n + 2 \quad (1)$$

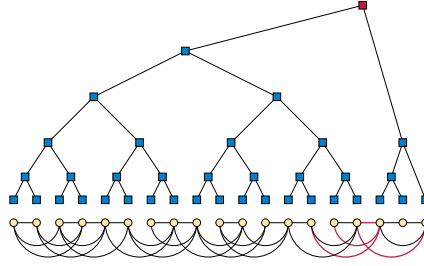
It follows that $FCB(T) = O(m)$ if and only if $stretch(T) = O(1)$.

2.3 Linear arrangements

A bijective map $\phi: V(G) \rightarrow \{1, 2, \dots, n\}$ is called a *linear arrangement* of G . For any subset of vertices $S \subseteq V(G)$ if $s \in S$ maximizes ϕ restricted to S we call it the *right endpoint* of S ; similarly if s minimizes ϕ restricted to S we call it the *left endpoint* of S . If u and v are the left and right endpoints of S we define the *spread* of S to be $\phi(v) - \phi(u)$. For any vertex v we call the sets $\{u \in V(G) \mid \phi(u) < \phi(v)\}$ and $\{u \in V(G) \mid \phi(v) < \phi(u)\}$ the *left and right sides* of v , respectively.

2.4 The arrangement tree

Given a linear arrangement ϕ of G the *arrangement tree* A is defined as a balanced binary tree with the following two properties. The leaves of A are in bijection with $V(G)$ and each internal node v is mapped to the subgraph of G induced by the vertices corresponding to the descendent leaves of v . More specifically we construct A as follows: let n be the number of vertices in G , and let p be the largest power of two that is less than n . Let the left subtree of A be constructed recursively from the first p vertices in the linear arrangement, and let the right subtree be constructed recursively from the remaining $n - p$ vertices (Figure 1).



■ **Figure 1** Linear arrangement of a graph of bandwidth three and its arrangement tree. The root node and the edges split by the root node are marked in red.

We denote the induced subgraph of the leaves descending from v by G_v . Consider the children x and y of v in A . The induced subgraph G_v has the form $G_v = G_x \cup G_y \cup S_v$ where S_v is the set of edges connecting G_y and G_x . We call S_v the set of edges *split* by v . Note that each edge is split by exactly one vertex.

2.5 Bandwidth and cutwidth

The *bandwidth* of a linear arrangement ϕ of a graph G is defined as

$$\max_{(u,v) \in E(G)} |\phi(u) - \phi(v)|.$$

Note that $|\phi(u) - \phi(v)|$ is the spread of (u, v) with respect to the arrangement tree arising from ϕ . The bandwidth of G is the minimum bandwidth over all possible linear arrangements. In a graph with bandwidth b we have $\deg(v) \leq 2b$ for all $v \in V(G)$. Hence, when $b = O(1)$ we have $|E(G)| = O(n)$. Consider the induced subgraph $G_v = G_x \cup G_y \cup S_v$ corresponding to node v of A with x as the left and y as the right child of v . Any edge $(q, r) \in S_v$ with q and r from G_x and G_y , respectively, has spread at most b . So, if r is i positions away from the left endpoint of G_y then q is at most $b - i$ positions away from the right endpoint of G_x . It follows that

$$|S_v| \leq \frac{1}{2}(b - 1)(b - 2) = O(b^2). \tag{2}$$

The *cutwidth* of a linear arrangement ϕ of a graph G is defined as

$$\max_{i \in \mathbb{Z}} |\{(u, v) \in E(G) \mid \phi(u) \leq i, \phi(v) \geq i + 1\}|.$$

The cutwidth of G is the minimum cutwidth over all linear arrangements. The cutwidth measures the number of edges that cross a fixed position in the linear arrangement.

2.6 Tree decompositions

A *tree decomposition* of a graph G is a tree $\mathcal{D} = (\mathcal{I}, \mathcal{E})$ where the vertex set \mathcal{I} is in bijection with a collection $\{B_i\}_{i \in \mathcal{I}}$ of subsets of $V(G)$, called *bags*, meeting the following conditions.

1. Every vertex $v \in V(G)$ is contained in some bag. That is, $\bigcup_{i \in \mathcal{I}} B_i = V(G)$.
 2. For every edge $(u, v) \in E(G)$ there exists an $i \in \mathcal{I}$ with $u, v \in B_i$.
 3. For all $v \in V(G)$ the subgraph induced by the set of bags containing v is a tree.
- The *width* of a tree decomposition is defined to be $\max_{i \in \mathcal{I}} |B_i| - 1$. The *treewidth* of G is the minimum width over all of its tree decompositions, denoted k . We will use the notation $D(B)$ to refer to the set of vertices in the bag B and the descendants of B . Similarly, by $A(B)$ we denote the set of vertices in B and the ancestors of B . We call a tree decomposition a *nice tree decomposition* if it meets the following extra conditions.
4. \mathcal{D} is a rooted binary tree.
 5. If $i, j, k \in \mathcal{I}$ with j and k the children of i , then $B_i = B_j = B_k$.
 6. If j is the child of i and $\deg(i) = 2$ then either $B_j \subset B_i$ and $|B_i| = |B_j| + 1$ or $B_i \subset B_j$ and $|B_i| = |B_j| - 1$.

We call the parent bags satisfying property 5 *join nodes*. We call the parent bags satisfying the two conditions of property 6 *introduce nodes* and *forget nodes*, respectively. Without loss of generality we may assume all tree decompositions are nice since any tree decomposition can be transformed into a nice tree decomposition in polynomial time [6]. Further, we may assume that every leaf bag contains only one vertex and the root bag is a forget node containing only one vertex.

3 Spanning trees from linear arrangements

Both our construction of a random family of spanning trees with low expected stretch on each edge and our construction of a deterministic spanning tree with low mean stretch will depend on a construction of spanning trees from arrangement trees, which we now describe.

Although we will use a different construction algorithm, our tree can be described as the one constructed by the following greedy algorithm:

■ **Algorithm 1** Spanning tree from a linear arrangement.

Given a graph G and arrangement tree A :
 $T \leftarrow \emptyset$
 for node $x \in A$ in leaf-to-root order:
 for edge $e \in S_x$ in increasing order by spread:
 if $T \cup \{e\}$ is acyclic, add e to T .
 Return T .

This algorithm is simply Kruskal's algorithm for the minimum spanning tree of G , with each edge weighted by the height in the arrangement tree of the least common ancestor of the edge endpoints with ties broken by spread. Because the result is a minimum spanning tree for these edge weights, we can construct the same tree by any other minimum spanning tree algorithm. Finding the lowest common ancestor for all edges in G can be done in $O(n)$ time [18]. The algorithm of Fredman and Willard [13], which finds a minimum spanning tree of a graph with integer weights in $O(n)$ time, implies that our algorithm can be implemented in linear time.

► **Lemma 1.** *Let e be an arbitrary edge of G , let i and j (with $1 \leq i < j \leq n$) be the positions of the endpoints of e in the linear arrangement, and let p be the largest power of two that divides an integer in the half-open interval $[i, j)$. Then the stretch of e in the tree constructed as above is $O(p)$.*

Proof. Let v be the node of the arrangement tree with $e \in S_v$. From our construction of the arrangement tree it follows that the number of leaf descendants of v is at least $p + 1$ and at most $2p$. By the greedy algorithm for the construction of a spanning tree, the spanning tree contains a path connecting the endpoints of e within these at most $2p$ descendants, for otherwise e itself would have been added to the spanning tree. Therefore, the stretch of e is at most $2p - 1$. ◀

4 Embedding into a distribution of trees

Let G be any graph having a linear arrangement ϕ of bandwidth b . In this section, we construct a random distribution over spanning trees \mathcal{T} of G with the property that each edge of G has expected stretch $O(b^2)$. That is, for an arbitrary edge e (chosen independently from the construction of \mathcal{T}) we have $\mathbb{E}_{\mathcal{T}}[\text{stretch}(e)] = O(b^2)$. A single tree from the distribution can be sampled in time $O(n)$, and the entire distribution can be constructed explicitly in time $O(n^2)$.

Let n be the number of vertices in G , and let n' be the smallest power of two greater than or equal to $2n$ (so, $n' = \Theta(n)$). Let G' be formed from G by adding $n' - n$ isolated vertices. Consider the $n' - n \geq n$ different linear arrangements ϕ_i of G' obtained from the linear arrangement ϕ of G by placing i isolated vertices before the vertices of G and $n' - n - i$ vertices after the vertices of G (for $0 \leq i \leq n' - n$). Denote the collection of arrangement trees of these linear arrangements by $\mathcal{A} = \{A_i\}_{i=1}^{n'-n}$. For each arrangement tree $A_i \in \mathcal{A}$, Algorithm 1 produces a tree T_i . Our random distribution \mathcal{T} is generated by choosing i uniformly at random and, based on that choice, selecting tree T_i .

Given a fixed choice of edge e , define $\ell(A_i)$ to be the node v of the arrangement tree A_i such that $e \in S_v$ (that is, the endpoints of e are in distinct children of v). Given two arrangement trees A_i and A_j we say $A_i \equiv A_j$ if the rightmost leaf descendants of the left children of $\ell(A_i)$ and $\ell(A_j)$ are equal. That is, $A_i \equiv A_j$ are equivalent if and only if e is split in the same position of the linear arrangements ϕ_i and ϕ_j . Note that \equiv is an equivalence relation that is defined with respect to a fixed e .

Therefore, we can calculate the expected spread of e by concentrating only on a single equivalence class $[A]$ of \equiv . Since the bound holds for every equivalence class, the same expected spread will hold for our entire random distribution, by averaging over the equivalence classes.

Given an arrangement tree A_i (chosen from a fixed equivalence class $[A]$) let v_i be the node of A_i such that $e \in S_{v_i}$ (that is, v_i splits e), and let h_i be the height of v_i in the arrangement tree. Then for all A_j in the same equivalence class with $h_i = h_j$, we have $G_{v_i} = G_{v_j}$ and the edges in this induced subgraph have the same minimum spanning tree weights, so they also have $T_i \cap G_{v_i} = T_j \cap G_{v_j}$. Within these two subtrees these nodes have the same two paths connecting the endpoints of e . Because this path depends only on the height h_i and not on i itself, we denote it P_{h_i} . Different heights may have the same associated paths. We say that h_i is a *critical height* if $P_{h_i} \neq P_{h_i-1}$; that is, if h_i is the lowest height that gives rise to its path.

► **Lemma 2.** *For a fixed choice of edge e and equivalence class $[A]$ there are $O(b)$ critical heights.*

Proof. Let $A_i, A_j \in [A]$ be arrangement trees that split the edge e at vertices v_i and v_j , respectively. Further, we assume v_i and v_j are at heights h_i and $h_j = h_i - 1$ where h_i is a critical height. We denote the spanning trees produced by Algorithm 1 with input A_i and A_j by T_i and T_j . The associated induced subgraphs are related by the inclusion $G_{v_j} \subset G_{v_i}$.

We now describe the ways in which $T_{v_i} = T_i \cap G_{v_i}$ can differ from $T_{v_j} = T_j \cap G_{v_j}$. By the construction of the equivalence relation every edge split by v_j is also split by v_i , that is $S_{v_j} \subset S_{v_i}$. The edges in $T_{v_j} \setminus S_{v_j}$ must be included in T_{v_i} since their weights are the same in both A_i and A_j . This is because in the linear arrangement G_{v_i} adds an equal number of vertices to the left and right of G_{v_j} and this number is equal to a power of two. It follows that T_{v_i} differs from T_{v_j} by the addition of non-split edges, the potential addition of split edges, and the potential removal of split edges.

Consider the case when there exists some edge $e' \in T_{v_j} \cap S_{v_j}$ but $e' \notin T_{v_i}$. The edge e' was added to T_{v_j} by Algorithm 1 because it connected to previously disconnected components of G_{v_j} . These connected components must have already been contained in a larger connected component of G_{v_i} , since otherwise Algorithm 1 would have picked e' for T_{v_i} . It follows that these connected components must have been connected by the addition of a non-split edge not contained in T_{v_j} .

When $e' \in T_{v_i} \cap S_{v_i}$ but not in T_{v_j} then e' must contain an endpoint outside of G_{v_j} . Since there are $O(b)$ vertices within b positions away from the split point, and once a critical height excludes a split edge it cannot be reintroduced to the spanning tree, we see that at most $O(b)$ split edges can be added across all critical heights.

The height h_i can only be a critical height if $T_{v_i} \cap G_{v_j}$ differs from T_{v_j} , otherwise $P_{h_i} = P_{h_j}$. Hence, h_i can only be a critical height if T_{v_i} excludes a split edge appearing in T_{v_j} . The number of split edges at the smallest critical height is $O(b)$ because these edges form an acyclic subgraph on the $O(b)$ vertices within b positions away from the split point. Since an edge can be excluded from the spanning tree at a critical height at most once we conclude that there are $O(b)$ critical heights. ◀

► **Theorem 3.** *For an arbitrary edge e (chosen independently from the construction of \mathcal{T}) the expected stretch of e is $O(b^2)$.*

Proof. Let $[A]$ be any equivalence class of the equivalence relation \equiv , and let $stretch_{[A]}(e)$ denote the expected stretch of e over all arrangement trees from the class $[A]$. Also, let $h_1 < h_2 < \dots < h_k$ be the critical heights of e in $[A]$. Finally, let v be the (random) vertex in the arrangement tree that splits e , and let H_v be the random variable of v 's height. We have

$$\mathbb{E}[stretch_{[A]}(e)] = \sum_{i=1}^k len(P_{h_i}) \cdot Pr[P_{h_i}],$$

where $Pr[P_{h_i}]$ is the probability that P_{h_i} is the path connecting the endpoints of e in the (randomly) selected tree. It follows, by the definition of critical heights, that

$$Pr[P_{h_i}] = Pr[h_i \leq H_v < h_{i+1}] \leq Pr[h_i \leq H_v] = O(spread(e)/2^{h_i}).$$

In addition, we have $len(P_{h_i}) = O(2^{h_i})$ by Lemma 1. Putting everything together, we have

$$\mathbb{E}[stretch_{[A]}(e)] = \sum_{i=1}^k O(2^{h_i}) \cdot O(spread(e)/2^{h_i}) = O(k \cdot spread(e)) = O(b^2),$$

as $k = O(b)$ by Lemma 2, and $spread(e) \leq b$ by the definition of bandwidth.

Since $stretch(e)$ is a weighted average of $stretch_{[A]}(e)$ for different classes $[A]$, and $stretch_{[A]}(e) = O(b^2)$ for all classes $[A]$, we conclude that $stretch(e) = O(b^2)$. ◀

5 Bounded cutwidth

A theorem from Chung [10] says that for any graph G with cutwidth c there exists a subdivision of G with bandwidth c . However, this entails expanding the number of edges by a factor of c , so combining this with our construction of low-stretch spanning trees for low-bandwidth graphs would give us a tree with average stretch $O(c^3)$. In this section we provide a direct construction that obtains stretch $O(c^2)$. The proof of Theorem 4 is almost identical to that of Theorem 3, however since we do not have the inequality $spread(e) \leq c$ we instead compute the expected stretch of the tree rather than the expected stretch of a single edge.

► **Theorem 4.** *A graph G with cutwidth c has a spanning tree with expected stretch $O(c^2)$.*

Proof. We apply the same construction for a random distribution of spanning trees as in Theorem 3 to a linear arrangement of G with cutwidth c . We show that the expected stretch a spanning tree produced by Algorithm 1 on a randomly chosen arrangement tree from the distribution is $O(c^2)$. Therefore, there exists a spanning tree with stretch at least as good as this expected value.

As before, we fix an equivalence class of arrangement trees $[A]$ from our random distribution. Let $h_1 < h_2 < \dots < h_k$ denote the critical heights of $[A]$. Since at h_k there are at most $O(c)$ split edges, we can conclude that there are at most $O(c)$ critical heights. As in the proof of Theorem 3, for a fixed edge e the expected stretch is given by

$$\mathbb{E}[stretch_{[A]}(e)] = \sum_{i=1}^k len(P_{h_i}) \cdot Pr[P_{h_i}].$$

We have that $Pr[P_{h_i}] = O(spread(e)/2^{h_i})$ and $len(P_{h_i}) = O(2^{h_i})$, hence $\mathbb{E}[stretch_{[A]}(e)] = O(c \cdot spread(e))$. Let T be the spanning tree constructed by Algorithm 1 from the randomly selected arrangement tree. We compute the expected stretch of T by

$$\mathbb{E}[stretch(T)] = \frac{1}{m} \sum_{e \in E(G)} O(c \cdot spread(e)).$$

Note that $\sum_{e \in E(G)} spread(e) \leq cn$ since by the definition of cutwidth at most c edges cross any given interval in the linear arrangement. Hence, $\mathbb{E}[stretch(T)] = O(c^2)$. ◀

► **Corollary 5.** *Any graph with cutwidth c has a fundamental cycle basis with weight $O(c^2n)$.*

Because this method produces high expected stretch for edges of high spread, it is not clear how to strengthen this result to obtain a distribution with low-stretch for each edge, as we did for bandwidth. We leave the question of whether this is possible as open for future research.

6 Bounded treewidth

In this section we consider simple, connected, unweighted graphs with fixed treewidth k . We provide a dynamic programming approach computing a spanning tree that minimizes the total stretch over all spanning trees of G . The dynamic programming table indexes partial solutions based on a localized view of the complete solution from a bag of the tree decomposition. This is done by indexing the table with trees that correspond with weighted contracted spanning trees of G that retain the stretch of the edges inside the current bag. The approach yields a dynamic programming table whose size is polynomial in n but superexponential in k . The goal of this section is to prove the following theorem.

► **Theorem 6.** *A minimum stretch spanning tree of a graph with n vertices and treewidth k can be computed in $O(2^{3k}k^{2k}n^{k+1})$ time.*

6.1 Spanning trees conforming to a configuration

Let \mathcal{T} be a spanning tree of G and (T, c) be a tuple consisting of a tree T and a weight function c on the edges of T . Fix a bag B in the tree decomposition of G . We say that \mathcal{T} *conforms* to (T, c) if \mathcal{T} can be transformed into T by in the following way. Initialize $c(e) = 1$ for every edge in \mathcal{T} and update by applying the following contractions while any of them is possible.

1. If e is not contained in any (u, v) -path where $u, v \in B$ then contract e .
2. If $e = (u, v)$ where $u, v \notin B$ and $\deg_{\mathcal{T}}(v) = 2$ then contract e . Let e' be the other edge incident to v . Set $c(e') := c(e) + c(e')$.
3. If $e = (u, v)$ where $u \in B$, $v \notin B$, and $\deg_{\mathcal{T}}(v) = 2$ then contract e . Let e' be the other edge incident to v . Set $c(e') := c(e) + c(e')$.

T is the unique minimal minor of \mathcal{T} retaining the structure of the paths between vertices in B . We call a tuple (T, c) a *configuration* of the bag B . In Lemma 7, we will show that any spanning tree \mathcal{T} conforms to a bounded number of configurations. Our dynamic program will maintain an array of forests $\text{DP}_i[T, c]$ indexed by a bag B_i of the tree decomposition and all configurations with respect to the bag. Each configuration at B_i will describe a spanning tree \mathcal{T} on G that has been contracted in the way described above. We say a forest F *meets* a configuration (T, c) if by following the contraction rules stated above F can be transformed into $T \setminus S^A$ for some $S^A \subseteq V(T) \setminus V(B)$. We will define the subset S^A in the following paragraph. The solution stored at $\text{DP}_i[T, c]$ will be the minimum cost forest of $G[D(B_i)]$ meeting the configuration (T, c) . We will describe how to calculate the cost of F in the next subsection. We will use $\text{DP}_i[T, c]$ to refer to the total stretch of the partial solution and use F to denote the partial solution that has been computed.

Let \mathcal{T} be a tree built by our dynamic program conforming to (T, c) and let v_1, \dots, v_n be a path in T such that $v_1, v_n \in V(B)$ and $v_2, \dots, v_{n-1} \in V(T) \setminus V(B)$. By property 3 of the tree decomposition either $v_2, \dots, v_n \in D(B)$ or $v_2, \dots, v_n \in A(B)$. We call the vertices in $V(T) \setminus V(B)$ Steiner vertices and partition them into two sets S^A and S^B , the Steiner vertices above the bag and the Steiner vertices below the bag. A forest F meets the configuration (T, c) if it can be transformed into $T \setminus S^A$ following our contraction scheme. The cost of F is defined to be the sum $\sum_{e \in E(G)} \text{stretch}_F(e)$ where $\text{stretch}_F(e)$ is the stretch of e in F when e 's endpoints are in the same connected component of F , when e 's endpoints are in different connected components we set $\text{stretch}_F(e)$ to be the distance between e 's endpoints in T weighted by the cost function c . Our dynamic program will process the bags of the tree decomposition in a leaf-to-root order. Paths in S^A will represent paths that will eventually be added to the complete solution by the dynamic program and paths in S^B will represent paths that have already been added to the partial solution by the dynamic program.

► **Lemma 7.** *Let \mathcal{T} be a spanning tree of G . There is a configuration (T, c) at bag B that \mathcal{T} conforms to such that $|V(T)| = O(k)$.*

We now describe how to populate each entry in the dynamic programming table by considering each type of bag separately. We will prove that the forests indexed at each entry $\text{DP}_i[T, c]$ span $D(B_i)$ and minimize the cost over all forests meeting the configuration (T, c) . We will prove each case by induction using the fact that any solution stored at a leaf node is a single vertex as our base case.

6.2 Leaf nodes

If B_i is a leaf node in the tree decomposition it contains one vertex v . The only configuration on B_i is $(\{v\}, \emptyset)$ where \emptyset is the empty function. We initialize $F_i := \{v\}$ and $\text{DP}_i[\{v\}, \emptyset] := 0$.

6.3 Introduce nodes

When B_i is an introduce node with child B_j we have $B_i = B_j \cup \{v\}$ where v is the vertex being introduced to B_i . Let (T_i, c_i) and (T_j, c_j) be configurations of B_i and B_j . Let F_j be the partial solution stored at $\text{DP}_j[T_j, c_j]$. We say (T_i, c_i) and (T_j, c_j) are compatible with one another if (T_i, c_i) can be constructed from (T_j, c_j) in a way that extends F_j to a partial solution F_i in the following way. If \mathcal{T} is a spanning tree conforming to (T_j, c_j) such that $\mathcal{T}[D(B_j)] = F_j$ we construct F_i and (T_i, c_i) such that $\mathcal{T}[D(B_i)] = F_i$ and \mathcal{T} conforms to (T_i, c_i) . We enumerate the six ways F_j can be extended to F_i meeting this criteria; by $N(v)$ and $I(v)$ we denote the neighbors of a vertex v and the edges incident to v .

- 11 Let $e = (v, u) \in E(G)$ with $u \in E(G[B_i])$. Define $T_i := T_j \cup \{e\}$ and $c_i(e) = \ell(e)$. This extends F_j to $F_i := F_j \cup \{e\}$.
- 12 Let v be adjacent to some set of vertices $B_v \subseteq B$ in G and let $s \in S_j^A$ such that $B_v = N(s) \cap B_j$ and $c_j(b, s) = 1$ for each $b \in B_v$. Define $S_i^A := S_j^A \setminus \{v\}$, $S_i^B := S_j^B$, and $E(T_i) := E(T_j) \cup I(v) \cap E(B_i)$ with $c_i(e) = 1$ for all $e \in I(v) \cap E(B_i)$ and $c_i(e) = c_j(e)$ for all $e \notin I(v) \cap E(B_i)$. This extends F_j to $F_i := F_j \cup (I(v) \cap I(B_v))$.
- 13 Let v be adjacent to some vertex $b \in B_j$ in G . Let b be adjacent to some Steiner vertex $s \in S_j^A$ with $c_j(b, s) > 1$. Define $T_i := T_j \cup \{(v, b), (v, s)\}$ with $c_i(v, s) := c_j(v, s) - 1$. This extends F_j to $F_i := F_j \cup \{(v, b)\}$.
- 14 Let $s \in S_j^A$ and define $T_i := T_j \cup \{(v, s)\}$ with $1 \leq c_i(v, s) \leq n$. This extends F_j to $F_i := F_j \cup \{v\}$.
- 15 Define $S_i^A := S_j^A \cup \{s\}$ and let $b \in B_j$. Define $T_i := T_j \cup \{(v, s), (b, s)\}$ with $1 \leq c_i(v, s) \leq n$ and $1 \leq c_i(b, s) \leq n$. This extends F_j to $F_i := F_j \cup \{v\}$.
- 16 Let $s \in S_j^A$ be a Steiner vertex with $\deg(s) > 2$ and let $b \in B_j$ be adjacent to s in T_j . We remove (b, s) and introduce a new Steiner vertex s' with edges (b, s') , (s, s') , and (v, s') . Hence $S_i^A := S_j^A \cup \{s'\}$ and $T_i := (T_j \setminus \{(b, s)\}) \cup \{(b, s'), (s, s'), (v, s')\}$ such that $c_i(b, s') + c_i(s, s') = c_j(b, s)$ and $1 \leq c_i(v, s') \leq n$. This extends F_j to $F_i := F_j \cup \{v\}$.

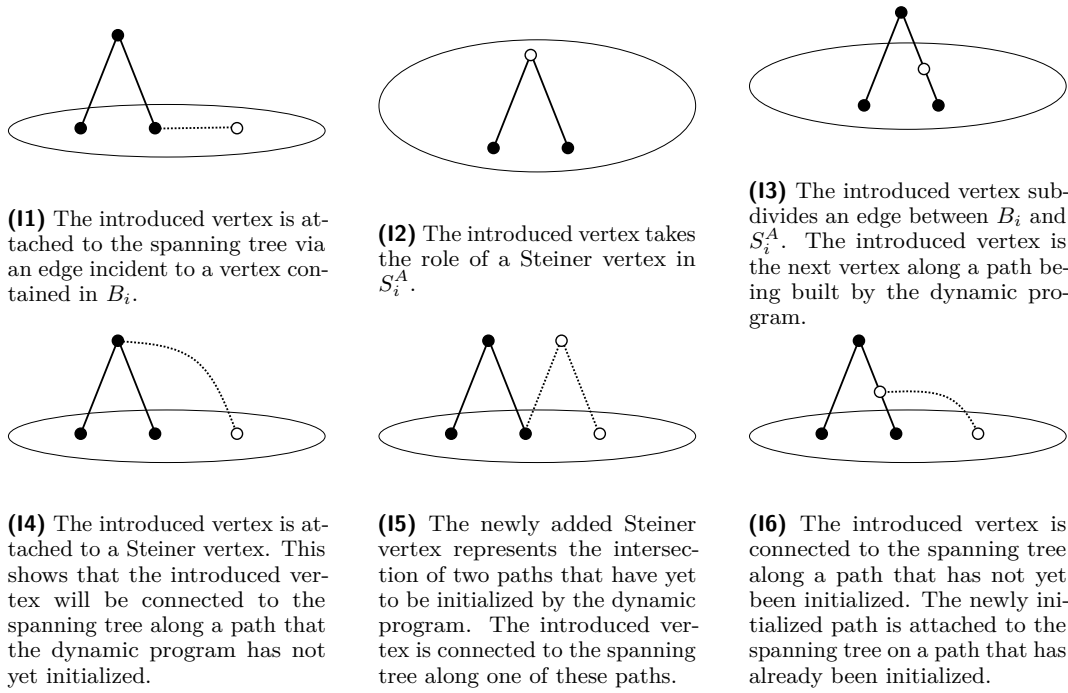
Each of these six constructions correspond to a possible way that v can be connected to the complete solution constructed by the dynamic program. See Figure 2 for an example of each case. In I1 v is directly connected to the partial solution at $\text{DP}_i[T_i, c_i]$ via some edge in $E(B_i)$. In I2 and I3 v can be thought of as the next vertex along the paths being built by the dynamic program. In I4, I5, and I6 v is connected to the complete solution via some path that has yet to be built by the dynamic program.

We now prove that these are the only six ways we can extend F_j to F_i while preserving the conformity.

► **Lemma 8.** *Let \mathcal{T} be a spanning tree of G conforming to a configuration (T_j, c_j) of the bag B_j . Let B_i be the parent of B_j introducing the vertex v . It follows that \mathcal{T} conforms to a configuration (T_i, c_i) of B_i if and only if (T_i, c_i) was constructed from (T_j, c_j) via I1 through I6.*

The value of a subproblem at an introduce node is given by

$$\text{DP}_i[T_i, c_i] = \min \left\{ \text{DP}_j[T_j, c_j] + \sum_{e \in I(v)} \text{stretch}_{T_i}(e) \right\} \quad (3)$$



■ **Figure 2** The six types of compatible configurations at an introduce node. The original tree consists of the black vertices and solid edges. The modifications are represented by the white vertices and dashed edges. The white vertex inside the circle is the vertex being introduced. The vertices enclosed in the circle are contained in the B_i and the vertices above the circle are contained in S_i^A .

where the minimum is taken over all compatible configurations of B_j . F_i is constructed from F_j and the inclusion of v . Since $D(B_i) = D(B_j) \cup \{v\}$ the inductive hypothesis implies that F_i spans $D(B_i)$. Finally, we show that the cost of F_i is minimum over all forests meeting (T_i, c_i) .

► **Lemma 9.** Fix a spanning tree \mathcal{T} of G and an introduce node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.

6.4 Forget nodes

When B_i is a forget node with child B_j we have $B_i = B_j \setminus \{v\}$ where v is the vertex being forgotten in B_i . Let (T_i, c_i) and (T_j, c_j) be configurations of B_i and B_j . We say (T_i, c_i) and (T_j, c_j) are compatible with one another if (T_i, c_i) can be constructed from (T_j, c_j) in the following way.

- F1. If v is a leaf construct T_i by contracting the edge incident to v . If this edge is incident to a Steiner vertex of degree 2 contract it as well.
- F2. If v is an internal vertex let $S \subseteq S_j^D$ be the set of Steiner vertices with degree 2 adjacent to v . Construct T_i by contracting each edge (v, s) for $s \in S$. Set $S_i^D := S_j^D \cup \{v\}$ and $c_i(v, s') := c_j(v, s) + c_j(s, b)$ where $b \in B_j$ is the other neighbor of s .

► **Lemma 10.** Let \mathcal{T} be a spanning tree of G conforming to a configuration (T_j, c_j) of the bag B_j . Let B_i be the parent of B_j forgetting the vertex v . It follows that \mathcal{T} conforms to a configuration (T_i, c_i) of B_i if and only if (T_i, c_i) was constructed from (T_j, c_j) via F1 or F2.

15:12 Low-Stretch Spanning Trees of Graphs with Bounded Width



■ **Figure 3** A pair of compatible configurations at a forget node. The figure on the left is the original tree, and the white vertex is the vertex being forgotten. The figure on the right is the result of the contraction.

The value of a subproblem at a forget node is given by the recurrence

$$\text{DP}_i[T_i, c_i] = \min \text{DP}_j[T_j, c_j] \quad (4)$$

where the minimum is taken over all (T_j, c_j) compatible with (T_i, c_i) . We set $F_i := F_j$ where F_j is the partial solution stored in the minimum $\text{DP}_j[T_j, c_j]$. Since $D(B_i) = D(B_j)$ it follows inductively that F_i spans $D(B_i)$. We now use the inductive hypothesis to prove that F_i is the minimum cost forest meeting (T_i, c_i) .

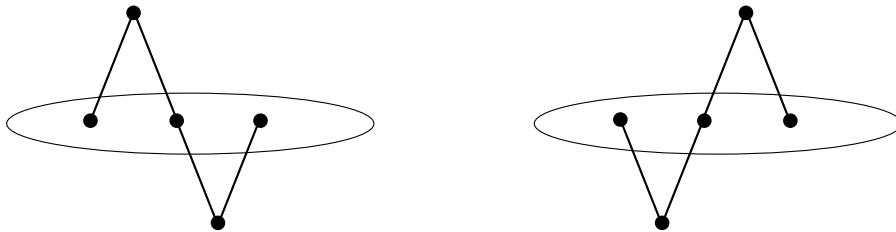
► **Lemma 11.** *Fix a spanning tree \mathcal{T} of G and a forget node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.*

6.5 Join nodes

When B_i is a join node with children B_j and B_k we have $B_i = B_j = B_k$. Given a configuration (T_i, c_i) of B_i we show how to build compatible configurations (T_j, c_j) and (T_k, c_k) of B_j and B_k . At a join node we decide which previously computed paths in the partial solutions at B_j and B_k to keep in the partial solution at B_i .

For a fixed configuration (T, c) of a bag B let \mathcal{S} be the set of maximal, connected, induced subgraphs of S^D . We *invert* a tree $S \in \mathcal{S}$ by setting $S^D := S^D \setminus S$ and $S^A := S^A \cup S$. If $(u, v) \in E(S)$ or (u, v) has $u \in S^D$ and $v \in B$ we add (u, v) to $E(S^A)$. Moreover, we do not change the value of $c(u, v)$. Inverting S does not change the structure of the tree it only changes the way we interpret the Steiner vertices in S .

We enumerate over the subsets S' of \mathcal{S} . In one child of B_i we invert S' and in the other we invert $\mathcal{S} \setminus S'$. For each configuration (T_i, c_i) of B_i and subset of trees $S' \subset \mathcal{S}$ we build a compatible triplet of configurations in the following way. Define T_j to be the tree constructed by inverting S' in T_i and T_k to be the tree constructed by inverting $\mathcal{S} \setminus S'$ in T_i . The cost functions c_j and c_k are inherited from c_i .



■ **Figure 4** A pair of compatible configurations corresponding to an inverted tree.

Fix a tree $S \in \mathcal{S}'$. The configuration (T_j, c_j) is anticipating the construction of a subtree isomorphic to S in order to connect the vertices in B_j . Similarly, the configuration (T_k, c_k) has already constructed a subtree isomorphic to S connecting the vertices in B_k . Since

$D(B_j) \setminus B_i$ and $D(B_k) \setminus B_i$ are disjoint we can safely merge the solutions to form the solution at the configuration (T_i, c_i) . Hence, the stretch of the partial solution at a join node is given by

$$\text{DP}_i[T_i, c_i] = \min \left\{ \text{DP}_j[T_j, c_j] + \text{DP}_k[T_k, c_k] - \sum_{e \in E(B_i)} \text{stretch}_{T_i}(e) \right\}. \quad (5)$$

The minimization is taken over all triplets of compatible configurations. We subtract

$$\sum_{e \in E(B_i)} \text{stretch}_{T_i}(e)$$

to prevent double counting the stretch of the edges in B_i since $B_i = B_j = B_k$. If F_j and F_k are the partial solutions at $\text{DP}_j[T_j, c_j]$ and $\text{DP}_k[T_k, c_k]$ then the result of the join node is $F_i := F_j \cup F_k$. By induction F_j spans $D(B_j)$ and F_k spans $D(B_k)$, hence F_i spans $D(B_i)$.

► **Lemma 12.** *Fix a spanning tree of \mathcal{T} of G and a join node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.*

6.6 Correctness

Let B_r be the root node of the tree decomposition of G . Without loss of generality we can assume that B_r is a forget node containing one vertex v_r . The only configuration on B_r is $(\{v_r\}, \emptyset)$ which is a single vertex. Since every spanning tree of G conforms to $(\{v_r\}, \emptyset)$ the solution indexed at $\text{DP}_r[\{v_r\}, \emptyset]$ must be a minimum stretch spanning tree of G . We analyze the runtime and provide complete proofs in the appendix.

References

- 1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 781–790, Washington, DC, USA, 2008. IEEE Computer Society. doi:10.1109/FOCS.2008.62.
- 2 Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 395–406, New York, NY, USA, 2012. ACM.
- 3 Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995. doi:10.1137/S0097539792224474.
- 4 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k -tree. *SIAM. J. on Algebraic and Discrete Methods*, 8(2):277–284, 1987.
- 5 Yair Bartal, Douglas E. Carroll, Adam Meyerson, and Ofer Neiman. Bandwidth and low dimensional embedding. *Theor. Comput. Sci.*, 500:44–56, 2013. doi:10.1016/j.tcs.2013.05.038.
- 6 Hans L. Bodlaender, Paul Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, pages 41–53, Cham, 2013. Springer International Publishing.
- 7 Bela Bollobas. *Modern Graph Theory*. Springer, 1998.
- 8 Douglas E. Carroll, Ashish Goel, and Adam Meyerson. Embedding bounded bandwidth graphs into ℓ_1 . In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part I*, ICALP'06, pages 27–37, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11786986_4.

- 9 A. C. Cassell, J. C. de C. Henderson, and A. Kaveh. Cycle bases for the flexibility analysis of structures. *International Journal for Numerical Methods in Engineering*, 8(3):521–528, 1974. doi:10.1002/nme.1620080308.
- 10 F. R. K. Chung. On the cutwidth and topological bandwidth of a tree. *SIAM Journal of Algebraic Discrete Methods*, 6:268–277, 1985.
- 11 Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, May 2008.
- 12 Yuval Emek. k -outerplanar graphs, planar duality, and low stretch spanning trees. *Algorithmica*, 61(1):141–160, September 2011.
- 13 Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, June 1994.
- 14 Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991.
- 15 F. Gavril. Some NP-complete problems on graphs. *11th Conference on Information Sciences and Systems*, pages 91–95, 1977.
- 16 Petra M. Gleiss. *Short Cycle*. PhD thesis, Universitat Wien, 2001.
- 17 Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and ℓ_1 -embeddings of graphs. *Combinatorica*, 24(2):233–269, April 2004.
- 18 Dov Harel. A linear time algorithm for the lowest common ancestors problem. *21st Annual Symposium on Foundations of Computer Science*, 1980.
- 19 Tom Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag, 1994.
- 20 Ekkehard Köhler, Rolf H. Möhring, and Gregor Wünsch. Minimizing total delay in fixed-time controlled traffic networks. In *OR*, pages 192–199, 2004.
- 21 I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving sdd linear systems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 235–244, 2010.
- 22 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly- $m \log n$ time solver for sdd linear systems. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.
- 23 James R. Lee and Anastasios Sidiropoulos. Pathwidth, trees, and random embeddings. *Combinatorica*, 33(3):349–374, June 2013. doi:10.1007/s00493-013-2685-8.
- 24 Christian Liebchen. Periodic timetable optimization in public transport. In Karl-Heinz Waldmann and Ulrike M. Stocker, editors, *Operations Research Proceedings 2006*, pages 29–36, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 25 Ch. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.
- 26 David Peleg and Eilon Reshef. Deterministic polylog approximation for minimum communication spanning trees. In *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 670–681. Springer, 1998.
- 27 Alexander Reich. *Cycle Bases of Graphs and Spanning Trees with Many Leaves*. PhD thesis, BTU Cottbus, 2014.

A Deterministic low-stretch spanning tree

In this section we show that given a graph G and a linear arrangement ϕ with bandwidth b Algorithm 1 produces a spanning tree of G with stretch $O(b^3)$. As stated in Section 3 such a spanning tree can be constructed in linear time. Throughout this section we denote the spanning tree produced by Algorithm 1 by T , and we denote the arrangement tree arising from ϕ by A .

We use a charging scheme to pay for the cycles in the fundamental basis created by our spanning tree algorithm. Each fundamental cycle with sufficiently large spread will be assigned a charge. Moreover, the sum of the charges is an upper bound on the sum of the lengths of the cycles.

We are now ready to define the key component to our charging scheme. Let x be a node in A . A *long component* of G_x is a connected component of G_x that includes at least one vertex within distance b of each endpoint in the linear arrangement of G_x . The number of long components in G_x will be denoted with ℓ_x .

► **Lemma 13.** *For any $x \in V(G)$ we have (1) $\ell_x \leq b$ and (2) if x is the parent of y then $\ell_x \leq \ell_y$.*

Proof. Since a long component is a special type of connected component a vertex can be in at most one long component. A long component contains at least one vertex from the first b vertices in the linear arrangement. This implies there can be at most b long components, hence $\ell_x \leq b$.

Let x be a node in A with left child y and right child z . Recall that $G_x = G_y \cup G_z \cup S_x$. The left endpoint of G_x is the left endpoint of G_y , and the right endpoint of G_x is the right endpoint of G_z . Any edge in S_x connects a vertex within the rightmost b vertices of G_y to a vertex within the leftmost b vertices of G_z . Therefore a long component in G_x must contain a long component in G_y , a long component in G_z , and an edge in S_x , thus $\ell_x \leq \ell_y$. ◀

Here we introduce a charging scheme that will be used to pay for the cycles added to our basis. For any node x in the arrangement tree A let n_x be the number of leaf descendants of x . If x is the parent of y and z such that $\ell_x < \ell_y$ and $\ell_x < \ell_z$ we assign a charge $c_x = n_y + n_z$ to x , if $\ell_x < \ell_y$ and $\ell_x = \ell_z$ we assign a charge $c_x = n_y$ to x , similarly if $\ell_z < \ell_x$ and $\ell_y = \ell_z$ we assign $c_x = n_z$, otherwise $c_x = 0$. Next, we show that the sum over all charges is $O(n)$.

► **Lemma 14.** *The sum of the charges is linear in the number of vertices in G . That is, $\sum_{x \in V(A)} c_x \leq bn$.*

Proof. Consider the set J of nodes with exactly j long components and non-zero charge. If $u, v \in J$ such that v is a descendent of u , then all nodes on the u to v path are in J since by Lemma 13 the number of long components is monotonic in depth. Let x be a node on this path, let z be its child on the path, and let y be its child off the path. If both x and y have j long components our charging scheme makes $c_y = 0$, therefore $y \notin J$ and c_x is the number of leaf descendants of y . Therefore, the sets of leaf descendants from which every node in J derives its charge are disjoint. Thus, $\sum_{x \in J} c_x \leq n$. By Lemma 13 the number of long components in any induced subgraph is at most b , therefore $\sum_{x \in V(A)} c_x \leq bn$. ◀

Recall that the spread of a fundamental cycle C is defined to be $\phi(v_\ell) - \phi(v_r)$ where v_ℓ and v_r are the left and right endpoints of C . In Lemma 15 we show that the spread of C is within a constant factor of its length. In Lemma 16 we show that C 's fundamental edge induces a charge that is within a constant factor of the spread of C . This justifies the use of our charging scheme.

► **Lemma 15.** *If C is a cycle with length $|C|$ and spread s , then we have the inequality $\frac{2s}{b} \leq |C| \leq s + 1$.*

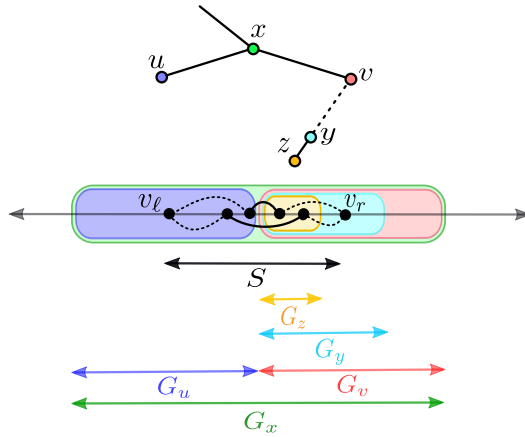
Proof. The upper bound is trivial. Conversely, decompose C into the two unique v_ℓ -to- v_r paths. Each edge in these paths has a spread of at most b in the linear arrangement, so each path needs at least $\frac{s}{b}$ edges. Therefore, $\frac{2s}{b} \leq |C|$. ◀

15:16 Low-Stretch Spanning Trees of Graphs with Bounded Width

Let C be a fundamental cycle of T with length $|C|$, spread $s \geq 4b$, and whose fundamental edge is in S_x . Since C 's fundamental edge is in S_x , G_x must be the first induced subgraph in the leaf-to-root ordering that contains C since every tree edge of C must be added to T before the fundamental edge is considered by Algorithm 1. Let $S = \{v \in V(G) \mid \phi(v_\ell) \leq \phi(v) \leq \phi(v_r)\}$ where v_ℓ and v_r are the left and right endpoints of C . Let u and v be the left and right child of x in A , respectively. We call $S \cap G_u$ the left half of S and $S \cap G_v$ the right half of S . Without loss of generality assume that $|S \cap G_v| \geq |S \cap G_u|$. Let y be the deepest descendant of x such that G_y contains the right half of S . Note that it may be the case that $y = v$. We call y the charging node of C . This is illustrated in Figure 5. In the following lemma we show that the existence of C implies that $c_y = \Theta(|C|)$. This is the charge that will pay for C in the cycle basis.

► **Lemma 16.** *Let C be a fundamental cycle of T as described above. It follows that C 's charging node y has $c_y > 0$ and y 's left child z contributes n_z to its charge. Moreover, $\frac{1}{4}(|C| - 1) \leq c_y \leq b \cdot |C|$.*

Proof. Consider the two unique v_ℓ -to- v_r paths, P_1 and P_2 , in C . Since there are at least b vertices in G_z there must be edges $e_1 \in E(P_1)$ and $e_2 \in E(P_2)$ connecting G_u to G_z . One of these edges belongs to T , and the other is the fundamental edge of C . The right endpoints of e_1 and e_2 must belong to long components of G_z since they belong to P_1 and P_2 which extend to v_r . Moreover, these long components are distinct. For otherwise, C 's right endpoint would be in G_z , contradicting our choice of y . By the existence of P_1 and P_2 , these long components are merged in G_y . Since y is the parent of z with $\ell_y < \ell_z$, we have $c_y \geq n_z$. We also have that $c_y \leq n_y = 2n_z$. Further, by our choice of y as the deepest descendant, $n_z \leq s \leq 4n_z$. Combining these inequalities with those of Lemma 15 yields $\frac{1}{4}(|C| - 1) \leq c_y \leq b \cdot |C|$. ◀



■ **Figure 5** An illustration of the conditions of Lemma 16. The colored region encloses the linear arrangement of G_x , and the partitions represent the subgraphs induced by the descendants of x . The dotted lines represent the paths P_1 and P_2 . The solid lines represent the edges that induce the charge c_y .

We are now ready to prove the main theorem of the section.

► **Theorem 17.** *The spanning tree T of G produced by Algorithm 1 has $FCB(T) \leq 4b^3n$.*

Proof. There are at most $\frac{1}{2}(b-1)(b-2)$ edges in S_x by (2). By Lemma 16, sum of the lengths of all of the fundamental cycles with spread at least $4b$ is at most

$$\begin{aligned} \sum_{y \in V(A)} \frac{1}{2}(b-1)(b-2)(4c_y + 1) &\leq n + 2(b-1)(b-2) \sum_{y \in V(A)} c_y \\ &\leq n + 2(b-1)(b-2)bn \\ &\leq 3b^3n \end{aligned}$$

Where the first and second inequalities come from Lemmas 16 and 14, respectively. All fundamental cycles with spread at most $4b$ have their non-tree edges a node of A of height at most $\log 4b$. Therefore, there are at most n nodes in A with $|V(G_x)| \leq 4b$ that contain a cycle. These contribute at most $\frac{1}{2}(b-1)(b-2)n$ to the sum of the lengths of the fundamental cycles. In total we have

$$FCB(T) \leq 3b^3n + b^2n \leq 4b^3n$$

as desired. ◀

► **Corollary 18.** *The tree T produced by our spanning tree algorithm has $stretch(T) \leq 4b^3 + 2$.*

Proof. According to (1), the weight of the fundamental cycle basis and the minimum stretch spanning tree are related by

$$stretch(T) = \frac{1}{m}(FCB(T) - m + 2n + 2).$$

The result follows immediately from the fact that $n \leq m \leq bn$. ◀

B Details for the bounded treewidth dynamic program

► **Lemma 7.** *Let \mathcal{T} be a spanning tree of G . There is a configuration (T, c) at bag B that \mathcal{T} conforms to such that $|V(T)| = O(k)$.*

Proof. Let (T, c) be the configuration obtained by applying the contraction rules to \mathcal{T} . Every vertex $v \in V(T) \setminus B$ is an internal vertex of T , otherwise its incident edge is not contained in a path connecting a pair of vertices from B and should have been contracted. Further, any vertex of $V(T) \setminus B$ with degree 2 in T is adjacent to two vertices of B . Therefore, T is a tree with at most $k + 1$ leaves and $k + 1$ vertices of degree 2. It follows that $|V(T)| = O(k)$. ◀

► **Lemma 8.** *Let \mathcal{T} be a spanning tree of G conforming to a configuration (T_j, c_j) of the bag B_j . Let B_i be the parent of B_j introducing the vertex v . It follows that \mathcal{T} conforms to a configuration (T_i, c_i) of B_i if and only if (T_i, c_i) was constructed from (T_j, c_j) via I1 through I6.*

Proof. If (T_i, c_i) was constructed from (T_j, c_j) from one of the six methods described in the preceding subsection then either T_i and T_j are isomorphic (I2) and \mathcal{T} conforms to (T_i, c_i) or T_i differs from T_j by the inclusion of v , or the inclusion of v and some Steiner vertex. In I1, I4, I5, and I6 we have $\deg(v) = 1$ and v is either adjacent to another vertex in B_i , a Steiner vertex with degree 2 whose second neighbor is in B_i , or a Steiner vertex of degree of degree at least 3. In each of these cases \mathcal{T} conforms to (T_i, c_i) . In I3 v has degree two and is adjacent to a vertex in the bag and some Steiner vertex. This case is equivalent to

15:18 Low-Stretch Spanning Trees of Graphs with Bounded Width

subdividing the edge incident to the Steiner vertex to make v , hence the Steiner vertex still meets the conforming criteria.

Conversely, assume \mathcal{T} conforms to (T_i, c_i) . If v is a leaf in T_i then it is connected to some other vertex in B_i along some path consisting of zero or more Steiner vertices. Since \mathcal{T} conforms to (T_j, c_j) this path must have been contracted in T_j . Hence, to build T_i we need to undo the contraction. This corresponds to I1, I4, I5, and I6. If v is an internal vertex in T_i with $\deg(v) = 2$ with one neighbor in S_i^A then v must have been contracted when building T_j . In this case T_i is built by undoing the contraction which corresponds to I3. If v is any other internal vertex in T_i then it is contained in some path whose endpoints are in B_j . Moreover, its neighbors must also be contained in such a path otherwise they would have been contracted. It follows that T_i is isomorphic to T_j which corresponds to I1 where the only change is a relabeling of the vertices. \blacktriangleleft

► **Lemma 9.** *Fix a spanning tree \mathcal{T} of G and an introduce node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.*

Proof. When B_i is an introduce node we have $B_i = B_j \cup \{v\}$ where B_j is the child of B_i . Let (T_j, c_j) be a configuration of B_j that is compatible with (T_i, c_i) . We need to show that if \mathcal{T} conforms to (T_i, c_i) then \mathcal{T} also conforms to (T_j, c_j) . Since (T_i, c_i) and (T_j, c_j) are compatible T_i differs from T_j by at most the inclusion of v and possibly a Steiner vertex s adjacent to v . By contracting the newly added edges incident to s and v we see that \mathcal{T} conforms to (T_j, c_j) . By the inductive hypothesis we have $\text{DP}_j[T_j, c_j] \leq \sum_{e \in G[D(B_j)]} \text{stretch}_{\mathcal{T}}(e)$. Since $D(B_i) = D(B_j) \cup \{v\}$ it follows that

$$\text{DP}_i[T_i, c_i] \leq \text{DP}_j[T_j, c_j] + \sum_{e \in I(v) \cap B_j} \text{stretch}_{\mathcal{T}}(e) \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e). \quad \blacktriangleleft$$

► **Lemma 10.** *Let \mathcal{T} be a spanning tree of G conforming to a configuration (T_j, c_j) of the bag B_j . Let B_i be the parent of B_j forgetting the vertex v . It follows that \mathcal{T} conforms to a configuration (T_i, c_i) of B_i if and only if (T_i, c_i) was constructed from (T_j, c_j) via F1 or F2.*

Proof. Assume \mathcal{T} conforms to (T_i, c_i) . Since \mathcal{T} conforms to (T_j, c_j) and $B_j \setminus B_i = \{v\}$ it follows that T_i differs from T_j by the contraction of edges incident to v . These edges are the edges contracted by rules F1 and F2. Conversely, assume (T_i, c_i) was constructed from (T_j, c_j) by either F1 or F2. Since F1 and F2 apply the contraction rules for conformity on the edges incident to v it follows that \mathcal{T} conforms to (T_i, c_i) . \blacktriangleleft

► **Lemma 11.** *Fix a spanning tree \mathcal{T} of G and a forget node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.*

Proof. When B_i is a forget node we have $B_i = B_j \setminus \{v\}$ where B_j is the child of B_i , hence $D(B_i) = D(B_j)$. If \mathcal{T} conforms to (T_i, c_i) then \mathcal{T} conforms to some configuration (T_j, c_j) of B_j . The configuration (T_j, c_j) can be found by undoing the contractions made by F1 and F2 and choosing the minimum such $\text{DP}_j[T_j, c_j]$. It follows that (T_i, c_i) and (T_j, c_j) are compatible, hence $\text{DP}_i[T_i, c_i] = \text{DP}_j[T_j, c_j]$. Applying the inductive hypothesis $\text{DP}_j[T_j, c_j] \leq \sum_{e \in G[D(B_j)]} \text{stretch}_{\mathcal{T}}(e)$ proves the claim. \blacktriangleleft

► **Lemma 12.** *Fix a spanning tree of \mathcal{T} of G and a join node B_i with configuration (T_i, c_i) . If \mathcal{T} conforms to (T_i, c_i) then $\text{DP}_i[T_i, c_i] \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$.*

Proof. Let B_i be a join node with children B_j and B_k with configurations (T_j, c_j) and (T_k, c_k) . When (T_i, c_i) , (T_j, c_j) , and (T_k, c_k) are compatible with each other the trees T_i , T_j , and T_k are isomorphic since they only differ by the labeling of the Steiner vertices. Hence,

if \mathcal{T} conforms to (T_i, c_i) also conforms to (T_j, c_j) and (T_k, c_k) . By the inductive hypothesis we have $\text{DP}_j[T_j, c_j] \leq \sum_{e \in G[D(B_j)]} \text{stretch}_{\mathcal{T}}(e)$ and $\text{DP}_k[T_k, c_k] \leq \sum_{e \in G[D(B_k)]} \text{stretch}_{\mathcal{T}}(e)$. From the equality

$$\sum_{e \in G[D(B_k)]} \text{stretch}_{\mathcal{T}}(e) + \sum_{e \in G[D(B_k)]} \text{stretch}_{\mathcal{T}}(e) - \sum_{e \in G[B_i]} \text{stretch}_{\mathcal{T}}(e) = \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e)$$

it follows that

$$\text{DP}_i[T_i, c_i] \leq \text{DP}_j[T_j, c_j] + \text{DP}_k[T_k, c_k] - \sum_{e \in G[B_i]} \text{stretch}_{\mathcal{T}}(e) \leq \sum_{e \in G[D(B_i)]} \text{stretch}_{\mathcal{T}}(e),$$

which proves the theorem. \blacktriangleleft

B.1 Runtime analysis for Section 6

In this section we analyze the runtime of our dynamic program on a graph G with treewidth k . We begin by analyzing the size of the three dimensional array $\text{DP}_i[T, c]$. The subscript i represents a bag in the nice tree decomposition of G . It is known that a graph with n vertices has a nice tree decomposition of width k with at most $4n$ bags [19]. It follows from Lemma 7 that any tree T used as an index in our array has at most $2k$ vertices. By Cayley's formula there are at most $(2k)^{2k-2} = O(2^{2k} k^{2k})$ such trees that will ever be built as an index by our dynamic program. The cost function c has domain $E(T)$ which has size k . The range of c is $\{1, \dots, n\}$ since the value of the cost function is only ever incremented by one when an edge is contracted. Hence the total number of possible cost functions is n^k . We conclude that the total size of our dynamic programming table is $O(2^{2k} k^{2k} n^{k+1})$.

Next we analyze the complexity of filling in the entries of our dynamic programming table. We will need to analyze introduce, forget, and join nodes as separate cases. In each case we find the compatible configurations of the child nodes by undoing the operations described in the previous section.

At an introduce node B_i we compute the value of $\text{DP}_i[T_i, c_i]$ by undoing the six methods used to build a pair of compatible configurations. For each $v \in V(T_i) \cap B_i$ we transform (T_i, c_i) into (T_j, c_j) by reversing the methods described in the introduce nodes section with v being treated as the vertex introduced to B_i . When v is a leaf in T_i or an internal vertex with one neighbor in B_i this is done by contracting the added edges. Otherwise, we take $(T_j, c_j) := (T_i, c_i)$. Hence, equation 3 takes the minimum over $O(k)$ compatible configurations.

When B_i is a forget node forgetting a vertex v there are two methods for finding compatible configurations of (T_i, c_i) . In the case that v was a leaf in T_j we attach v to each of the $O(k)$ vertices in $V(T_i) \cap B_i$ to construct each possible compatible configuration (T_j, c_j) . We have to consider the two cases where v is adjacent to the vertex in $V(T) \cap B_j$ and where there exists one intermediate Steiner vertex of degree 2 in between them. In the case that v was an internal vertex we consider each of the $O(k)$ Steiner vertices in S_i^D that are adjacent to some vertex in $V(T_i) \cap B_i$ via some edge of cost 1. To undo the operation we subdivide each of its incident edges with cost greater than 1 whose endpoint is in B_i . It follows that equation 4 takes its minimum over $O(k)$ compatible configurations.

When B_i is a join node there is a pair of compatible configurations for each of the $O(2^k)$ subsets of S_i^D . It follows that equation 5 takes the minimum over $O(2^k)$ values. Computing the entries of $\text{DP}_i[T, c]$ is dominated by the time it takes to compute the value at join nodes. We have now proven the main theorem of the section.

Parameterized Complexity of Two-Interval Pattern Problem

Prosenjit Bose

School of Computer Science, Carleton University, Ottawa, Canada
jit@scs.carleton.ca

Saeed Mehrabi

School of Computer Science, Carleton University, Ottawa, Canada
saeed.mehrabi@carleton.ca

Debajyoti Mondal

Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
d.mondal@usask.ca

Abstract

A *2-interval* is the union of two disjoint intervals on the real line. Two 2-intervals D_1 and D_2 are *disjoint* if their intersection is empty (i.e., no interval of D_1 intersects any interval of D_2). There can be three different relations between two disjoint 2-intervals; namely, preceding ($<$), nested (\sqsubset) and crossing (\bowtie). Two 2-intervals D_1 and D_2 are called *R-comparable* for some $R \in \{<, \sqsubset, \bowtie\}$, if either D_1RD_2 or D_2RD_1 . A set \mathcal{D} of disjoint 2-intervals is *R-comparable*, for some $\mathcal{R} \subseteq \{<, \sqsubset, \bowtie\}$ and $\mathcal{R} \neq \emptyset$, if every pair of 2-intervals in \mathcal{R} are *R-comparable* for some $R \in \mathcal{R}$. Given a set of 2-intervals and some $\mathcal{R} \subseteq \{<, \sqsubset, \bowtie\}$, the objective of the *2-interval pattern problem* is to find a largest subset of 2-intervals that is *R-comparable*.

The 2-interval pattern problem is known to be $W[1]$ -hard when $|\mathcal{R}| = 3$ and NP -hard when $|\mathcal{R}| = 2$ (except for $\mathcal{R} = \{<, \sqsubset\}$, which is solvable in quadratic time). In this paper, we fully settle the parameterized complexity of the problem by showing that it is $W[1]$ -hard for both $\mathcal{R} = \{\sqsubset, \bowtie\}$ and $\mathcal{R} = \{<, \bowtie\}$ (when parameterized by the size of an optimal solution). This answers the open question posed by Vialette [Encyclopedia of Algorithms, 2008].

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases Interval graphs, Two-interval pattern problem, Comparability, Multicoloured clique problem, Parameterized complexity, $W[1]$ -hardness

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.16

Funding The work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1 Introduction

Interval graphs and their generalizations are often used to study problems in resource allocation, scheduling, and DNA mapping. In 2002, Vialette [13] proposed a geometric description of RNA helices in an attempt to improve the understanding of the computational complexity for finding structured patterns in RNA sequences. In particular, Vialette modeled the RNA secondary structure using a set of 2-intervals, which inspired subsequent research (e.g., see [15]) on examining the properties of the geometric graphs arising from such representations. Vialette [14] introduced the *2-interval pattern problem*, which is now a widely-studied problem and the main topic of this paper.

A *2-interval* is the union of two disjoint intervals on the real line. Two 2-intervals D_1 and D_2 are *disjoint* if their intersection is empty; that is, no interval of D_1 intersects any interval of D_2 . We can define three different relations between two disjoint 2-intervals: one



© Prosenjit Bose, Saeed Mehrabi, and Debajyoti Mondal;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

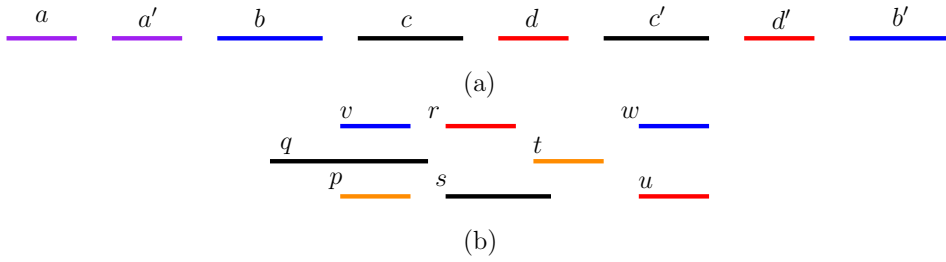
Editor: Susanne Albers; Article No. 16; pp. 16:1–16:10



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

16:2 Two-Interval Pattern Problem



■ **Figure 1** (a) An example for showing the three possible relations between a pair of 2-intervals; here, the same-colour intervals form a 2-interval. Then, $(a, a') < (d, d')$, $(d, d') \sqsubset (b, b')$ and $(c, c') \bowtie (d, d')$. (b) An instance of 2-interval pattern problem with $\mathcal{R} = \{<, \bowtie\}$ and the 2-intervals are $\{(u, w), (q, s), (r, u), (p, t)\}$. The 2-intervals $\{(p, t), (r, u)\}$ form a largest subset that is \mathcal{R} -comparable.

2-interval lies entirely to the left of the other one (called *preceding* and denoted by $<$), one 2-interval is nested within the other one (called *nested* and denoted by \sqsubset), and the intervals of the two 2-intervals alternate on the real line (called *crossing* and denoted by \bowtie). See Figure 1(a) for an example; a formal definition is given in Section 2. Two 2-intervals D_1 and D_2 are R -comparable for some $R \in \{<, \sqsubset, \bowtie\}$ if either $D_1 R D_2$ or $D_2 R D_1$. A set \mathcal{D} of disjoint 2-intervals is \mathcal{R} -comparable, for some $\mathcal{R} \subseteq \{<, \sqsubset, \bowtie\}$ and $\mathcal{R} \neq \emptyset$, if every pair of 2-intervals in \mathcal{R} are R -comparable for some $R \in \mathcal{R}$. In the *2-interval pattern problem*, we are given a set of 2-intervals and a set $\mathcal{R} \subseteq \{<, \sqsubset, \bowtie\}$, and the objective is to compute a largest subset of 2-intervals that is \mathcal{R} -comparable. Figure 1(b) illustrates such an example.

The 2-interval pattern problem can model various scenarios in the context of RNA structure prediction. While looking for certain RNA structures, some common approaches to cope with intractability are either to restrict the class of pseudoknots [11] or to apply heuristics [4, 10, 12]. Vialette [14] proposed that one can obtain a relevant set of 2-intervals from an RNA sequence by selecting stable stems, e.g., using a simplified thermodynamic model without accounting for loop energy [12, 14, 16]. Then, the prediction of the RNA structure is equivalent to finding a maximum subset of non-conflicting (i.e., disjoint) 2-intervals.

Related work. Vialette [14] observed that if $|\mathcal{R}| = 1$, then the 2-interval pattern problem is polynomial-time solvable by reductions to the maximum independent set problem on interval graphs, or to the maximum clique problem on comparability graphs. The running time of these algorithms have been improved since then, and expressed in terms of the number of input 2-intervals and various interval-related parameters such as their lengths or overlap [3]. For the case when $|\mathcal{R}| = 2$, the problem is solvable in polynomial time when $\mathcal{R} = \{<, \sqsubset\}$ [14]. However, the problem is known to be NP-hard if $\mathcal{R} = \{<, \bowtie\}$ or $\mathcal{R} = \{\sqsubset, \bowtie\}$, even if the intervals of every 2-interval have unit length [14, 2]. If $|\mathcal{R}| = 3$ (i.e., $\mathcal{R} = \{<, \sqsubset, \bowtie\}$), then the problem is APX-hard [1].

The approximability of the NP-hard models of the 2-interval pattern problem was studied by Crochemore et al. [5]. They gave polynomial-time algorithms for the problem with approximation factors 4 when $\mathcal{R} = \{<, \sqsubset, \bowtie\}$ or $\mathcal{R} = \{\sqsubset, \bowtie\}$, and 6 when $\mathcal{R} = \{<, \sqsubset\}$. They also showed that the results hold for the weighted case; i.e., when each 2-interval is associated with a weight and the goal is to find a maximum weight subset. For the weighted version and when $\mathcal{R} = \{<, \sqsubset, \bowtie\}$, a 2-approximation algorithm was given by Bar-Yehuda et al. [1]. These factors are improved to 3 when the intervals of every input 2-interval have unit length [5], where they also considered the case when the 2-intervals are weighted. For $\mathcal{R} = \{<, \bowtie\}$ (and arbitrary input 2-intervals), Jiang [7] improved the approximation factor to 2 and subsequently to $1 + \epsilon$ for any $\epsilon > 0$ [8].

The problem is $W[1]$ -hard when $\mathcal{R} = \{<, \sqsubset, \checkmark\}$, because in this case, the problem is equivalent to computing a maximum independent set on 2-interval graphs, and the latter is known to be $W[1]$ -hard [6]; see Section 3 for more details. For $|\mathcal{R}| = 2$, to the best of our knowledge, the only parameterized algorithm is the work of Blin et al. [2] who proved that the problem is fixed-parameter tractable, but only when $\mathcal{R} = \{\sqsubset, \checkmark\}$, the input intervals all have unit length and the tractability is with respect to the *forward crossing number*: the maximum number of 2-intervals that cross a 2-interval “from the right”.

Our results. In this paper, we answer a question of Vialette [15] by proving that the 2-interval pattern problem is $W[1]$ -hard for $\mathcal{R} = \{\sqsubset, \checkmark\}$ and $\mathcal{R} = \{<, \checkmark\}$ when parameterized by the size of an optimal solution. Our $W[1]$ -hardness result is inspired by the reduction of the k -independent set problem used by Fellows et al. [6]. Their reduction requires all three relations (i.e., they prove the $W[1]$ -hardness of the 2-interval pattern problem when $\mathcal{R} = \{<, \checkmark, \sqsubset\}$). Prior to our work, it was known that the complexity of the problem is polynomial when $\mathcal{R} = \{<, \sqsubset\}$ [3], but it was unknown whether the problem is fixed-parameter tractable (when parameterized by the size of an optimal solution) or it is $W[1]$ -hard for $\mathcal{R} = \{\sqsubset, \checkmark\}$ and $\mathcal{R} = \{<, \checkmark\}$. Hence, our $W[1]$ -hardness result fully settles the parameterized complexity of the 2-interval pattern problem.

2 Preliminaries

In this section, we give some definitions and notation that will be used throughout the paper.

A 2-interval D is the union of two disjoint intervals on the real line; that is, $D = (A, B)$ and the interval A lies to the left of the interval B . For a pair of disjoint intervals I, J , we write $I < J$ when I is to the left of J . Two 2-intervals D_i and D_j are *disjoint* if $(I_i \cup J_i) \cap (I_j \cup J_j) = \emptyset$. Moreover, for two disjoint 2-intervals D_i and D_j , we say that D_i is *preceding* (resp., *nested in*, *crossing*) D_j if $I_i < J_i < I_j < J_j$ (resp., $I_j < I_i < J_i < J_j$, $I_i < I_j < J_i < J_j$). We write $D_i < D_j$ (resp., $D_i \sqsubset D_j$, $D_i \checkmark D_j$) when D_i is preceding (resp., nested in, crossing) D_j .

We say that two 2-intervals D_i and D_j are R -comparable, for some $R \in \{<, \sqsubset, \checkmark\}$, if (i) D_i and D_j are disjoint and (ii) either $D_i R D_j$ or $D_j R D_i$. Let S be a set of n 2-intervals on the real line, and let $\mathcal{R} \subseteq \{<, \sqsubset, \checkmark\}$ such that $\mathcal{R} \neq \emptyset$. Then, a set $\mathcal{D} \subseteq S$ is called \mathcal{R} -comparable if every pair of 2-intervals in \mathcal{D} are R -comparable for some $R \in \mathcal{R}$. Given S and some $\mathcal{R} \subseteq \{<, \sqsubset, \checkmark\}$, the objective of the 2-interval pattern problem is to compute a largest subset $\mathcal{D} \subseteq S$ such that \mathcal{D} is \mathcal{R} -comparable.

Given a graph G and a parameter k , the k -independent set problem asks whether there is an independent set of size k in G . Fellows et al. [6] proved that the k -independent set problem is $W[1]$ -hard on 2-interval graphs when $\mathcal{R} = \{<, \checkmark, \sqsubset\}$. A 2-interval graph is the intersection graph of a set of 2-intervals on the real line. Our $W[1]$ -hardness results are also based on showing reductions from the k -Multicoloured Clique Problem, which is known to be $W[1]$ -hard [6]. The problem is defined as follows.

Problem: k -Multicoloured Clique.

Input: A graph G , and a vertex-colouring $c : V(G) \rightarrow \{1, 2, \dots, k\}$ for G .

Question: Is there a clique of size k in G such that, for each $c \in \{1, 2, \dots, k\}$, there is exactly one vertex in the clique that has colour c ?

3 W[1]-Hardness

In this section, we prove that the 2-interval pattern problem is W[1]-hard when $\mathcal{R} = \{\sqsubset, \emptyset\}$ and $\mathcal{R} = \{<, \emptyset\}$. Our reduction is inspired by that of Fellows et al. [6]. Let (G, c, k) be an instance of the k -multicoloured clique problem (we assume w.l.o.g. for our purposes that c is a proper colouring¹). We construct a set \mathcal{F} of 2-intervals such that G has a multicoloured clique of size k if and only if \mathcal{F} contains a set of $k' = 2k + 4\binom{k}{2}$ disjoint 2-intervals that are pairwise comparable in one of the relations in \mathcal{R} ; the value of k' will be clear from our construction. We first describe an outline of the construction and the corresponding gadgets. Then, we give the details on how to organize the gadgets on the real line specific to each of the sets $\mathcal{R} = \{\sqsubset, \emptyset\}$ and $\mathcal{R} = \{<, \emptyset\}$. For a colour $i \in \{1, 2, \dots, k\}$, let $V_i(G)$ denote the set of vertices of G that have colour i . Moreover, for every distinct pair of colours i, j , let $E_{(i,j)}(G)$ denote the set of edges (u, v) of G such that $\{c(u), c(v)\} = \{i, j\}$. That is, $E_{(i,j)}(G)$ consists of all the edges whose end vertices are coloured with two distinct colours i and j .

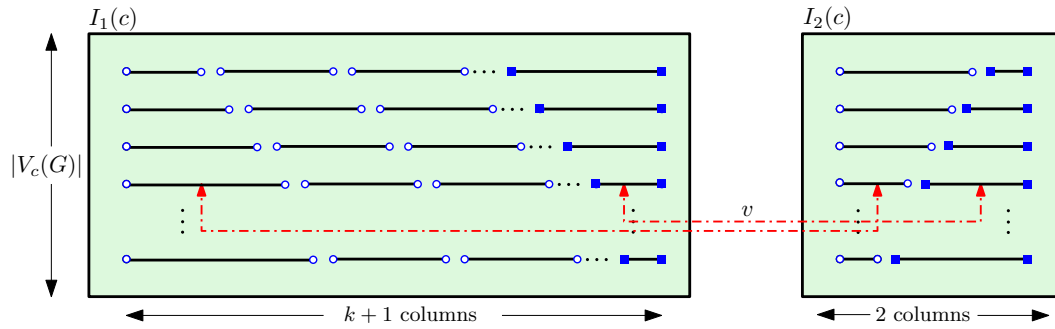
Outline. The construction consists of two main types of gadgets: *selection* and *validation*. By selection gadgets, we ensure that 2-intervals representing k vertices with distinct colours and $\binom{k}{2}$ edges with distinct pairs of colours are selected. By validation gadgets, we ensure that the selected set of 2-intervals are valid in the sense that the k selected vertices are actually adjacent in the graph and the selected edges are indeed over the selected set of vertices. We group the 2-intervals corresponding to vertices of the same colour together in a *vertex-selection* gadget in such a way that any feasible solution for the 2-interval pattern problem will have 2-intervals corresponding to one vertex per vertex-selection gadget. Similarly, we group the 2-intervals corresponding to edges with the same pairs of distinct colours $\{i, j\}$ together in a *edge-selection* gadget such that any feasible solution for the 2-interval pattern problem will have 2-intervals corresponding to one edge (u, v) with $\{c(u), c(v)\} = \{i, j\}$. We will then organize the gadgets on the real line in such a way that any feasible solution will contain 2-intervals that are \mathcal{R} -comparable.

Given (G, c, k) , we associate one 2-interval I_v for each vertex $v \in V(G)$. Moreover, we associate four 2-intervals for each edge $(u, v) \in E(G)$: two 2-intervals $I_{(u,v)}$ and $I_{(v,u)}$ for each “direction” of the edge and two 2-intervals $I_{\{u,v\}}$ and $I'_{\{u,v\}}$ that are undirected. The 2-intervals for “directed” edges will be used for validation, and we will show below how they are constructed. Therefore, the number of 2-intervals of the constructed instance will be $|V(G)| + 4|E(G)|$. We next give the details of each type of gadgets.

Vertex-selection gadget. For each colour $c \in \{1, 2, \dots, k\}$, we construct a vertex-selection gadget. The gadget has two components, which we denote by $I_1(c)$ and $I_2(c)$; see Figure 2 for an illustration. The component $I_1(c)$ has $|V_c(G)|$ “rows” of intervals, each of which has $(k + 1)$ “columns”; each row corresponds to a vertex of G with colour c . The intervals in the same column pairwise intersect. Moreover, for the intervals in a fixed column j , we assign an offset such that each interval in row $i > 1$ intersects the interval that is in column $j + 1$ and row $i - 1$; see Figure 2. The component $I_2(c)$ consists of two columns of intervals, and each column has $|V_c(G)|$ rows. Here, we assign an offset such that the interval in the first column and row i intersects the interval in the second column and row $i + 1$ (see Figure 2).

For each vertex $v \in V_c(G)$, we associate two 2-intervals I_v and I'_v as follows. The first (resp., second) 2-interval I_v (resp., I'_v) is composed of the interval in the first (resp., last) column of $I_1(c)$ that corresponds to v and the interval in the first (resp., second) column of

¹ Otherwise, one can remove the edges whose end vertices are coloured with the same color.



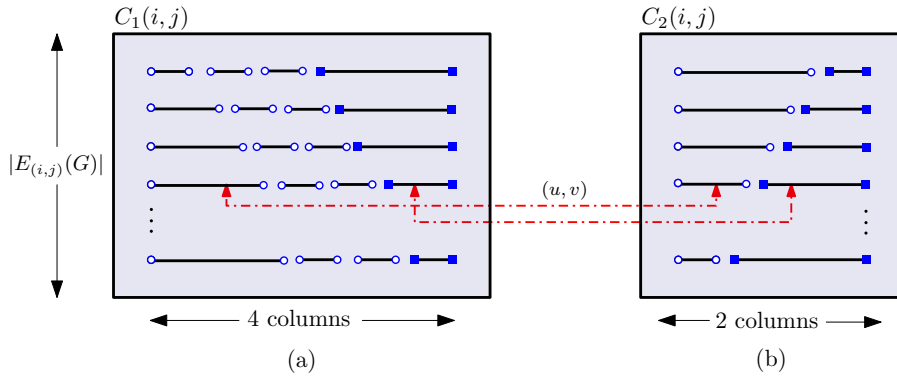
■ **Figure 2** A vertex-selection gadget and the two 2-intervals I_v and I'_v corresponding to a vertex v .

$I_2(c)$ that corresponds to v . These 2-intervals are illustrated with dashed lines in Figure 2. Each of the remaining k columns in $I_1(c)$ corresponds to a colour in $\{1, 2, \dots, k\} \setminus \{c\}$. These $|V_c(G)| \times (k - 1)$ intervals are later paired with intervals from edge-selection gadgets to form 2-intervals that correspond to “directed” edges. Notice that the intervals of the first column of $I_1(c)$ pairwise intersect, ensuring that at most one 2-interval corresponding to a vertex with colour c can appear in any feasible solution. Similarly, for the $|V_c(G)| \times (k - 1)$ intermediate intervals of $I_1(c)$ (i.e., the intervals of $I_1(c)$ excluding those in the first and last column), it means that all the edges of a k -multicoloured clique with at least one endpoint with colour c are incident to the same vertex in $V_c(G)$.

► **Lemma 1.** *Let S be feasible a solution for the 2-interval pattern problem, and consider the vertex-selection gadget T corresponding to colour c . Moreover, let $M \subseteq S$ be the set of 2-intervals such that each 2-interval in M has at least one interval in T . If $|M| \geq k + 1$, then all the intervals in $M \cap T$ are selected from the same row of T .*

Proof. Since there are $(k + 1)$ columns in the component $I_1(c)$ of T , M cannot have more than $(k + 1)$ 2-intervals, where each containing at least one interval from T . Hence, $|M| = k + 1$. This means that M must contain exactly one interval from every column of $I_1(c)$ and hence, one from every column of $I_2(c)$. Consider the interval in the first column of $I_1(c)$ (that is in M) and assume that this interval is in row i , for some $1 \leq i \leq |V_c(G)|$; it corresponds to a vertex $v \in V_c(G)$. We now show that every other interval in $M \cap T$ must also be in row i . Since $M \subseteq S$ and S is a feasible solution, then the interval in the first column of $I_2(c)$ and row i must also be in M because these two intervals form one of the two 2-intervals corresponding to v . Now, suppose that the interval in M that is from the second column of $I_2(c)$ is in row i' . Clearly, $i' \leq i$ (i.e., i' lies below i) because otherwise the interval of M that is in the first column of $I_2(c)$ would intersect this interval due to the offset. Since $M \subseteq S$ and S is a feasible solution, M must contain the interval in the last column of $I_1(c)$ that is in row i' (as only these two would form a valid 2-interval while considering $I_2(c)$). If $i' < i$, then it is not possible to have exactly one interval from column j of $I_1(c)$ in M for all $j = 2, 3, \dots, k$ because the offset would imply that at least two intervals must intersect in M . Therefore, $i' = i$. In the same way, we can show that the subsequent intervals of $M \cap T$ must also be in row i . ◀

Observe that the assignment of two 2-intervals for each vertex $v \in V_c(G)$ and placement of their second intervals in $I_2(c)$ with an offset allowed us to argue that the remaining intervals are also selected from the same row of the vertex-selection gadget. We will use a similar construction to argue the same for edge-selection gadgets. Before we continue, one might

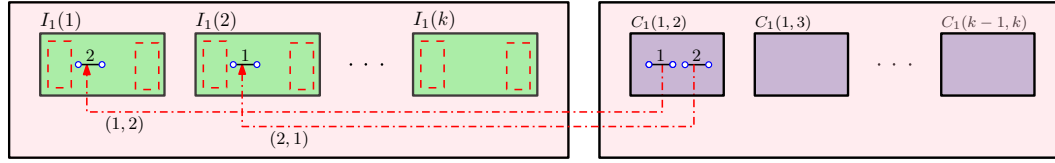


■ **Figure 3** An illustration of the two components of an edge-selection gadget; namely, (a) $C_1(i, j)$ and (b) $C_2(i, j)$. The two 2-intervals $I_{\{u,v\}}$ and $I'_{\{u,v\}}$ corresponding to the edge $(u, v) \in E(G)$ are shown dashed-dotted red.

wonder why we needed $I_2(c)$ and why could not we have only $I_1(c)$ with one 2-interval for each vertex. Although this would force the selection of remaining intervals from the same row, it is impossible to place such a gadget on the real line while maintaining \mathcal{R} -comparability. To ensure \mathcal{R} -comparability, we will need to place $I_1(c)$ and $I_2(c)$ on different parts of the real line, possibly far apart from each other.

Edge-selection gadget. For each distinct pair of colours (i, j) , we construct an edge-selection gadget. The gadget has two main components, which we denote by $C_1(i, j)$ and $C_2(i, j)$. The component $C_1(i, j)$ has $|E_{(i,j)}|$ rows of intervals each of which corresponds to an edge (u, v) of G such that $\{c(u), c(v)\} = \{i, j\}$; see Figure 3(a). Each row has four columns of intervals; the intervals in the same column pairwise intersect. Moreover, there is an offset such that an interval in column t intersects the interval in column $t + 1$ that is in the row immediately above it. The component $C_2(i, j)$ has $|E_{(i,j)}|$ rows and only two columns. There is also an offset between the intervals similar to the offset defined for the intervals in $C_1(i, j)$; see Figure 3(b). The row r in $C_1(i, j)$ corresponds to an edge $(u, v) \in E(G)$ if and only if the row r in $C_2(i, j)$ corresponds to the edge $(u, v) \in E(G)$.

Recall that for each edge in $E(G)$, we associate four 2-intervals; we next describe the construction of these 2-intervals. Let $(u, v) \in E(G)$ such that $c(u) = i, c(v) = j$ and $i < j$. Then, the 2-interval $I_{\{u,v\}}$ (resp., $I'_{\{u,v\}}$) is composed of the interval in the first column (resp., last column) of the row corresponding to (u, v) in $C_1(i, j)$ and the first interval (resp., second interval) of the row corresponding to (u, v) in $C_2(i, j)$. See Figure 3 for an illustration. The 2-interval $I_{(u,v)}$ (associated with the “directed” edge (u, v)) is composed of the interval in the second column of $C_1(i, j)$ and the interval in the vertex-selection gadget of i that is in the row corresponding to vertex u and the column for colour j . The 2-interval corresponding to the “directed” edge (v, u) is constructed in a similar way: it consists of the interval in the third column of $C_1(i, j)$ and the interval in the vertex-selection gadget of j that is in the row corresponding to vertex v and the column for colour i . Figure 4 illustrates an example for constructing the two 2-intervals corresponding to such “directed” edges. Note that the latter two 2-intervals that correspond to “directed” edges are used for validation: they ensure that if the 2-intervals of a vertex u with colour i is selected, then all the selected edges with an endpoint of colour i are incident to u .



■ **Figure 4** An illustration of the two 2-intervals corresponding to the “directed” edges (u, v) and (v, u) , assuming $c(u) = 1$ and $c(v) = 2$. The dashed (red) rectangles shown in gadgets $I_1(\cdot)$ indicate the first and last columns of intervals in the gadget.

► **Lemma 2.** *Let S be a feasible solution for the 2-interval pattern problem, and consider an edge-selection gadget T . If there are four 2-intervals in S such that each of them has at least one interval in T , then all such four 2-intervals must have intervals from the same row of T .*

Proof. The proof uses an argument similar to the one we used in the proof of Lemma 1. Suppose that T corresponds to edges with colours i and j , where $i < j$. Now, consider the gadget $C_1(i, j)$. Clearly, S can have at most one interval from each column of $C_1(i, j)$. Since S has four 2-intervals that have at least one interval in T , the set S contains exactly one interval from each column of T . Suppose that the interval of the first column of $C_1(i, j)$ (that is in S) is at row t for some $1 \leq t \leq |E_{(i,j)}|$. Notice that this interval forms a 2-interval with the first interval in row t of $C_2(i, j)$ and so that interval must also be in S (these two intervals form a valid 2-interval and S is a feasible solution). We now show that the interval of the last column of $C_1(i, j)$ (that is in S) must also be at row t . Suppose for the sake of contradiction that it is at a row $t' \neq t$. First, by construction, this interval forms a 2-interval with the second interval in row t' of $C_2(i, j)$ and so that interval must also be in S . If $t' > t$, then the second interval in row t' of $C_2(i, j)$ intersects with the first interval in row t of $C_2(i, j)$ by the construction and so they cannot both be in S – a contradiction. Moreover, if $t' < t$, then S cannot contain an interval from both the second and third columns of $C_1(i, j)$ because at least one of them intersects the interval of S that is in either the first or the last column of $C_1(i, j)$ – a contradiction. Therefore, $t' = t$ and so the two intervals in S that are in $C_2(i, j)$ are also from the same row t . Finally, the fact that $t' = t$ forces the intervals in the second and third columns of $C_1(i, j)$ (that are in S) to be also from the row t . ◀

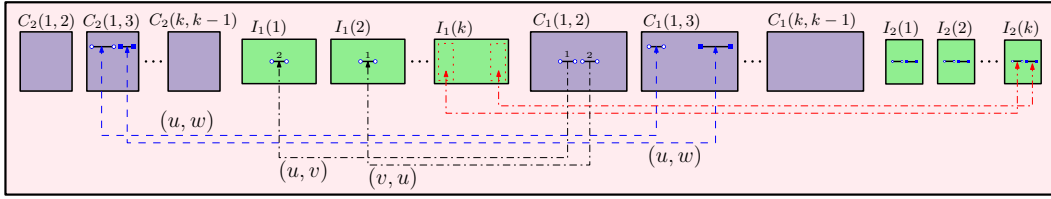
By the above constructions, we obtain the set \mathcal{F} of 2-intervals as

$$\mathcal{F} = \{I_v, I'_v | v \in V(G)\} \cup \{I_{\{u,v\}}, I'_{\{u,v\}}, I_{(u,v)}, I_{(v,u)} | (u, v) \in E(G)\}.$$

Since we associate each vertex with two 2-intervals and each edge with four 2-intervals, we have $|\mathcal{F}| = 2|V(G)| + 4|E(G)|$. The construction of our gadgets can all be done in FPT-time. In the following, we show the arrangement of the gadgets on the real line specific to each of $\mathcal{R} = \{\square, \emptyset\}$ and $\mathcal{R} = \{<, \emptyset\}$. Then, we show that any k -multicoloured clique in G corresponds to $2k + 4\binom{k}{2}$ pairwise disjoint 2-intervals of \mathcal{F} . For brevity, let $k' = 2k + 4\binom{k}{2}$ for the rest of this section.

3.1 Hardness for $\mathcal{R} = \{\square, \emptyset\}$

We now show how to arrange the gadgets on the real line when $\mathcal{R} = \{\square, \emptyset\}$. To this end, consider the ordering $\{1, 2, \dots, k\}$ of colours. We place the gadgets on disjoint regions of the real line from left to right as follows. First, for each pair of distinct colours i and j , $1 \leq i < j \leq k$, we place the gadget $C_2(i, j)$ on the line in lexicographic order; that is, we first place the gadgets $C_2(1, j)$ for all $j = 2, \dots, k$, then the gadgets $C_2(2, j)$ for all $j = 3, \dots, k$



■ **Figure 5** The arrangement of gadgets for $\mathcal{R} = \{\square, \diamond\}$.

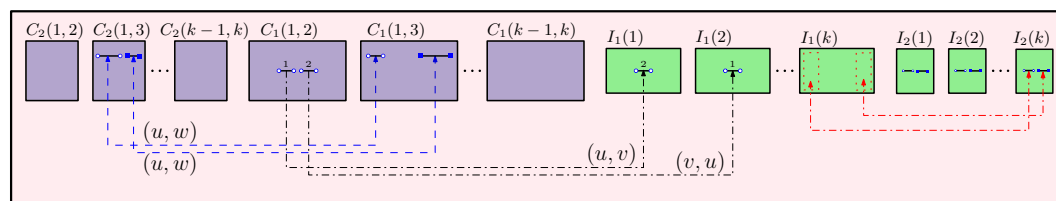
and so on. Then, we place the gadgets $I_1(c)$ ($1 \leq c \leq k$) from left to right in the increasing order of c . Next, we place the gadgets $C_1(i, j)$, $1 \leq i < j \leq k$ in the same order as we placed their corresponding gadgets $C_2(i, j)$. Finally, we place the gadgets $I_2(c)$ ($1 \leq c \leq k$) in the same order as we placed their corresponding gadgets $I_1(c)$. See Figure 5 for an example. This forms our instance $(\mathcal{F}, \mathcal{R}, k')$ of the 2-interval pattern problem, where $\mathcal{R} = \{\square, \diamond\}$ and $k' = 2k + 4\binom{k}{2}$. Clearly, this arrangement can be done in FPT-time. Moreover, one can verify that any two 2-intervals in this instance are \mathcal{R} -comparable, where $\mathcal{R} = \{\square, \diamond\}$.

► **Lemma 3.** *Graph G has a k -multicoloured clique if and only if the 2-interval pattern problem on \mathcal{F} has a feasible solution of size k' with respect to $\mathcal{R} = \{\square, \diamond\}$.*

Proof. (\Rightarrow) Suppose that G has a k -multicoloured clique. For each colour c , let v_c be the vertex in the clique with colour c . Then, for every colour c , we select the two 2-intervals I_{v_c} and I'_{v_c} from the vertex-selection gadget corresponding to c . Moreover, for every pair of colours i and j with $i < j$, let (u_i, u_j) be the edge in the clique such that $c(u_i) = i$ and $c(u_j) = j$. Then, we select the four 2-intervals $I_{\{u_i, u_j\}}, I'_{\{u_i, u_j\}}, I_{(u_i, u_j)}$ and $I_{(u_j, u_i)}$. In this way, we have selected k' 2-intervals in total. Moreover, by the arrangement of gadgets on the real line, one can verify that this set of k' 2-intervals is \mathcal{R} -comparable.

(\Leftarrow) Consider a set S of k' 2-intervals that is a feasible solution for the 2-interval pattern problem with respect to $\mathcal{R} = \{\square, \diamond\}$. First, observe that S can have at most one interval from the first column of every vertex-selection gadget. We now show that it must contain at least one such interval from the first column of every vertex-selection gadget. Let $S_1 \subseteq S$ (resp., $S_2 \subseteq S$) be the set of 2-intervals such that each 2-interval in S_1 has at least one interval in a vertex-selection gadget (resp., an edge-selection gadget). Moreover, let $S_3 \subseteq S$ (resp., $S_4 \subseteq S$) be the set of 2-intervals such that each 2-interval in S_3 (resp., S_4) has exactly two intervals from the same vertex-selection gadget (resp., the same edge-selection gadget). Observe that $|S_2| \leq 4\binom{k}{2}$ because the component $C_1(\cdot)$ of an edge-selection gadget has four columns and no two intervals in S can come from the same column of any given $C_1(\cdot)$. This means that $|S_3| \geq 2k$. But, there are exactly k vertex-selection gadgets and at most two 2-intervals of S_3 can be from the same vertex-selection gadget. Hence, $|S_3| = 2k$ and so $|S_2| = 4\binom{k}{2}$. Since there are exactly $\binom{k}{2}$ edge-selection gadgets, it follows that we have exactly four 2-intervals in S that come from the same edge-selection gadget. By Lemma 2, all the 2-intervals coming from the same edge-selection gadget lie in the same row of the gadget.

On the other hand, $|S_1 \setminus S_3| \leq k(k-1) = 2\binom{k}{2}$ because we have k vertex-selection gadgets, the component $I_1(\cdot)$ of any vertex-selection gadget has $k-1$ “internal” intervals, and at most one of such internal intervals (per column, per vertex-selection gadget) can be in S . Notice that a 2-interval has exactly one interval in a vertex-selection gadget if and only if it has exactly one interval in an edge-selection gadget. Therefore, $S_2 \setminus S_4 = S_1 \setminus S_3$. Since $\{S_1, S_3, S_4\}$ (or, S_2, S_3, S_4) forms a partition of S , we must have $|S_2 \setminus S_4| = |S_1 \setminus S_3| = 2\binom{k}{2}$. That is, there are exactly $2\binom{k}{2}$ 2-intervals that have exactly one interval in a vertex-selection gadget and the other interval in an edge-selection gadget. Notice that at most $k-1$ of



■ **Figure 6** The arrangement of gadgets for $\mathcal{R} = \{<, \emptyset\}$.

such $2^{\binom{k}{2}}$ 2-intervals can come from the same vertex-selection gadget. Since there are k vertex-selection gadgets, there are exactly $k - 1$ of them from each vertex-selection gadget. This means that, for each vertex-selection gadget, there are $k + 1$ 2-intervals in S that come from this gadget. By Lemma 1, these $k + 1$ 2-intervals all come from the same row of the gadget. Hence, we select the k vertices corresponding to these k rows. We now claim that they are a feasible solution for the k -multicoloured clique. Clearly, each selected vertex has a unique colour. Moreover, take any colour c and let u be the vertex that we selected with colour c . Recall that all the intervals of S that come from the vertex-selection gadget c are in the same row as that of u . There are $k - 1$ of them (excluding those corresponding to u itself) and each is paired with an interval in an edge-selection gadget corresponding to the pair (c, c') of colours, for all colours $c' \neq c$. Therefore, there exists an edge between u and every other selected vertex and so the k selected vertices are indeed a feasible solution for the k -multicoloured clique. ◀

► **Theorem 4.** *The 2-interval pattern problem is $W[1]$ -hard when $\mathcal{R} = \{\square, \emptyset\}$.*

3.2 Hardness for $\mathcal{R} = \{<, \emptyset\}$

We now show that the 2-interval pattern problem is $W[1]$ -hard even when $\mathcal{R} = \{<, \emptyset\}$. To this end, we show how to arrange the gadgets on the real line such that any pair of two 2-intervals are $\{<, \emptyset\}$ -comparable. Then, one can prove a result similar to Lemma 3 for $\mathcal{R} = \{<, \emptyset\}$, concluding that the problem is $W[1]$ -hard even for $\mathcal{R} = \{<, \emptyset\}$. Here, we only show the arrangement.

Consider the ordering $\{1, 2, \dots, k\}$ of colours. We place the gadgets on disjoint regions of the real line from left to right as follows. First, for each pair of distinct colours i and j , $1 \leq i < j \leq k$, we place the gadget $C_2(i, j)$ on the line in this order; that is, we first place the gadgets $C_2(1, j)$ for all $j = 2, \dots, k$, then the gadgets $C_2(2, j)$ for all $j = 3, \dots, k$ and so on. Then, we place the gadgets $C_1(i, j)$, $1 \leq i < j \leq k$ in the same order as we placed their corresponding gadgets $C_2(i, j)$. Next, we place the gadgets $I_1(c)$ ($1 \leq c \leq k$) from left to right in the increasing order of c . Finally, we place the gadgets $I_2(c)$ ($1 \leq c \leq k$) in the same order as we placed their corresponding gadgets $I_1(c)$. See Figure 6 for an example. This forms our instance $(\mathcal{F}, \mathcal{R}, k')$ of the 2-interval pattern problem, where $\mathcal{R} = \{<, \emptyset\}$ and $k' = 2k + 4\binom{k}{2}$. Clearly, this arrangement can be done in FPT-time and one can verify that every of pair of 2-intervals are $\{<, \emptyset\}$ -comparable.

► **Theorem 5.** *The 2-interval pattern problem is $W[1]$ -hard when $\mathcal{R} = \{<, \emptyset\}$.*

4 Conclusion

We showed that 2-interval pattern problem is $W[1]$ -hard when $\mathcal{R} = \{\square, \emptyset\}$ and $\mathcal{R} = \{<, \emptyset\}$; hence, fully settling the parameterized complexity of the problem when parameterized by the size of an optimal solution. It would be interesting to examine FPT-algorithms with

respect to other parameters such as the maximum number of pairwise intersecting 2-intervals. Another direction would be to consider the complexity of a variant of the problem in which every two input intervals are pairwise disjoint; i.e., each interval is a point on the real line [9].

References

- 1 Reuven Bar-Yehuda, Magnús M. Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15, 2006.
- 2 Guillaume Blin, Guillaume Fertin, and Stéphane Vialette. New results for the 2-interval pattern problem. In *proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004), Istanbul, Turkey*, pages 311–322, 2004.
- 3 Erdong Chen, Linji Yang, and Hao Yuan. Improved algorithms for largest cardinality 2-interval pattern problem. *J. Comb. Optim.*, 13(3):263–275, 2007.
- 4 Jih-H. Chen, Shu-Yun Le, and Jacob V. Maizel. Prediction of common secondary structures of RNAs: a genetic algorithm approach. *Nucleic Acids Research*, 28(4):991–999, 2000.
- 5 Maxime Crochemore, Danny Hermelin, Gad M. Landau, Dror Rawitz, and Stéphane Vialette. Approximating the 2-interval pattern problem. *Theor. Comput. Sci.*, 395(2-3):283–297, 2008.
- 6 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.
- 7 Minghui Jiang. A 2-approximation for the preceding-and-crossing structured 2-interval pattern problem. *J. Comb. Optim.*, 13(3):217–221, 2007.
- 8 Minghui Jiang. A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model. In *proceedings of the First International Conference on Combinatorial Optimization and Applications (COCOA 2007), Xi'an, China*, pages 378–387, 2007.
- 9 Shuai Cheng Li and Ming Li. On two open problems of 2-interval patterns. *Theor. Comput. Sci.*, 410(24-25):2410–2423, 2009.
- 10 Jihong Ren, Baharak Rastegari, Anne Condon, and Holger H. Hoos. HotKnots: Heuristic prediction of RNA secondary structure including pseudoknots. *RNA*, 11:1194–1504, 2005.
- 11 Elena Rivas and Sean R. Eddy. A dynamic programming algorithm for rna structure prediction including pseudoknots. *J. of Molecular Biology*, 285(5):2053–2068, 1999.
- 12 Jianhua Ruan, Gary D. Stormo, and Weixiong Zhang. An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. *Bioinformatics*, 20(1):58–66, 2004.
- 13 Stéphane Vialette. Combinatorial pattern matching, 13th annual symposium, CPM 2002, fukuoka, japan, july 3-5, 2002, proceedings. In Alberto Apostolico and Masayuki Takeda, editors, *Combinatorial Pattern Matching, 13th Annual Symposium, CPM 2002, Fukuoka, Japan, July 3-5, 2002, Proceedings*, volume 2373 of *Lecture Notes in Computer Science*, pages 53–63. Springer, 2002.
- 14 Stéphane Vialette. On the computational complexity of 2-interval pattern matching problems. *Theor. Comput. Sci.*, 312(2-3):223–249, 2004.
- 15 Stéphane Vialette. Two-interval pattern problems. In *Encyclopedia of Algorithms – 2008 Edition*. Springer, 2008.
- 16 Jizhen Zhao, Russell L. Malmberg, and Liming Cai. Rapid *ab initio* RNA folding including pseudoknots via graph tree decomposition. In *proceedings of the sixth International Workshop on Algorithms in Bioinformatics (WABI 2006), Zurich, Switzerland*, pages 262–273, 2006.

Recoloring Interval Graphs with Limited Recourse Budget

Bartłomiej Bosek 

Theoretical Computer Science Department, Faculty of Mathematics and Computer Science,
Jagiellonian University in Kraków, Poland
bosek@tcs.uj.edu.pl

Yann Disser 


Department of Mathematics, TU Darmstadt, Germany
disser@mathematik.tu-darmstadt.de

Andreas Emil Feldmann 

Department of Applied Mathematics, Charles University in Prague, Czech Republic
<https://sites.google.com/site/aefeldmann>
Andreas.Feldmann@mff.cuni.cz

Jakub Pawlewicz 

University of Warsaw, Poland
pan@mimuw.edu.pl

Anna Zych-Pawlewicz 

University of Warsaw, Poland
anka@mimuw.edu.pl

Abstract

We consider the problem of coloring an interval graph dynamically. Intervals arrive one after the other and have to be colored immediately such that no two intervals of the same color overlap. In each step only a limited number of intervals may be recolored to maintain a proper coloring (thus interpolating between the well-studied online and offline settings). The number of allowed recolorings per step is the so-called *recourse budget*. Our main aim is to prove both upper and lower bounds on the required recourse budget for interval graphs, given a bound on the allowed number of colors.

For general interval graphs with n vertices and chromatic number k it is known that some recoloring is needed even if we have $2k$ colors available. We give an algorithm that maintains a $2k$ -coloring with an amortized recourse budget of $\mathcal{O}(\log n)$. For maintaining a k -coloring with $k \leq n$, we give an amortized upper bound of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, and a lower bound of $\Omega(k)$ for $k \in \mathcal{O}(\sqrt{n})$, which can be as large as $\Omega(\sqrt{n})$.

For unit interval graphs it is known that some recoloring is needed even if we have $k + 1$ colors available. We give an algorithm that maintains a $(k + 1)$ -coloring with at most $\mathcal{O}(k^2)$ recolorings per step in the worst case. We also give a lower bound of $\Omega(\log n)$ on the amortized recourse budget needed to maintain a k -coloring.

Additionally, for general interval graphs we show that if one does not insist on maintaining an explicit coloring, one can have a k -coloring algorithm which does not incur a factor of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ in the running time. For this we provide a data structure, which allows for adding intervals in $\mathcal{O}(k^2 \log^3 n)$ amortized time per update and querying for the color of a particular interval in $\mathcal{O}(\log n)$ time. Between any two updates, the data structure answers consistently with some optimal coloring. The data structure maintains the coloring implicitly, so the notion of recourse budget does not apply to it.

2012 ACM Subject Classification Theory of computation \rightarrow Dynamic graph algorithms; Theory of computation \rightarrow Online algorithms; Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Colouring, Dynamic Algorithms, Recourse Budget, Interval Graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.17



© Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, and Anna Zych-Pawlewicz;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 17; pp. 17:1–17:23



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding

Bartłomiej Bosek: National Science Centre of Poland, project number 2017/26/D/ST6/00264.

Yann Disser: ‘Excellence Initiative’ of the German Federal and State Governments, and Graduate School CE at TU Darmstadt.

Andreas Emil Feldmann: Czech Science Foundation GAČR (grant #17-10090Y), and Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

Anna Zych-Pawlewicz: National Science Centre of Poland, project number 2017/26/D/ST6/00264.

1 Introduction

Graph coloring is one of the most prominent disciplines within graph theory, with plenty of variants, applications, and deep connections to theoretical computer science. A *proper k -coloring* of a graph, for a positive integer k , is an assignment of colors in $\{1, \dots, k\}$ to the vertices of the graph in such a way that no two adjacent vertices share a color. The chromatic number of the graph is the smallest integer k for which a proper k -coloring exists. In general, it is NP-hard [17, 36] to approximate the chromatic number of an n -vertex graph to within a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$. The literature offers many results for restricted graph classes.

In this paper, we consider the class of interval graphs, for which a linear time greedy algorithm achieves the optimum coloring [26]. Our main interest is in a dynamic setting, where intervals arrive one at a time, and one needs to maintain the coloring after each interval addition. We mainly study how many vertex recolorings are needed to maintain a reasonable coloring. The number of changes one needs to introduce to the maintained solution (in our case vertex recolorings) upon an update is referred to as *recourse bound* or *recourse budget* in the literature. A recourse budget of zero coincides with the online setting, where the algorithm’s decisions are irrevocable. The online model is natural for many problems [14, 24, 27, 29] and has been widely studied, very often revealing pessimistic lower bounds. It is natural to ask if the situation improves if one allows a limited recourse budget. This model has been successfully applied to a variety of problems, including spanning tree and Steiner tree variants, bipartite matchings, and coloring [2, 3, 9, 10, 11, 16, 23, 21]. The proposed algorithms could often be efficiently implemented [3, 9, 21].

Formally, we are interested in the following problem. We get a sequence of half-open intervals $\{[a_i, b_i)\}_{i=1}^n$, which defines a sequence of instances $\mathcal{I}_j = \{[a_i, b_i)\}_{i=1}^j$, where \mathcal{I}_j differs from \mathcal{I}_{j-1} by one interval. The instances may be interpreted as graphs, where the nodes are intervals and the edges connect intersecting intervals. The intervals arrive one at a time. After the j -th interval is revealed, the algorithm needs to compute a proper coloring C_j for the intersection graph of \mathcal{I}_j . We wish to minimize the recourse budget, which is the number of vertices with different colors in C_j and C_{j-1} . We also consider the special case of *unit* interval graphs, where each interval is of the form $b_i = a_i + 1$. For the sake of simplicity we assume that every instance \mathcal{I}_j is k -colorable and k is known a priori, but it is not difficult to get rid of this assumption. Our results are summarized in Table 1 together with some known results from the literature for comparison. Unless stated otherwise, all the bounds in the table are amortized, i.e., they bound the average recourse budget per insertion.

For general interval graphs our first result shows that if we allow $\mathcal{O}(\log n)$ recolorings per interval insertion, we can improve the ratio of 3 of the online algorithm by Kierstead and Trotter [20] to 2. Since the ratio of 3 is best possible in the online setting, our result shows that only a modest number of recolorings are needed to obtain an improvement. If we allow a higher number of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ recolorings per update, we can even maintain an optimal solution. A trivial algorithm that recolors all intervals in each step has a recourse

■ **Table 1** Our results for interval graphs (top) and unit interval graphs (bottom). All runtimes are amortized, if not otherwise stated.

	colors	upper bound	recourse budget	lower bound
general	$3k - 2$	0 [20]		0
	$2k$	$\mathcal{O}(\log n)$ (Thm 5)		> 0 [20]
	k	$\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ (Thm 10)		$\begin{cases} \Omega(k) \text{ (Cor 21)} & \text{for } k \in \mathcal{O}(\sqrt{n}) \\ \Omega(\sqrt{n}) \text{ (Cor 20)} & \text{for } k \in \Theta(\sqrt{n}) \\ \Omega(\log n) \text{ (Thm 1)} & \text{for } k = 2 \end{cases}$
unit interval	$2k - 1$	0 [5]		0
	$k + 1$	$\mathcal{O}(k^2)$ worst case (Thm 2)		> 0 [5]
	k	$\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ (Thm 10)		$\Omega(\log n)$ (Thm 1)

budget of n , resulting in the bound $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ of Table 1. Note that this bound is non-trivial (i.e., smaller than n) for $k \in \mathcal{O}(\frac{\log n}{\log \log n})$. We complement these results with a lower bound for the budget of $\Omega(k)$, which can be as high as $\Omega(\sqrt{n})$ if k grows with n . We obtain another lower bound of $\Omega(\log n)$ for $k = 2$. The latter bound is even valid for unit interval graphs, for which we also show that if we allow a budget of $\mathcal{O}(k^2)$ recolorings (i.e., independent of n), we can maintain a solution using just one extra color compared to the optimum. Due to our lower bound of $\Omega(\log n)$ for maintaining an optimal coloring, it is clear that an extra color is necessary if we want to keep the budget constant for a constant k .

It is straightforward to see that our algorithms, except for the exact algorithm for general interval graphs that uses an amortized recourse budget of $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, can be implemented efficiently. However, we can improve the exact algorithm significantly if we do not insist on maintaining an explicit coloring, i.e., if we do not require that the color of an interval can be retrieved in constant time. In Section 5 we provide a data structure, which allows for adding intervals in $\mathcal{O}(k^2 \log^3 n)$ amortized time per update and querying for the color of a particular interval in $\mathcal{O}(\log n)$ time. Between two updates the data structure answers queries consistently with some optimal coloring. The data structure maintains the coloring implicitly, so the notion of recourse budget does not apply to it.

1.1 Related work

Due to the inapproximability of the graph coloring problem, the positive results for dynamic coloring of general graphs are mostly of heuristic and experimental nature [25, 28, 30, 32, 35]. From the theoretical perspective, just recently there have been a few results concerning the recourse budget for coloring general graphs [2, 33] and dynamic general graph coloring with $\Delta + 1$ colors [4], where Δ is the maximum degree in the graph.

Barba et al. [2] devise two complementary algorithms for the regime of adding and removing edges. For any $d > 0$, the first (resp. second) algorithm maintains a $k(d + 1)$ -coloring (resp. $k(d + 1)n^{1/d}$ -coloring) of a k -colorable graph and recolors at most $(d + 1)n^{1/d}$ (resp. d) vertices per update, where updates include edge and vertex additions and removals. The authors also show that the first trade-off is essentially tight, and the bad example is a tree. So if one insists on a constant approximation ratio, one must incur polynomial recourse budget for every class of graphs that contains trees. The symmetry between these trade-offs may make it tempting to believe that the second trade-off is also tight. However, Solomon

and Wein [33] show, that in the regime of adding and removing edges, there is a deterministic algorithm for maintaining an $\mathcal{O}(\frac{k}{d} \log^3 n)$ -coloring with $\mathcal{O}(d)$ recolorings per update step for any $d \in \mathcal{O}(\log n)$. They also show that a randomized algorithm performs slightly better. Solomon and Wein additionally consider bounded arboricity graphs, for which, using their result on the recourse budget, they provide an efficient dynamic algorithm maintaining an $\mathcal{O}(\alpha \log^2 n)$ -coloring with polyloglog amortized time per update. Bhattacharya et al. [4] studied the problem of efficient dynamic coloring when the maximum degree of the dynamic graph remains bounded by Δ at all times. They present a randomized (resp. deterministic) algorithm for maintaining a $(\Delta + 1)$ -coloring (resp. $\Delta(1 + o(1))$ -coloring) with amortized $\mathcal{O}(\log \Delta)$ (resp. $\text{polylog}(\Delta)$) update time.

To the best of our knowledge, no dynamic algorithms for the class of interval graphs have been proposed in the literature. Our motivation for studying this class of graphs in the incremental regime stems from the rich literature on the problem of online poset coloring. Schmerl asked whether an effective online chain partitioning algorithm exists, and this was answered in the affirmative by Kierstead in [18]. His algorithm uses at most $(5^w - 1)/4$ chains on posets of width w . Szemerédi proved a quadratic lower bound of $\binom{w+1}{2}$ (see [5, 19] for a proof). In [7], Bosek and Krawczyk provide an online algorithm that partitions posets of width w into at most $w^{13 \log_2 w}$ chains. This yields the first subexponential upper bound for the online chain partitioning problem. In [6] Bosek et al. improve this to $w^{6.5 \log_2 w + 7}$ with a shorter proof. Very recently, in [8] Bosek and Krawczyk present an online algorithm that partitions posets of width w into $w^{O(\log \log w)}$ chains. At this point, the problem of whether there is an online algorithm using polynomially many chains is still open.

The problem of online interval poset chain coloring is equivalent to the problem we are studying with the recoloring budget limited to zero. It has been extensively studied in many different variants [1, 5, 12, 20]. A well-known theorem of Kierstead and Trotter [20], translated to our setting, states that a $(3k - 2)$ -coloring of k -colorable graph can be maintained online and this is the best we can do if we do not allow recolorings. A folklore result [5] states that for unit interval graphs a $(2k - 1)$ -coloring of k -colorable graph can be maintained online and this is also tight. It is natural to wonder how many recolorings we need when the approximation ratio is going from 3 down to 1 for general interval graphs, or from 2 down to 1 for unit interval graphs. This is the main question we aim to answer in this paper. Nevertheless, it is not hard to show that our $(k + 1)$ -coloring algorithm for unit interval graphs can be extended to a fully dynamic setting (allowing also interval removals). In particular, this gives one more non-trivial class of graphs where the lower bound of Barba et al. [2] does not apply.

2 Unit intervals

In this section we focus on the class of unit interval graphs. This class is equivalent with proper interval graphs, i.e., interval graphs where no interval is contained in another interval [31]. We show a lower-bound of $\Omega(\log n)$ for the recourse budget for maintaining an optimal coloring.

► **Theorem 1.** *Maintaining an optimum coloring of a 2-colorable unit interval graph requires an amortized recourse budget of $\Omega(\log n)$.*

Proof. We describe a 2-colorable interval graph that appears online in the form of recursively constructed gadgets. We start with the gadget G_0 consisting of the two intersecting intervals $[0, 1)$ and $[0.5, 1.5)$ that can be 2-colored without recoloring. Obviously, G_0 admits a unique 2-coloring (up to renaming colors).

Now, for $i \in \{1, 2, \dots\}$, assume we have a recursive construction G_{i-1} that admits a unique 2-coloring (up to renaming colors), and that all intervals in this coloring fall into $[a, b)$ in one color and into $[a+0.5, b+0.5)$ in the other color. This means that, regarding 2-colorings, G_{i-1} behaves macroscopically exactly like two intervals of the form $[a, b), [a + 0.5, b + 0.5)$. To obtain G_i , we first introduce, one after the other, two G_{i-1} gadgets shifted so that they behave exactly like the pairs of intervals $[a, b), [a + 0.5, b + 0.5)$ and $[b + 1, c), [b + 1.5, c + 0.5)$, respectively. See Fig. 1 along with the following.

Up to renaming colors, there are two ways of coloring the gadgets. If $[a, b)$ and $[b + 1, c)$ receive the same color, we introduce the additional intervals $[b + 0.25, b + 1.25)$ and $[c, c + 1)$. Otherwise, $[a, b)$ and $[b + 1.5, c + 0.5)$ receive the same color, and we introduce the additional intervals $[b, b + 1)$ and $[b + 0.5, b + 1.5)$. In both cases, there is no way of consistently coloring the new intervals without recoloring one of the two gadgets. Since the gadgets admit a unique 2-coloring up to renaming colors, we need to completely recolor one of them by changing the color of all of its intervals. Afterwards, G_i admits a unique 2-coloring (up to renaming colors), and all intervals fall into $[a, c + 0.5)$ in one color and $[a + 0.5, c + 1)$ in the other color. We can therefore proceed with the recursive construction.

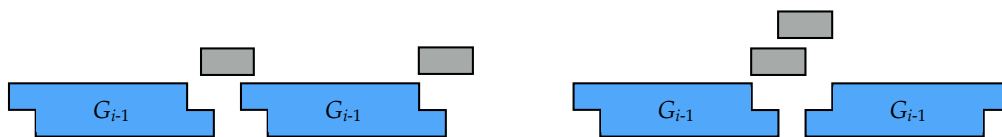
The number of intervals n_i of G_i is given by $n_0 = 2$ and $n_i = 2n_{i-1} + 2$ for $i \in \{1, 2, \dots\}$, which yields $n_i = 2^{i+1} + \sum_{j=1}^i 2^j = 2^{i+2} - 2$. The number of recolorings required during the recursive construction of G_i is given by $r_0 = 0$ and $r_i = 2r_{i-1} + n_{i-1}$ for $i \in \{1, 2, \dots\}$, which yields $r_i = \sum_{j=1}^i 2^{i-j} n_{j-1} = \sum_{j=1}^i 2^{i-j} (2^{j+1} - 2) = i \cdot 2^{i+1} - 2 \sum_{j=0}^{i-1} 2^j = i \cdot 2^{i+1} - 2^i + 1$. This means that, asymptotically, we have $n_i = \Theta(2^i)$ and the amortized number of required recolorings is $r_i/n_i = \Theta(i) = \Theta(\log(n_i))$. ◀

We now prove an upper bound of $\mathcal{O}(k^2)$ for the worst-case recourse budget, which holds if the algorithm can use one extra color. This is in contrast with the lower bound of Theorem 1, which is $\Omega(\log n)$ recourse budget per update for an exact algorithm. We note that our algorithm can also be made to work in the fully dynamic setting (allowing also interval removals) with the same bounds on the required recolorings.

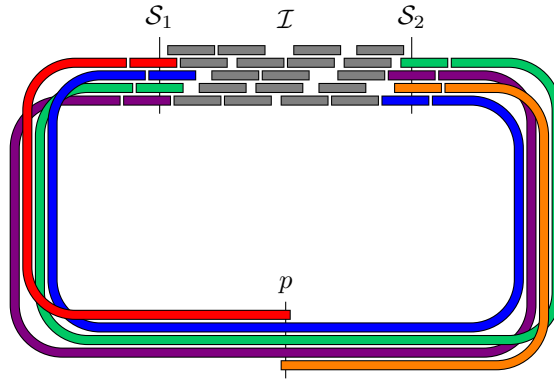
Before we begin, we introduce some definitions. Let $\mathcal{I} = \{[a_1, b_1), \dots, [a_n, b_n)\}$ be a unit interval instance ordered by a_i . A left boundary $\xi_l(\mathcal{I})$ (respectively right boundary $\xi_r(\mathcal{I})$) is a set of intervals intersecting the largest integer smaller than b_1 (respectively the smallest integer larger or equal a_n). Note that $[a_1, b_1) \in \xi_l(\mathcal{I})$ and $[a_n, b_n) \in \xi_r(\mathcal{I})$. A circular arc graph is an intersection graph of (open) arcs lying on the same circle.

► **Theorem 2.** *There exists an algorithm which maintains a $(k + 1)$ -coloring of a k -colorable unit interval graph with $\mathcal{O}(k^2)$ worst case recourse budget per update.*

Proof. We partition the current instance \mathcal{I} into smaller instances $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_m$ and separators between them. Each instance is of size at least lk (except for the last one, which may be smaller), and at most $2lk + k$ for $l = \max\{4, k + 1\}$. The reason for this particular choice of l will become apparent later. In the beginning there is just one instance \mathcal{I}_1 . Whenever an instance \mathcal{I}_i grows above size $2lk + k$, we pick a point p , such that there are at least lk intervals in \mathcal{I}_i completely to the left and lk intervals completely to the right of p . This point



■ **Figure 1** Illustration of the two cases in the recursive construction of G_i .



■ **Figure 2** Illustration of the proof of Lemma 3 for $k = 5$.

partitions \mathcal{I}_i in the desired way. We declare the intervals intersecting p to be a separator \mathcal{S}_i . At any point in time we maintain a partition of the current instance \mathcal{I} into small instances and separators: $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{S}_1 \cup \mathcal{I}_2 \cup \mathcal{S}_2 \dots \cup \mathcal{S}_{m-1} \cup \mathcal{I}_m$, where $m \in \Theta(n/(k^2))$. When adding a new interval, we will recolor the instance \mathcal{I}_i into which the new interval falls, or separator \mathcal{S}_i with neighboring instances if the new interval hits the integer point defining \mathcal{S}_i . The next lemma will be used to do this with at most $k + 1$ colors without changing the colors of the neighboring separators (which are given by some boundary integer points).

► **Lemma 3.** *Let \mathcal{I} be a k -colorable unit interval instance. If $|\mathcal{I}| \geq lk$ for $l = \max\{4, k + 1\}$, then, for any fixed coloring on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ using colors from $[k]$, one can complete this coloring on \mathcal{I} using colors from $[k + 1]$.*

Proof. We first reduce the color completion problem from the lemma statement to the problem of coloring circular arc graphs. This reduction is shown in Figure 2. We draw the intervals of \mathcal{I} as arcs on the north half of a circle, in a way that preserves the intersection relation. Let p be the south pole of the circle, i.e., the point extending the most to the south. For each pair of intervals $(I_1, I_2) \in \xi_l(\mathcal{I}) \times \xi_r(\mathcal{I})$ such that I_1 and I_2 are precolored with the same color, we stretch I_1 (respectively I_2) anticlockwise (respectively clockwise) so that they reach p and then glue them together to form the same arc. The remaining intervals of $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ are only stretched to reach (and intersect) p and are not glued with anything.

We now make use of the following lemma from [34], which allows us to color the obtained circular arc graph instance. We note that this theorem was also used in [15].

► **Lemma 4** ([34]). *Let G be a circular arc graph, $L(G)$ be the maximum number of arcs intersecting a common point on the circle, and $l(G)$ be the smallest number of intervals that cover the circle. If $l(G) \geq 5$ then $\left\lceil \frac{l(G)-1}{l(G)-2} L(G) \right\rceil$ colors suffice to color G and there is a linear time coloring algorithm.*

In order to apply Lemma 4, we need to consider quantities $L(G)$ and $l(G)$ for the instance G that we created. Before the transformation, since \mathcal{I} is k -colorable, there are at most k intervals intersecting one point. After the transformation, if we cut out from the circle $[p - \epsilon, p + \epsilon]$ for some $\epsilon > 0$, we get a stretched instance \mathcal{I} . So for any point on the circle outside $[p - \epsilon, p + \epsilon]$ there are at most k arcs intersecting it. Within $[p - \epsilon, p + \epsilon]$ also at most k arcs intersect, since for every color used on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ there is precisely one arc intersecting p . So $L(G) \leq k$. Also, because $|\mathcal{I}| \geq lk$ and all intervals have unit length, the distance between $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$ is at least l , and so the minimal number of intervals

needed to cover the circle is at least $l + 1$, i.e., $l(G) \geq l + 2$. Setting $l = \max\{4, k + 1\}$ ensures $l(G) \geq 5$ so that the assumptions of Lemma 4 are satisfied and we ensure that $l(G) \geq k + 2$. Due to Lemma 4, we can color \mathcal{I} with a number of colors bounded by $\left\lceil \frac{l(G)-1}{l(G)-2} L(G) \right\rceil = \left\lceil \left(1 + \frac{1}{l(G)-2}\right) L(G) \right\rceil \leq \left\lceil \left(1 + \frac{1}{k}\right) k \right\rceil = k + 1$. Also, any intervals $I_1 \in \xi_l(\mathcal{I})$ and $I_2 \in \xi_r(\mathcal{I})$ are colored the same if and only if their precoloring is the same. Hence, we can permute colors in the obtained coloring so that it complies with the precoloring on $\xi_l(\mathcal{I})$ and $\xi_r(\mathcal{I})$. ◀

When a new interval I_{new} is added, it either fits into an instance \mathcal{I}_i or it belongs to a separator \mathcal{S}_j . In the first case, we recolor $\mathcal{I}_i \cup \{I_{\text{new}}\}$ consistently with the current coloring on \mathcal{S}_{i-1} and \mathcal{S}_i . In the second case, we color the new interval I_{new} with the first color not used on \mathcal{S}_j and recolor \mathcal{I}_j and \mathcal{I}_{j+1} consistently with the current coloring on \mathcal{S}_{j-1} , \mathcal{S}_j , and \mathcal{S}_{j+1} . What remains to be proved is that we can always recolor the chosen piece using $k + 1$ colors. This follows directly from Lemma 3. ◀

3 Low recourse budget for general interval graphs

In this section we focus on presenting the exact algorithm for arbitrary interval graphs with an amortized recourse budget of $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$. Before we move to that, let us mention the bounds for approximating the number of colors (maintaining a ck -coloring is referred to as c -approximation). The algorithm of Kierstead and Trotter [20] can be turned into a 2-approximation if we allow an amortized $\mathcal{O}(\log n)$ recourse budget. The proof of Theorem 5 can be divided into two lemmas that follow below.

► **Theorem 5.** *There is an algorithm maintaining a 2-approximate coloring of an interval graph with amortized recourse budget $\mathcal{O}(\log n)$.*

► **Lemma 6** ([20]). *There is an online algorithm which receives an interval graph G in an online way and produces a partition of G into subgraphs P_1, \dots, P_ω , where each P_i is a sum of disconnected paths and ω is a clique number of G .*

► **Lemma 7.** *There is an incremental algorithm which uses 2 colors on a sum of disconnected paths P with $n \log_2 n$ total changes, where n is a size of P .*

Proof of Lemma 6. While the algorithm receives next vertices, it tries to satisfy the following invariant.

- (I) For any $j \leq \omega$ each clique in $P_1 \cup P_2 \cup \dots \cup P_j$, has size at most j .
- (II) For any $j \leq \omega$ and for any vertex $u \in P_j$ there is a clique in $P_1 \cup P_2 \cup \dots \cup P_{j-1} \cup \{u\}$ of size j .

When new vertex v is presented, the algorithm finds the last j for which the invariant (I) does not hold plus one, i.e. algorithm finds $j_0 := \max\{j \in \mathbb{N} : \omega(P_1 \cup P_2 \cup \dots \cup P_{j-1} \cup \{v\}) \geq j\}$. Then, it adds v to P_{j_0} , i.e., defines a new partition P_1^+, \dots, P_ω^+ of a new graph $G^+ = G \cup \{v\}$ in this way that $P_{j_0}^+ := P_{j_0} \cup \{v\}$ and $P_i^+ := P_i$ for $i \neq j_0$. The invariant (I) for P_j^+ 's is trivially satisfied. The number $j_0 - 1$ is too small, i.e., there is a clique $K \in P_1 \cup P_2 \cup \dots \cup P_{j_0-1} \cup \{v\}$ of size j_0 , which contains the newly presented vertex v . Exactly this clique $K \subseteq P_1^+ \cup \dots \cup P_{j_0-1}^+ \cup \{v\}$ is a witness for the invariant (II) for v . Moreover, the number j_0 is defined so that it will never be greater than the clique number of the graph G .

To understand why each P_i is a sum of disconnected paths, let's consider the interval representation \mathcal{I} of graph G . It means that \mathcal{I} is a family of closed intervals in \mathbb{R} . Moreover, for each $j \leq \omega$ let's define \mathcal{I}_j as a family of intervals corresponding to the vertices of P_j . First, we note the following claim.

17:8 Recoloring Interval Graphs with Limited Recourse Budget

▷ **Claim 8.** There is no interval in \mathcal{I}_j which is covered by the rest of the intervals from \mathcal{I}_j .

Proof. For the contradiction let's assume that there are different intervals $I_0, I_1, \dots, I_t \in \mathcal{I}_j$ such that $I_0 \subseteq I_1 \cup \dots \cup I_t$. Let's $K \subseteq P_1 \cup \dots \cup P_j$ be a clique for I_0 from invariant **(II)**. Each clique in the interval representation can be identified with some real number that belongs to all intervals corresponding to elements from that clique. Let's $r \in \mathbb{R}$ be such a number corresponding to the clique K . Then $r \in I_1 \cup \dots \cup I_t$ and in consequence $r \in I_s$ for some $s \leq t$. If vertex v_s corresponds to the interval I_s then $K \cup \{v_s\} \subseteq P_1 \cup \dots \cup P_j$ forms a clique of size $j + 1$ which contradicts the invariant **(I)**. ◀

The above claim directly implies the following statement.

▷ **Claim 9.** Each vertex in P_j has at most two neighbours in P_j .

Proof. Again, let's \mathcal{I}_j be a family of interval corresponding to vertices from P_j . For the contradiction let's assume that v_0 has three neighbours v_1, v_2, v_3 which corresponds to the intervals I_0, I_1, I_2, I_3 . At the beginning, notice that the sum $I_0 \cup I_1 \cup I_2 \cup I_3$ form also some interval in \mathbb{R} . Let's l and r be the left and the right endpoint of $I_0 \cup I_1 \cup I_2 \cup I_3$, respectively. One of the intervals I_1, I_2, I_3 does not contain any points of l, r . Without loss of generality let us assume that this interval is I_3 . Then $I_3 \subseteq I_0 \cup I_1 \cup I_2$ which is contradictory to the previous claim. ◀

Finally, it is worth noting that interval graphs are also chordal, so they can not contain simple cycles. So, the only possibility is that P_j is a sum of disconnected paths. ◀

Proof of Lemma 7. When new vertex v is coming, it combines two paths. If neighbours of v have the same color then the algorithm colors vertex v on the other one. If neighbours of v have the different colors then the algorithm recolors the shortest path. The given vertex u was recolored when the length of the path containing u increased by at least twice. This causes the vertex u to be recolored at most $\log_2 n$ times. Which gives the total number of recoloring equal $n \log_2 n$. ◀

In the remainder of this section we show a k -coloring algorithm with $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$ recourse budget. Both for the algorithm and the analysis we use the greedy algorithm for coloring interval graphs [26]. The greedy algorithm sorts intervals by their begin coordinates. It processes intervals in that order, and assigns the smallest available colour to the currently processed interval. This simple algorithm was proven optimal [26]. We are now ready to prove the main theorem of this section.

► **Theorem 10.** *There is an algorithm maintaining an optimum coloring of a k -chromatic interval graph with an amortized recourse budget of $\min\{n, \mathcal{O}(k \cdot k! \cdot \sqrt{n})\}$.*

Proof. Note that a trivial algorithm, which recolors all intervals in each step has recourse budget n . We will show that there also is an algorithm with amortized budget $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$, which proves the claim. This algorithm is directly implied by Lemma 11, which is proved next. Due to this lemma n interval insertions into an n -element instance can be executed with a total recourse budget of $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$. The implication is as follows. Imagine we make a total of m insertions. We break the insertion sequence into powers of 2: once we inserted 2^i intervals, we add 2^i more using $\mathcal{O}(k \cdot k! \cdot 2^i \sqrt{2^i})$ recolorings. Let s be such that $2^{s-1} < m \leq 2^s$. The total number of recolorings is bounded by $\sum_{i=1}^s \mathcal{O}(k \cdot k! \cdot 2^i \sqrt{2^i}) = \mathcal{O}(k \cdot k! \cdot \sqrt{m} \sum_{i=1}^s 2^i) = \mathcal{O}(2^{s+1} k \cdot k! \cdot \sqrt{m}) = \mathcal{O}(k \cdot k! \cdot m\sqrt{m})$. ◀

► **Lemma 11.** *There is an algorithm, which, given n intervals, maintains the exact coloring over the course of n interval insertions and recolors a total of $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$ intervals.*

Proof. We move on to presenting the algorithm, followed by the analysis. The idea is to maintain a partition of the dynamically changing instance \mathcal{I} into l disjoint instances $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_l$. We maintain the invariant that the size of each instance is at most $2\lceil\sqrt{n}\rceil + 2k$, and that the size of each instance but the last one is at least $\lceil\sqrt{n}\rceil$. This invariant guarantees that $l \in \mathcal{O}(\sqrt{n})$. At the beginning, the algorithm starts with n intervals, so $|\mathcal{I}| = n$. Then, it is easy to find such a partition. Let $\mathcal{I} = \{[a_1, b_1), \dots, [a_n, b_n)\}$ be sorted by end coordinates. We let $x_1 = b_{\lceil\sqrt{n}\rceil}$ be the first separator point. It may happen that up to k intervals end in the same coordinate, so there are at most $\lceil\sqrt{n}\rceil + k$ intervals to the left of x_1 . We remove intervals to the left and intersecting x_1 from \mathcal{I} and continue in the same manner in order to find separating points $x_2 \dots x_{l-1}$. We let \mathcal{I}_i be the intervals contained between x_{i-1} and x_i , and we define separator \mathcal{S}_i to be the set of all intervals intersecting x_i . Note that the separators are not necessarily disjoint, since intervals can span a long stretch in which many smaller intervals live.

Now consider the dynamically growing instance. If at any time some instance \mathcal{I}_i grows to more than $2\lceil\sqrt{n}\rceil + 2k$, we split it into instances \mathcal{I}'_i and \mathcal{I}''_i , both of size at least $\lceil\sqrt{n}\rceil$, since the separator takes away at most k intervals, and we possibly have to put $\lceil\sqrt{n}\rceil + k$ intervals into \mathcal{I}'_i . At this point \mathcal{I}_i ceases to exist. This ensures that our size invariant remains satisfied at all times.

In each step, the algorithm takes a new interval I_{new} as input. It uses a procedure `total-recolor`(i, j) as a subroutine. Procedure `total-recolor`(i, j) takes two numbers $i, j \in \{1, \dots, l-1\}$, $i \leq j$ as parameters. It is an invariant that I_{new} is entirely contained in (x_{i-1}, x_j) . The procedure recolors the new instance $\mathcal{I} \cup \{I_{\text{new}}\}$ in the following way. It leaves the current coloring as it is on $\mathcal{I}_1, \mathcal{S}_1, \mathcal{I}_2, \mathcal{S}_2, \dots, \mathcal{I}_{i-1}, \mathcal{S}_{i-1}$. Starting with the current coloring on \mathcal{S}_{i-1} , it colors $\mathcal{I}_i \cup \mathcal{S}_i \cup \dots \cup \mathcal{I}_j \cup \mathcal{S}_j \cup \{I_{\text{new}}\}$ greedily. The greedy coloring is consistent with the coloring of \mathcal{S}_{i-1} , but may not be consistent with the current coloring on \mathcal{S}_j . Nevertheless, we can permute the colors in order to obtain the new greedy coloring on \mathcal{S}_j . The procedure permutes the colors in the same way on the remaining part of the instance, i.e., for $\mathcal{I}_{j+1}, \mathcal{S}_{j+1}, \dots, \mathcal{S}_{l-1}, \mathcal{I}_l$. Procedure `total-recolor`(i, j) possibly recolors the whole graph, i.e., it triggers $\mathcal{O}(n)$ recolorings.

Having procedure `total-recolor`(i, j) at hand, the algorithm distinguishes two cases.

1. $I_{\text{new}} \in \mathcal{I}_j$ for some \mathcal{I}_j . In this case we try to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ with k colors in a way that is consistent with the current coloring on \mathcal{S}_{j-1} and \mathcal{S}_j (see the parameterized algorithm of Marx [22] for efficient implementation). There are two more cases now.
 - a. It is possible to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ consistently with \mathcal{S}_{j-1} and \mathcal{S}_j . In this case we perform $\mathcal{O}(\sqrt{n} + k)$ recolorings.
 - b. It is impossible to recolor $\mathcal{I}_j \cup \{I_{\text{new}}\}$ in this way. In this case we call `total-recolor`(j, j).
2. I_{new} intersects some separation point. If $x_i, x_{i+1}, \dots, x_{j-1}$ are the x -coordinates of the separation points intersected by I_{new} , we call `total-recolor`(i, j).

As for the analysis of the above algorithm, the recoloring budget claimed in Lemma 11 follows from Lemma 12 and Lemma 13 below. Observe, that the only expensive operation we need to amortize for is `total-recolor`(\cdot, \cdot), which performs $\mathcal{O}(n)$ recolorings. Due to Lemma 12, the total number of recolorings triggered by `total-recolor`(i, j) for $i \neq j$ is $\mathcal{O}(k \cdot n\sqrt{n})$. Due to Lemma 13, the total number of recolorings triggered by `total-recolor`(i, i) on a particular instance \mathcal{I}_i is $\mathcal{O}(k \cdot k! \cdot n)$. Observe, that the number of instances that

17:10 Recoloring Interval Graphs with Limited Recourse Budget

ever exist is $\mathcal{O}(\sqrt{n})$: the algorithm starts with n intervals, and for these initial intervals it creates $l \in \mathcal{O}(\sqrt{n})$ instances. Further on it creates at most $\mathcal{O}(\sqrt{n})$ more instances by splitting the existing ones. Summed over all instances that exist at some point of time this gives $\mathcal{O}(k \cdot k! \cdot \sqrt{n} \cdot n)$ recolorings. The total number of recolorings caused by case 1 a) of the algorithm is bounded by $\mathcal{O}(n(\sqrt{n} + k))$. The number of all recolorings the algorithm performs is hence bounded by $\mathcal{O}(k \cdot k! \cdot n\sqrt{n})$, as claimed. \blacktriangleleft

► **Lemma 12.** *The total number of calls to `total-recolor`(i, j) for any $i \neq j$ is in $\mathcal{O}(k\sqrt{n})$.*

Proof. The call to `total-recolor`(i, j) for $i \neq j$ is only made if I_{new} intersects some separator line. There are $\mathcal{O}(\sqrt{n})$ separator lines created by the algorithm, and at most k intervals may be added to each separator. This gives the claim of the lemma. \blacktriangleleft

► **Lemma 13.** *For every instance \mathcal{I}_j the algorithm calls `total-recolor`(j, j) at most $2k \cdot k!$ times overall in step 1b).*

Proof. Fix $i \in [1, \dots, l]$ and consider the pair of separators \mathcal{S}_{i-1} and \mathcal{S}_i . We say that \mathcal{I}_i is reset when procedure `total-recolor`(j_1, j_2) is called with $j_1 \neq j_2$ for $j_1 \leq i \leq j_2 + 1$. In what follows we will prove that between two consecutive resets of \mathcal{I}_i , procedure `total-recolor`(i, i) can be called at most $k!$ times. This will finish the proof, as any `total-recolor`(j_1, j_2) call resetting \mathcal{I}_i adds an interval to either \mathcal{S}_{i-1} or \mathcal{S}_i or both, so there can be at most $2k$ such calls. Note that non-resetting calls of `total-recolor`() do not alter \mathcal{S}_{i-1} and do not alter \mathcal{S}_i , so between two resets of \mathcal{I}_i separators \mathcal{S}_{i-1} and \mathcal{S}_i remain unchanged (although their colors may change). It may happen that we split \mathcal{I}_j , but then \mathcal{I}_j ceases to exist and hence is recolored no more (instead, the instances that \mathcal{I}_j splits into are recolored). In what follows we consider a time period between two consecutive resets of \mathcal{I}_i . We refer to this time period as a phase. The phase starts when an interval has been added to either \mathcal{S}_{i-1} or \mathcal{S}_i or both and lasts as long as no other interval is added to \mathcal{S}_{i-1} or \mathcal{S}_i and as long as \mathcal{I}_i is not split.

Let \mathcal{I}_i^f be the instance \mathcal{I}_i after the last insertion within the phase. In what follows we always view \mathcal{I}_i as a current instance, before inserting a new interval I_{new} . We let $\mathcal{J}_i = \mathcal{S}_{i-1} \cup \mathcal{I}_i \cup \mathcal{S}_i$ and $\mathcal{J}_i^f = \mathcal{S}_{i-1} \cup \mathcal{I}_i^f \cup \mathcal{S}_i$.

For solution SOL maintained by the algorithm we define SOL_{i-1} and SOL_i to be SOL restricted to \mathcal{S}_{i-1} and \mathcal{S}_i respectively. Similarly, for any optimum solution OPT for \mathcal{J}_i^f we define its restriction to \mathcal{S}_{i-1} and \mathcal{S}_i as OPT_{i-1} and OPT_i . Let $\text{Greedy}(\mathcal{J}_i^f)$ be the optimal greedy solution to \mathcal{J}_i^f . Observe that if we permute colors of an optimal solution for \mathcal{J}_i^f , the solution remains optimal. This leads us to define the optimal solution space $\Sigma = \mathfrak{S}_k \circ \text{Greedy}(\mathcal{J}_i^f)$, where \mathfrak{S}_k denotes the permutation group on $[k]$. In other words, Σ contains all color permutations of $\text{Greedy}(\mathcal{J}_i^f)$. Observe that Σ is closed under taking permutations.

Let now SOL be the solution produced by the algorithm at the beginning of the phase, i.e., after the reset insertion. Let $\text{OPT} \in \Sigma$ be the optimal solution such that $\text{OPT}_{i-1} = \text{SOL}_{i-1}$. One must exist, since we can permute the colors of $\text{Greedy}(\mathcal{J}_i^f)$ in order to match SOL on \mathcal{S}_{i-1} . Let $\tau_S \in \mathfrak{S}_k$ be any permutation such that $\text{SOL}_i = \tau_S \circ \text{OPT}_i$. Observe, that if τ_S can be chosen as identity permutation, `total-recolor`(i, i) is never called in this phase. Hence, we may assume that τ_S is not the identity. So far we have $\text{SOL}_{i-1} = \text{OPT}_{i-1}$ and $\text{SOL}_i = \tau_S \circ \text{OPT}_i$.

Within the phase there are two types of events that affect the coloring maintained by the algorithm on \mathcal{S}_{i-1} and \mathcal{S}_i . Event of type *A* is a call to `total-recolor`(j, k) for $k < i$, which permutes the colors on \mathcal{S}_{i-1} and \mathcal{S}_i with the same permutation. Event of type *B* is a call to `total-recolor`(i, i), which leaves the colors on \mathcal{S}_{i-1} intact while permuting colors on \mathcal{S}_i .

Let us define $\text{SOL}^{(j)}$ to be the solution maintained by the algorithm right after the j 'th event. For some $\sigma, \tau \in \mathfrak{S}_k$ we get $\text{SOL}_{i-1}^{(j)} = \sigma \circ \text{SOL}_{i-1}^{(j-1)}$, $\text{SOL}_i^{(j)} = \sigma \circ \text{SOL}_i^{(j-1)}$ if the j 'th event is of type A and $\text{SOL}_{i-1}^{(j)} = \text{SOL}_{i-1}^{(j-1)}$, $\text{SOL}_i^{(j)} = \tau \circ \text{SOL}_i^{(j-1)}$ if the j 'th event is of type B .

Also, after the j 'th event, we define $\sigma_j, \tau_j \in \mathfrak{S}_k$ to be such that $\text{SOL}_{i-1}^{(j)} = \sigma_j \circ \text{SOL}_{i-1}^{(j-1)}$ and $\text{SOL}_i^{(j)} = \tau_j \circ \text{SOL}_i^{(j-1)}$. Our goal is to obtain $\tau_j^{-1} \circ \sigma_j = \tau_S$ for some j . If that holds then $\text{total-recolor}(i, i)$ is never called again in this phase, because then we have $\text{SOL}_{i-1}^{(j)} = \sigma_j \circ \text{SOL}_{i-1} = \sigma_j \circ \text{OPT}_{i-1}$ and $\text{SOL}_i^{(j)} = \tau_j \circ \text{SOL}_i = \tau_j \circ \tau_S \circ \text{OPT}_i = \sigma_j \circ \text{OPT}_i$. But then there is optimal solution $\sigma_j \circ \text{OPT}$ that certifies that we can recolor \mathcal{J}_i in compliance with $\text{SOL}_{i-1}^{(j)}$ and $\text{SOL}_i^{(j)}$.

Now observe, that if we apply the same permutation $\alpha \in \mathfrak{S}_k$ to both SOL_{i-1}^j and SOL_i^j , i.e., if $\text{SOL}_{i-1}^{(j+1)} = \alpha \circ \text{SOL}_{i-1}^{(j)} = \alpha \circ \sigma_j \circ \text{SOL}_{i-1}$ and $\text{SOL}_i^{(j+1)} = \alpha \circ \text{SOL}_i^{(j)} = \alpha \circ \tau_j \circ \text{SOL}_i$, then $\tau_{j+1}^{-1} \circ \sigma_{j+1} = (\alpha \circ \tau_j)^{-1} \circ \alpha \circ \sigma_j = \tau_j^{-1} \circ \sigma_j$, so permutation $\tau_j^{-1} \circ \sigma_j$ stays the same when permuting colors on \mathcal{S}_{i-1} and \mathcal{S}_i in the same way. Hence, the only way it can change is due to $\text{total-recolor}(i, i)$.

However, if $\text{total-recolor}(i, i)$ is called, that means that the new interval causes that the current coloring $\text{SOL}_{i-1}^{(j)}$ and $\text{SOL}_i^{(j)}$ cannot be used on \mathcal{S}_{i-1} and \mathcal{S}_i now, and hence it cannot be used ever again in the future. This holds because we only add intervals, so any future instance contains the current instance, and any coloring for the future instance is a coloring for the current instance as well. This means that for $k > j$ we have $\tau_k^{-1} \circ \sigma_k \neq \tau_j^{-1} \circ \sigma_j$. For the proof of this fact assume otherwise: $\tau_j^{-1} \circ \sigma_j = \tau_k^{-1} \circ \sigma_k = (\alpha \circ \tau_j)^{-1} \circ \beta \circ \sigma_j$. This implies $\alpha = \beta$ and $\text{SOL}_{i-1}^{(k)} = \alpha \circ \text{SOL}_{i-1}^{(j)}$ and $\text{SOL}_i^{(k)} = \alpha \circ \text{SOL}_i^{(j)}$. But this cannot happen since we already know that the combined coloring $\text{SOL}_{i-1}^{(j)}$ and $\text{SOL}_i^{(j)}$ cannot be used for \mathcal{S}_{i-1} and \mathcal{S}_i , and neither can any permutation of this coloring. But permutation $\sigma_j^{-1} \circ \tau_j$ can only take $k!$ different values until it reaches τ_S . This concludes the proof. \blacktriangleleft

4 Lower bounds for general interval graphs

In this section we provide lower bounds on the recourse budget needed in order to maintain an optimum coloring of an interval graph. The following definition allows us to compare different colorings locally and to formulate necessary conditions for optimum colorings.

Definition 14. Let \mathcal{I} be a set of intervals, let $k \in \mathbb{N}$ be the chromatic number of \mathcal{I} , and let $R = [a, b) \subset \mathbb{R}$. The gap of a set $C \subseteq \mathcal{I}$ of disjoint intervals is given by $\text{gap}_R(C) := |R| - \sum_{I \in C} |R \cap I|$. The total gap of a partition \mathcal{C} of \mathcal{I} into disjoint sets wrt. R is $\text{gap}_R(\mathcal{C}) := \sum_{C \in \mathcal{C}} \text{gap}_R(C)$. The total gap of \mathcal{I} wrt. R is given by $\text{gap}_R(\mathcal{I}) := k \cdot |R| - \sum_{I \in \mathcal{I}} |R \cap I|$.

The following fact provides a formal criterion for optimality of a coloring. Note that in every proper coloring all intervals receiving the same color are disjoint.

Fact 15. We have $\text{gap}_R(\mathcal{I}) = \text{gap}_R(\mathcal{C}^*)$, where \mathcal{C}^* is a partition of \mathcal{I} corresponding to any optimum coloring of \mathcal{I} .

We are now ready to construct an instance that requires many recolorings. The main building block for the bad instance is a *staircase* gadget S_k that guarantees a linear number of recolorings overall (cf. Fig. 3). We will later use multiple copies of this gadget to force $\Omega(\sqrt{n})$ amortized recolorings.

The gadget consists of three sets L, X, R of intervals. We start with an initial configuration of intervals in these sets, which we assume can be colored optimally with k colors without ever recoloring (if an algorithm needs recolorings, this only strengthens our bound). We

17:12 Recoloring Interval Graphs with Limited Recourse Budget



■ **Figure 3** Illustration of the open (left) and closed (right) staircase gadget.

call the initial configuration *open*. Later, we introduce additional intervals in each of the three sets in such a way that the chromatic number increases by exactly one, to $k + 1$, and such that a significant portion of the previously colored intervals need to be recolored in order not to exceed $k + 1$ colors. We refer to the final configuration of the staircase as *closed*. Importantly, we ensure that both in the open and the closed configuration there is a unique way to optimally color the intervals (apart from renaming colors). This ensures that “from the outside” the gadget behaves like a clique of k intervals in the open configuration and a clique of $k + 1$ intervals in the closed configuration.

We start by describing the open (initial) configuration (cf. Fig. 3 (left)). We set $L = \{L_i\}_{i=1}^k := \{[i - \Delta, i)\}_{i=1}^k$, $X = \emptyset$, and $R = \{R_i\}_{i=1}^k := \{[i + \varepsilon, i + \Delta)\}_{i=1}^k$, where $0 < \varepsilon < 1/k$ is sufficiently small and $\Delta \geq k + 1$ is sufficiently large. Observe that the open staircase can be colored with k colors simply by coloring L_i, R_i with color i , and k colors are needed because L and R each are a clique of size k . The total gap in the interval $[1, k + \varepsilon)$ is $\text{gap}_{[1, k + \varepsilon)}(L \cup R) = k\varepsilon < 1$. By Fact 15, no optimal solution with k colors can therefore afford to leave a gap of size 1 or larger in any color. Since L and R each form a clique, assigning the same color to $L_1 = [1 - \Delta, 1)$ and $R_i = [i + \varepsilon, i + \Delta)$ with $i \geq 2$ leads to a gap of $i + \varepsilon - 1 > 1$, and it follows that L_1, R_1 must get the same color. Repeating this argument, so must L_i, R_i for every $i \in \{1, \dots, k\}$. This means that (up to permuting the colors) there is a unique coloring of the open staircase with k colors, as intended.

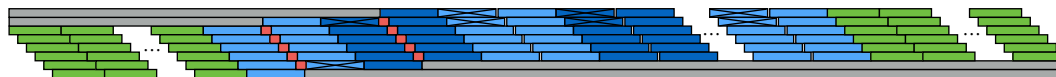
To obtain the closed configuration (cf. Fig. 3 (right)), we add the interval $L_0 := (-\infty, 1 + \varepsilon)$ to L , the interval $R_{k+1} := [k, \infty)$ to R , and the intervals $X = \{X_i\}_{i=1}^{k-1} := \{[i, i + 1 + \varepsilon)\}_{i=1}^{k-1}$. Note that the sets of intervals of the closed staircase can be colored with $k + 1$ colors and zero total gap in the interval $[1, k + \varepsilon)$: we can simply color L_{i-1}, R_i with color i for $i \in \{1, \dots, k + 1\}$ and X_i with color $i + 1$ for $i \in \{1, \dots, k - 1\}$. This means that every coloring with $k + 1$ colors must have zero total gap in the interval $[1, k + \varepsilon)$, by Fact 15. Since every point is the endpoint of at most two intervals in L, X, R , there is a unique way of coloring the closed staircase with $k + 1$ colors, as intended.

Finally, consider the bipartite graph that has the elements of L on one side and the elements of R on the other, with an edge connecting an interval from L to an interval from R if they do not intersect. The staircase matching induces a unique matching in this graph, where each edge selected in the matching corresponds to a color. We call this matching the *stair matching* of S_k and conclude the following lemma.

► **Fact 16.** *The staircase gadget S_k has chromatic number k when open and $k + 1$ when closed. In either configuration there is a unique optimum coloring (up to renaming colors), and the stair matchings of these two colorings are perfect and disjoint.*

Since the stair matchings are disjoint, when adding intervals to obtain the closed staircase from the open one, many intervals need to be recolored.

► **Fact 17.** *When transitioning from the open to the closed staircase gadget, at least k intervals of the open staircase must be recolored to maintain an optimum coloring.*



■ **Figure 4** Illustration of the construction in the proof of Theorem 19 after round 2. Green intervals are (shifted) copies of Z and crossed-out intervals are passive.

We now describe a *connector gadget* C_k that generalizes the interface between consecutive staircase gadgets as well as further gadgets. The connector gadget consists of an *L-connector* and an *R-connector*, and is defined as follows. The L-connector of size k is a set of intervals of the form $\{[a_i, x + i)\}_{i=1}^k$ with $a_i \leq x$, and the R-connector of size k is of the form $\{[x + i, b_i)\}_{i=1}^k$ with $b_i > x + k$. Here $x \in \mathbb{R}$ is an arbitrary offset. Together, the L-connector and R-connector form the connector. Observe that for $i \in \{1, \dots, k\}$ the intervals L_i are an R-connector, and the intervals R_i are an L-connector. The following property of connector gadgets is obvious.

► **Fact 18.** *There is a unique coloring of the connector gadget C_k with k colors (up to renaming colors).*

► **Theorem 19.** *For every $k \in \mathbb{N}$, there is an instance of online interval graph coloring with chromatic number $\Theta(k)$ and $\Theta(k^2)$ vertices that requires an amortized recourse budget of $\Omega(k)$ to maintain an optimum solution.*

Proof. We fix any number $k \in \mathbb{N}$ and any online coloring algorithm. We start by introducing a large set of intervals offline that we allow the algorithm to color in a batch (i.e., not online and without need to recolor), before introducing additional intervals online that each require significant recoloring.

We first describe the offline intervals. We introduce multiple gadgets that each start with an R-connector and end with an L-connector. In the following, each gadget (after the first) is shifted to the right, such that it forms a connector gadget with the previous gadget. Let $Z := \{[i, k + i)\}_{i=1}^k$, i.e., Z is both an R- and an L-connector. We introduce k copies of Z , each shifted as described (green intervals in Fig. 4).

Since the copies of Z form a chain of connector gadgets, by Fact 18, there is a unique way to color these gadgets with k colors. We further introduce k shifted open staircase gadgets and then another k shifted copies of Z . Overall, our construction so far uses $n_{\text{open}} = 4k^2$ intervals, has chromatic number k , and, by Fact 16 and Fact 18, there is a unique way to color all intervals with k colors.

We now present additional sets of intervals online in k rounds. In each round, we close the leftmost open staircase gadget by introducing $k + 1$ new intervals. In each round the new intervals of the form L_0 and R_{k+1} overlap all intervals outside the staircase being closed. Thus, while the chromatic number increases by one, the effective number of available colors in all gadgets to the right remains unchanged. We call an interval of a staircase *passive* if it is part of an R-connector (resp. L-connector) and shares a color with any interval of the form L_0 (resp. R_{k+1}), and *active* otherwise. This means in particular that in each round a single interval of every open staircase becomes passive. By Corollary 17, at least k intervals of a staircase need to be recolored when it is being closed. Since each active interval is part of a connector gadget outside the staircase, and since each such connector gadget and every other staircase must be colored in a unique way (Fact 16 and Fact 18), recoloring an active interval requires to recolor all other intervals of the same color to the left or to the right of the staircase. Thus, in round i , at least $k - i + 1$ active intervals need to be recolored, each affecting k copies of Z , such that the total number of intervals that need

17:14 Recoloring Interval Graphs with Limited Recourse Budget

to be recolored is at least $(k - i + 1)k$. After closing the staircase, by Fact 16, there is again a unique way to color it. This means that we can repeat the process with the next staircase, restricting everything to the colors that are occupied by the current gadget, and so on. Overall, the number of intervals that need to be recolored in k rounds is at least $\sum_{i=1}^k (k - i + 1)k = k^3 - k^2(k + 1)/2 + k^2 = \Omega(k^3)$.

Overall, we introduce $k + 1$ new intervals in each round, so the total number of intervals is $n = n_{\text{open}} + k(k + 1) = 5k^2 + k$. The chromatic number increases by one in every round, hence the chromatic number of the final graph is $k' = 2k$. The amortized recourse budget the algorithm needs thus is $\Omega(k^3/n) = \Omega(k)$, as claimed. ◀

The next statements follow from Theorem 19 by setting $k = \Theta(\sqrt{n})$, and by observing that we can always add isolated vertices without affecting k .

► **Corollary 20.** *Maintaining an optimum coloring of an interval graph online, requires an amortized recourse budget of $\Omega(\sqrt{n})$ in general (when k is not fixed).*

► **Corollary 21.** *Maintaining an optimum coloring of an interval graph with chromatic number $k \in \mathcal{O}(\sqrt{n})$ online, requires an amortized recourse budget of $\Omega(k)$ in general.*

5 Trading off recourse budget with query times

Up to this point we worked in a model where we need to maintain the coloring explicitly, i.e., after each insertion of an interval we need to recolor every interval whose color changes. We showed an algorithm, which achieves this by recoloring amortized $\mathcal{O}(k \cdot k! \cdot \sqrt{n})$ intervals, and for this algorithm an efficient implementation is not obvious. In this section we give an efficient algorithm maintaining the optimum coloring, but we relax the model. So far we insisted on recoloring all intervals immediately. This requirement allows us to retrieve the color of any interval in constant time, and is moreover crucial for some applications. In this section we do not focus on maintaining an explicit coloring, but rather we design a coloring oracle: a data structure that can be queried for the color of an interval. Our data structure supports interval additions in $\mathcal{O}(k^2 \log^3 n)$ amortized time, and it answers queries for a color of a particular interval in $\mathcal{O}(\log n)$ time. Between two consecutive updates it answers queries consistently with some optimal proper coloring. We only sketch the data structure here, and leave some details and the formal proof of the following theorem to Appendix A.

► **Theorem 22.** *There is a dynamic datastructure that stores a k -colorable set of intervals \mathcal{I} and returns the color of any $I \in \mathcal{I}$ according to an optimum proper coloring of \mathcal{I} in $\mathcal{O}(\log n)$ time. Furthermore, it needs $\mathcal{O}(k^2 \log^3 n)$ amortized time to insert a new interval.*

We store the intervals of the instance \mathcal{I} in a modified *interval tree* [13]. That is, we maintain a binary search tree T , for which each node v stores the x -coordinate $l_v \in \mathbb{R}$ of a vertical line and a subset $\mathcal{S}_v \subseteq \mathcal{I}$ of intervals. For a node v of T , let T_v be the subtree of T rooted at v , and let \mathcal{I}_v contain all intervals stored in the sets \mathcal{S}_u for nodes u of T_v . We say that an interval I is *stored in T_v* if T_v has a node u such that $I \in \mathcal{S}_u$, i.e., $I \in \mathcal{I}_v$. The tree T has the following properties.

1. If $\mathcal{I}_v = \emptyset$ then v is a leaf of T with undefined value l_v and empty set \mathcal{S}_v .
2. Otherwise, T_v has a defined value $l_v \in \mathbb{R}$ and two child nodes x and y in T , for which the (defined) values l_u of all nodes u of T_x are smaller than l_v , while the (defined) values l_u of all nodes u of T_y are larger than l_v . The trees T_x and T_y are called the *left* and *right subtree* of T_v , respectively.

3. The set $\mathcal{S}_v = \{I \in \mathcal{I}_v \mid \text{beg}(I) \leq l_v \leq \text{end}(I)\}$ contains all intervals of \mathcal{I}_v intersecting l_v . The left and right subtrees T_x and T_y of T_v recursively store all intervals in $\mathcal{I}_x = \{I \in \mathcal{I}_v \mid \text{end}(I) < l_v\}$ and $\mathcal{I}_y = \{I \in \mathcal{I}_v \mid \text{beg}(I) > l_v\}$, respectively.

The sets \mathcal{S}_v stored in all nodes v of T partition \mathcal{I} , and an interval $I \in \mathcal{I}$ is stored in the highest node v of T for which I contains l_v . Therefore each set \mathcal{S}_v is a separator of the intervals \mathcal{I}_v stored in T_v , i.e., no interval from the left subtree of T_v overlaps with an interval from the right subtree of T_v . Furthermore, the intervals of any set \mathcal{S}_v form a clique, as they all intersect l_v , and thus at most k intervals are stored in a node v if \mathcal{I} is k -colorable.

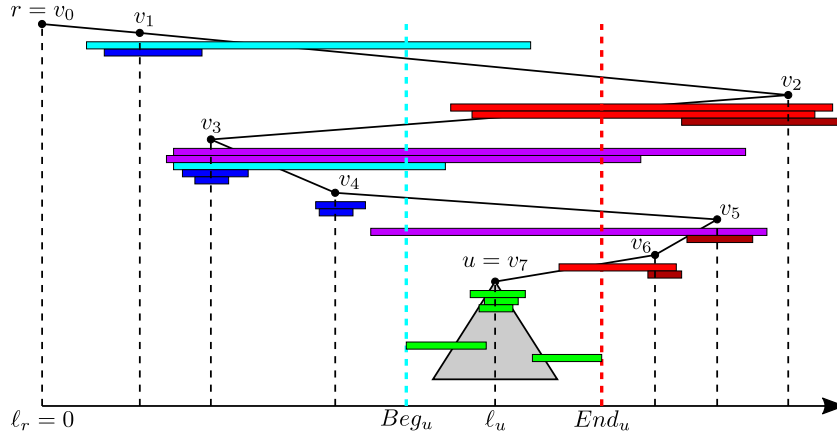
For reasons that will become apparent later, at all times we will make sure that the root r of T has x -coordinate $l_r = 0$, and we assume w.l.o.g. that $\text{beg}(I) > 0$ for all $I \in \mathcal{I}$. This means that all intervals of \mathcal{I} are stored in the right subtree of T . We will also make sure that for any node $v \neq r$ different from the root, if $\mathcal{S}_v = \emptyset$ then v is a leaf of T . This ensures that the number of nodes of T is linear in $n = |\mathcal{I}|$.

Instead of storing the color of each interval explicitly, we associate a permutation $\tau_e \in \mathfrak{S}_k$ with each edge e of the search tree T . Also, for each node v of T we store the intervals of the set \mathcal{S}_v in a fixed order, so that $\mathcal{S}_v = \{I_1, \dots, I_j\}$ for $j \leq k$. The color of an interval $I_i \in \mathcal{S}_v$ is obtained by applying the permutations along the path P_v from the root r of T to v to the index i . That is, let e_1, e_2, \dots, e_h be the sequence of edges of P_v such that e_1 is incident to r (note that e_1 connects r to the right subtree of T by our assumption that $l_r = 0$). We denote the composite permutation along the path P_v by $\sigma_{e_h} = (\tau_{e_h} \circ \dots \circ \tau_{e_2} \circ \tau_{e_1})$, and the color of $I_i \in \mathcal{S}_v$ is $\sigma_{e_h}(i)$. Thus the color of any interval can be retrieved in time linear in the height of the tree, by first finding its index i in the node storing it and then following the search path back to the root to compute the image of i in the composite permutation defining its color. It is also clear that there exist permutations for the edges that imply a proper k -coloring of the intervals if \mathcal{I} is k -colorable. In fact, only the permutation τ_{e_h} of the last edge e_h on P_v for some particular node v needs to be picked in relation to all previous permutations along P_v , so that the indices of \mathcal{S}_v are permuted according to a fixed proper k -coloring.

To obtain logarithmic query times, we make sure that the tree T is α -balanced [13] at all times. That is, let $n_v = |\mathcal{I}_v|$ be the number of intervals stored in subtree T_v rooted at v , and let α be a fixed constant such that $1/2 < \alpha < 1$. For any subtree $T_v \neq T$ (i.e., $v \neq r$) we maintain the property that $\max\{n_v^-, n_v^+\} \leq \lceil \alpha n_v \rceil$, where n_v^- and n_v^+ are the number of intervals stored in the left and right subtrees of T_v , respectively. As an easy consequence we get that the height of T is $\log_{1/\alpha}(n) + \mathcal{O}(1) = \mathcal{O}(\log n)$. To maintain this invariant, we store n_v in node v .

5.1 Updates

We now describe how to update the search tree T and the permutations on its edges, so that the colors induced by the permutations form a proper k -coloring and the tree is α -balanced at all times. When a new interval I_{new} arrives, it is stored in the interval tree T in the usual way [13]. That is, we follow the search path for I_{new} starting from the root. As soon as we encounter a node v in T such that I_{new} belongs to the set \mathcal{S}_v (because $l_v \in I_{\text{new}}$), we add I_{new} to \mathcal{S}_v . The index i of the new interval I_{new} in \mathcal{S}_v is the highest available, i.e., $i = |\mathcal{S}_v|$ when $I_{\text{new}} \in \mathcal{S}_v$. Additionally, we increase the variables n_u along the nodes u of the path P_v from v to the root of T by one each, to count the new interval I_{new} in the subtrees T_u . If no node v for which $l_v \in I_{\text{new}}$ is found, let w be the leaf of T at the end of the search path P_w for I_{new} . We set $l_w = \text{beg}(I_{\text{new}})$, and add I_{new} as the only interval in the set \mathcal{S}_w . We also create two new leaves and set them as the new left and right subtrees of w . Again, we increase the variables n_u along the nodes u of P_w .



■ **Figure 5** The path P_u with nodes v_0 to v_7 from the root r of the search tree T to the node u . The bars represent intervals, which are stored in the highest node w for which they contain l_w (black dashed lines). The subtree T_u is shaded in grey and stores \mathcal{I}_u (green intervals). The leftmost and rightmost points of these intervals are beg_u and end_u (blue and red dotted lines), which define the sets \mathcal{L}_u and \mathcal{R}_u (including the light blue and light red intervals, respectively). Some intervals can be in the intersection of \mathcal{L}_u and \mathcal{R}_u (purple intervals). The remaining intervals are either to the left of beg_u or to the right of end_u (dark blue and dark red intervals, respectively). In this example, $L = \{v_2v_3, v_6v_7\}$ and $R = \{v_1v_2, v_4v_5\}$.

When adding I_{new} the tree T may become unbalanced, i.e., there may be a node $u \neq r$ of T for which $\max\{n_u^-, n_u^+\} > \lceil \alpha n_u \rceil$. Note that u must be on the path P_v from the root r to the node v into which I_{new} was added. To make T α -balanced again, we identify the closest such node u to the root. We then rebalance T_u by first retrieving all intervals \mathcal{I}_u stored in T_u , and then sorting all the endpoints $\text{beg}(I)$ and $\text{end}(I)$ of the intervals $I \in \mathcal{I}_u$. Next a new balanced tree is built to take the place of T_u , using the standard recursive procedure to create interval trees. That is, it takes as input a set of intervals \mathcal{I}' (initially set to \mathcal{I}_u) and their sorted endpoints. The procedure creates a new root vertex w of the current tree, and sets l_w to the median of all endpoints of \mathcal{I}' . It then identifies the set \mathcal{S}_w containing all intervals of \mathcal{I}' that intersect l_w . The left and right subtrees are then recursively built for the subsets of \mathcal{I}' of all intervals to the left of l_w and to the right of l_w , respectively. In case $\mathcal{I}' = \emptyset$, a leaf is created and the recursion terminates. Note that the number of endpoints of value less than the median is at most $|\mathcal{I}'|$, as there are $2|\mathcal{I}'|$ endpoints. Since each interval has two endpoints, the left subtree will contain at most $|\mathcal{I}'|/2$ intervals, and analogously this is also true for the right subtree. Therefore this results in a new tree T_u for which $\max\{n_w^-, n_w^+\} \leq n_w/2$ for every node w of T_u , i.e., this tree is perfectly balanced.

To update the permutations we need some definitions (cf. Figure 5). For any node u of T , let $\text{beg}_u = \min\{\text{beg}(I) \mid I \in \mathcal{I}_u\}$ be the left-most point of any interval stored in T_u , and accordingly let $\text{end}_u = \max\{\text{end}(I) \mid I \in \mathcal{I}_u\}$ be the right-most point. We then define the two sets $\mathcal{L}_u = \{I \in \mathcal{I} \mid \text{beg}(I) \leq \text{beg}_u < \text{end}(I)\} \setminus \mathcal{I}_u$ and $\mathcal{R}_u = \{I \in \mathcal{I} \mid \text{beg}(I) \leq \text{end}_u < \text{end}(I)\} \setminus \mathcal{I}_u$ of intervals not stored in T_u but containing beg_u or end_u , respectively. Note that each interval in \mathcal{L}_u or \mathcal{R}_u must be stored in some internal node $w \notin \{r, u\}$ of P_u (meaning that it is contained in \mathcal{S}_w), as \mathcal{S}_w is non-empty and separates \mathcal{I}_u from the intervals stored in the (left or right) subtree of T_w not containing u . Let also L and R be the set of edges of P_u that cross the boundary defined by end_u in the sense that $xy \in L$ if x is the parent of y and $l_x \geq \text{end}_u > l_y$, and $xy \in R$ if x is the parent of y and $l_x < \text{end}_u \leq l_y$.

The algorithm performs the following steps after I_{new} was added to the set \mathcal{S}_v .

1. If there is a node $w \neq r$ on P_v for which $\max\{n_w^-, n_w^+\} > \lceil \alpha n_w \rceil$, then let w be the closest such node to the root r . Rebuild the subtree T_w to obtain a new perfectly balanced subtree T_w . In this case set $u = w$ in the following, while otherwise $u = v$.
2. First retrieve beg_u and end_u , and then \mathcal{L}_u and \mathcal{R}_u together with all colors of intervals in \mathcal{L}_u and \mathcal{R}_u using the permutations stored on the edges of P_u .
3. Starting with the current coloring of \mathcal{L}_u , use the greedy algorithm to color $\mathcal{I}_u \cup \mathcal{R}_u$ with at most k colors. As the intervals in \mathcal{R}_u form a clique (they all contain end_u), there is a permutation $\mu \in \mathfrak{S}_k$ mapping the old colors of \mathcal{R}_u to its new colors.
4. The permutations stored on edges e of P_u and T_u are updated to encode the new colors for the intervals in $\mathcal{I}_u \cup \mathcal{R}_u$ as follows. Let σ_e and σ'_e be the composite permutations along the path from the root to edge e before and after the update, respectively.
 - a. For any edge e of P_u that is neither in L nor in R , the permutation τ_e remains unchanged.
 - b. For any $e \in L$ the permutation τ_e is chosen such that $\sigma'_e = \sigma_e$.
 - c. For any $e \in R$ the permutation τ_e is chosen such that $\sigma'_e = \sigma_e \circ \mu$ for the permutation μ of step 3.
 - d. Permutations τ_e for edges e of T_u are simply chosen so that the σ_e induce the new colors of \mathcal{I}_u .

The proof of correctness and runtime analysis of this data structure can be found in Appendix A.

References

- 1 Patrick Baier, Bartłomiej Bosek, and Piotr Micek. On-line chain partitioning of up-growing interval orders. *Order*, 24(1):1–13, 2007. doi:10.1007/s11083-006-9050-0.
- 2 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Algorithms and data structures*, volume 10389 of *Lecture Notes in Comput. Sci.*, pages 97–108. Springer, Cham, 2017. doi:10.1007/978-3-319-62127-2_9.
- 3 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–959. SIAM, Philadelphia, PA, 2018. doi:10.1137/1.9781611975031.61.
- 4 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*, pages 1–20. SIAM, Philadelphia, PA, 2018. doi:10.1137/1.9781611975031.1.
- 5 Bartłomiej Bosek, Stefan Felsner, Kamil Kloch, Tomasz Krawczyk, Grzegorz Matecki, and Piotr Micek. On-line chain partitions of orders: a survey. *Order*, 29(1):49–73, 2012. doi:10.1007/s11083-011-9197-1.
- 6 Bartłomiej Bosek, H. A. Kierstead, Tomasz Krawczyk, Grzegorz Matecki, and Matthew E. Smith. An easy subexponential bound for online chain partitioning. *Electron. J. Combin.*, 25(2):Paper No. 2.28, 23, 2018. doi:10.37236/7231.
- 7 Bartłomiej Bosek and Tomasz Krawczyk. A subexponential upper bound for the on-line chain partitioning problem. *Combinatorica*, 35(1):1–38, 2015. doi:10.1007/s00493-014-2908-7.
- 8 Bartłomiej Bosek and Tomasz Krawczyk. On-line partitioning of width w posets into $w^{O(\log \log w)}$ chains. *CoRR*, arXiv:1810.00270, 2018. arXiv:1810.00270.

- 9 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014*, pages 384–393. IEEE Computer Soc., Los Alamitos, CA, 2014. doi:10.1109/FOCS.2014.48.
- 10 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. Shortest augmenting paths for online matchings on trees. *Theory Comput. Syst.*, 62(2):337–348, 2018. doi:10.1007/s00224-017-9838-x.
- 11 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In *LATIN 2018: Theoretical informatics*, volume 10807 of *Lecture Notes in Comput. Sci.*, pages 201–216. Springer, Cham, 2018. doi:10.1007/978-3-319-77404-6_1.
- 12 Marek Chrobak and Maciej Ślusarek. On some packing problem related to dynamic storage allocation. *RAIRO Inform. Théor. Appl.*, 22(4):487–499, 1988.
- 13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, third edition, 2009.
- 14 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to Adwords. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 982–991. ACM, New York, 2008.
- 15 Magnús M. Halldórsson and Christian Konrad. Improved distributed algorithms for coloring interval graphs with application to multicoloring trees. In *Structural information and communication complexity*, volume 10641 of *Lecture Notes in Comput. Sci.*, pages 247–262. Springer, Cham, 2017. doi:10.1007/978-3-319-72050-0_15.
- 16 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.
- 17 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for MaxClique, chromatic number and Min-3Lin-Deletion. In *Automata, languages and programming. Part I*, volume 4051 of *Lecture Notes in Comput. Sci.*, pages 226–237. Springer, Berlin, 2006. doi:10.1007/11786986_21.
- 18 Henry A. Kierstead. An effective version of Dilworth’s theorem. *Trans. Amer. Math. Soc.*, 268(1):63–77, 1981. doi:10.2307/1998337.
- 19 Henry A. Kierstead. Recursive ordered sets. In *Combinatorics and ordered sets (Arcata, Calif., 1985)*, volume 57 of *Contemp. Math.*, pages 75–102. Amer. Math. Soc., Providence, RI, 1986. doi:10.1090/conm/057/856233.
- 20 Henry A. Kierstead and William T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congr. Numer.*, 33:143–153, 1981.
- 21 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: efficient dynamic algorithms for the Steiner tree. In *STOC’15—Proceedings of the 2015 ACM Symposium on Theory of Computing*, pages 11–20. ACM, New York, 2015.
- 22 Dániel Marx. Parameterized coloring problems on chordal graphs. *Theoret. Comput. Sci.*, 351(3):407–424, 2006. doi:10.1016/j.tcs.2005.10.008.
- 23 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *Automata, languages, and programming. Part I*, volume 7391 of *Lecture Notes in Comput. Sci.*, pages 689–700. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_58.
- 24 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. AdWords and generalized online matching. *J. ACM*, 54(5):Art. 22, 19, 2007. doi:10.1145/1284320.1284321.
- 25 Cara Monical and Forrest Stonedahl. Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO ’14*, pages 469–476, New York, NY, USA, 2014. ACM. doi:10.1145/2576768.2598233.

- 26 Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Inform. Process. Lett.*, 37(1):21–25, 1991. doi:10.1016/0020-0190(91)90245-D.
- 27 Carlos A. S. Oliveira and Panos M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.*, 32(8):1953–1981, 2005. doi:10.1016/j.cor.2003.12.007.
- 28 Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *2011 International Conference on Communications, Computing and Control Applications, CCCA 2011*, pages 1–5, 2011. doi:10.1109/CCCA.2011.6031437.
- 29 Jean-Jacques Pansiot and Dominique Grad. On routes and multicast trees in the internet. *SIGCOMM Comput. Commun. Rev.*, 28(1):41–50, 1998. doi:10.1145/280549.280555.
- 30 Davy Preuveneers and Yolande Berbers. ACODYGRA: An agent algorithm for coloring dynamic graphs. In *6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 381–390, 2004. URL: <https://lirias.kuleuven.be/1654548>.
- 31 Fred S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968)*, pages 139–146. Academic Press, New York, 1969.
- 32 Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16*, pages 30:1–30:12, Piscataway, NJ, USA, 2016. IEEE Press. URL: <http://dl.acm.org/citation.cfm?id=3014904.3014945>.
- 33 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th European Symposium on Algorithms*, volume 112 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 72, 16. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- 34 Mario Valencia-Pabon. Revisiting Tucker’s algorithm to color circular arc graphs. *SIAM J. Comput.*, 32(4):1067–1072, 2003. doi:10.1137/S0097539700382157.
- 35 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Effective and efficient dynamic graph coloring. *Proc. VLDB Endow.*, 11(3):338–351, 2017. doi:10.14778/3157794.3157802.
- 36 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, New York, 2006. doi:10.1145/1132516.1132612.

A Correctness and runtime of the balanced interval tree

Recall that the interval tree T has the following properties.

1. If $\mathcal{I}_v = \emptyset$ then v is a leaf of T with undefined value l_v and empty set \mathcal{S}_v .
2. Otherwise, T_v has a defined value $l_v \in \mathbb{R}$ and two child nodes x and y in T , for which the (defined) values l_u of all nodes u of T_x are smaller than l_v , while the (defined) values l_u of all nodes u of T_y are larger than l_v . The trees T_x and T_y are called the *left* and *right subtree* of T_v , respectively.
3. The set $\mathcal{S}_v = \{I \in \mathcal{I}_v \mid \text{beg}(I) \leq l_v \leq \text{end}(I)\}$ contains all intervals of \mathcal{I}_v intersecting l_v . The left and right subtrees T_x and T_y of T_v recursively store all intervals in $\mathcal{I}_x = \{I \in \mathcal{I}_v \mid \text{end}(I) < l_v\}$ and $\mathcal{I}_y = \{I \in \mathcal{I}_v \mid \text{beg}(I) > l_v\}$, respectively.

To insert a new interval I_{new} , we follow the search path for I_{new} starting from the root. As soon as we encounter a node v in T such that I_{new} belongs to the set \mathcal{S}_v (because $l_v \in I_{\text{new}}$), we add I_{new} to \mathcal{S}_v . The index i of the new interval I_{new} in \mathcal{S}_v is the highest available, i.e., $i = |\mathcal{S}_v|$ when $I_{\text{new}} \in \mathcal{S}_v$. Additionally, we increase the variables n_u along the nodes u of the path P_v from v to the root of T by one each, to count the new interval I_{new} in the subtrees T_u . If no node v for which $l_v \in I_{\text{new}}$ is found, let w be the leaf of T at the end of the search

path P_w for I_{new} . We set $l_w = \text{beg}(I_{\text{new}})$, and add I_{new} as the only interval in the set \mathcal{S}_w . We also create two new leaves and set them as the new left and right subtrees of w . Again, we increase the variables n_u along the nodes u of P_w .

In order to rebalance a subtree T_u rooted at a node u we use the following standard procedure. It takes as input a set of intervals \mathcal{I}' (initially set to \mathcal{I}_u) and their sorted endpoints. The procedure creates a new root vertex w of the current tree, and sets l_w to the median of all endpoints of \mathcal{I}' . It then identifies the set \mathcal{S}_w containing all intervals of \mathcal{I}' that intersect l_w . The left and right subtrees are then recursively built for the subsets of \mathcal{I}' of all intervals to the left of l_w and to the right of l_w , respectively. In case $\mathcal{I}' = \emptyset$, a leaf is created and the recursion terminates. Note that the number of endpoints of value less than the median is at most $|\mathcal{I}'|$, as there are $2|\mathcal{I}'|$ endpoints. Since each interval has two endpoints, the left subtree will contain at most $|\mathcal{I}'|/2$ intervals, and analogously this is also true for the right subtree. Therefore this results in a new tree T_u for which $\max\{n_w^-, n_w^+\} \leq n_w/2$ for every node w of T_u , i.e., this tree is perfectly balanced.

We will now prove the correctness of the algorithm, and later turn to analyzing its amortized runtime.

► **Lemma 23.** *The algorithm maintains an α -balanced interval tree T for \mathcal{I} for which the permutations stored on the edges induce a proper k -coloring of \mathcal{I} , if k is the chromatic number of \mathcal{I} .*

Proof. That the algorithm maintains an α -balanced tree is clear from step 1 and the procedure to rebalance subtrees. That it is an interval tree follows from the fact that adding I_{new} to the first node along the search path for I_{new} of T will store it in the highest node v of T for which I_{new} contains l_v , as required. Furthermore, this property is also maintained when rebalancing a subtree T_w . As no interval will ever be added to the set \mathcal{S}_r of the root r (assuming $\text{beg}(I) > 0$ for all $I \in \mathcal{I}$) and since $T_r = T$ will never be considered for rebalancing, we maintain the invariant that $l_r = 0$ and all of \mathcal{I} is stored in the right subtree of T . Finally, when adding a new interval to a node or rebalancing a subtree, any node v with $\mathcal{S}_v = \emptyset$ will be a leaf of T , except for the root.

To prove that the coloring induced by the permutations of T 's edges is a proper k -coloring, we proceed by induction. The base case is when the tree does not store any intervals, which is trivial. Now consider one step of the algorithm in which some interval I_{new} is added to T , and let u be the node operated on during the execution, i.e., $u = w$ if w is rebalanced and then recolored, or $u = v$ if no subtree needs to be rebalanced and I_{new} is added to \mathcal{S}_v . The main observation is that \mathcal{L}_u and \mathcal{R}_u form separators. More concretely, let $\mathcal{L}_u^- = \{I \in \mathcal{I} \mid \text{end}(I) < \text{beg}_u\}$ and $\mathcal{R}_u^+ = \{I \in \mathcal{I} \mid \text{beg}(I) > \text{end}_u\}$, and note that \mathcal{I} is partitioned into \mathcal{L}_u^- , \mathcal{I}_u , \mathcal{R}_u^+ , and $\mathcal{L}_u \cup \mathcal{R}_u$ (\mathcal{L}_u and \mathcal{R}_u may share some intervals). For any $I \in \mathcal{I}_u \cup (\mathcal{R}_u \setminus \mathcal{L}_u) \cup \mathcal{R}_u^+$ we have $\text{beg}_u \leq \text{beg}(I)$, while for any $I \in \mathcal{I}_u \cup (\mathcal{L}_u \setminus \mathcal{R}_u) \cup \mathcal{L}_u^-$ we have $\text{end}_u \geq \text{end}(I)$. This means that \mathcal{L}_u separates \mathcal{L}_u^- from $\mathcal{I}_u \cup (\mathcal{R}_u \setminus \mathcal{L}_u) \cup \mathcal{R}_u^+$, and similarly \mathcal{R}_u separates \mathcal{R}_u^+ from $\mathcal{I}_u \cup (\mathcal{L}_u \setminus \mathcal{R}_u) \cup \mathcal{L}_u^-$. Thus a k -coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u$ and a k -colouring of $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ together form a k -coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$, if the two given colorings agree on the colors of the separator \mathcal{L}_u . Furthermore, a k -coloring of $\mathcal{R}_u \cup \mathcal{R}_u^+$ together with a k -coloring of $\mathcal{L}_u^- \cup \mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ forms a k -coloring of \mathcal{I} if the two given colorings agree on the colors of \mathcal{R}_u . Hence if we separately prove that the permutations induce a proper k -coloring for each of the three sets $\mathcal{L}_u^- \cup \mathcal{L}_u$, $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$, and $\mathcal{R}_u \cup \mathcal{R}_u^+$, then \mathcal{I} is properly k -colored.

Let I be any interval from $\mathcal{L}_u^- \cup \mathcal{L}_u$, w be the node of T storing I , and e be the last edge of P_w , i.e., which is farthest from the root r of T . If $I \in \mathcal{L}_u^-$, then no edge of P_w can be from T_u , by the above observation that \mathcal{L}_u separates \mathcal{L}_u^- from \mathcal{I}_u . The same is true for P_w if

$I \in \mathcal{L}_u$, since \mathcal{L}_u contains no interval from \mathcal{I}_u by definition. In case no edge of P_w belongs to L or R , according to step 4 every edge f of P_w stores the same permutation τ_f before and after I_{new} was added. This implies $\sigma'_e = \sigma_e$ for the respective composite permutations σ'_e and σ_e along P_w before and after the update. Otherwise, let xy be the farthest edge of P_w from the root r that belongs to $L \cup R$, where x is the parent of y . If $xy \in L$ then $\sigma'_{xy} = \sigma_{xy}$ by step 4, while τ_f is unchanged on any edge f of P_w that is farther than y from the root. Thus if π_{yw} is the composite permutation along P_w from y to w (with π_{yw} being the identity permutation in the trivial case when $y = w$) we obtain $\sigma'_e = \pi_{yw} \circ \sigma'_{xy} = \pi_{yw} \circ \sigma_{xy} = \sigma_e$. For the last case $xy \in R$, note that since T is a search tree, it must be that $l_z \geq \text{end}_u$ for any node z after y on the search path P_w : otherwise some edge after y on P_w would cross the boundary end_u , i.e., it would be in L , contradicting the fact that xy is the last edge of P_w that is in $L \cup R$. Hence for $z = w$ we obtain $\text{end}(I) \geq l_w \geq \text{end}_u$. But as $I \in \mathcal{L}_u \cup \mathcal{L}_u^-$ we also get $\text{beg}(I) \leq \text{beg}_u \leq \text{end}_u$ and so $I \in \mathcal{L}_u \cap \mathcal{R}_u$. Therefore the permutation μ of step 3 maps the color of I to itself, and if I is the i th interval of \mathcal{S}_w , by our choice of τ_{xy} in step 4 we get $\sigma'_e(i) = (\pi_{yw} \circ \sigma'_{xy})(i) = (\pi_{yw} \circ \sigma_{xy} \circ \mu)(i) = (\sigma_e \circ \mu)(i) = \sigma_e(i)$. In conclusion, every interval of $\mathcal{L}_u \cup \mathcal{L}_u^-$ has the same color before and after inserting I_{new} , and thus $\mathcal{L}_u \cup \mathcal{L}_u^-$ is properly k -colored by induction.

Next consider an interval I from $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$. We already know that if $I \in \mathcal{L}_u$ then it keeps its color from before the update, i.e., the permutations on T 's edges induce the same color of I that the greedy algorithm assigns to it. By step 4, any interval of \mathcal{I}_u (including I_{new}) also obtains the colors assigned to it by the greedy algorithm. If $I \in \mathcal{R}_u \setminus \mathcal{L}_u$ then $\text{beg}(I) > \text{beg}_u$ and $I \notin \mathcal{I}_u$. Thus the node w of T storing I is not in T_u . Furthermore, following the search path P_w from the root r must end in a node w for which $l_w > \text{end}_u$, if w is not in T_u and $l_w \geq \text{beg}(I) > \text{beg}_u$. As a consequence, P_w has some edge of R , since $l_r = 0$ and thus following the search path P_w there must be some edge of P_w that crosses end_u in order to reach w . Furthermore, if xy is the edge of P_w that lies in R and is farthest from the root, where x is the parent of y , then no edge of P_w between y and w can belong to L , as such an edge would cross over to the left of end_u but $l_w > \text{end}_u$. Hence by step 4 all edges f of P_w between y and w maintain their permutations τ_f during the update. Let e be the last edge of P_w and let π_{yw} denote the composite permutation along P_w from y to w , which is the identity permutation if $y = w$. By the choice of τ_{xy} in step 4, we have $\sigma'_e = \pi_{yw} \circ \sigma'_{xy} = \pi_{yw} \circ \sigma_{xy} \circ \mu = \sigma_e \circ \mu$. Thus the colors of all intervals of $\mathcal{R}_u \setminus \mathcal{L}_u$ are permuted according to μ , which by definition of μ in step 3 then means that all of $\mathcal{L}_u \cup \mathcal{I}_u \cup \mathcal{R}_u$ is colored according to the greedy algorithm. This implies a proper k -coloring of this set, due to the correctness of the greedy algorithm.

For the last set $\mathcal{R}_u \cup \mathcal{R}_u^+$ we already know that any interval I from \mathcal{R}_u is colored according to the permutation $\sigma_e \circ \mu$, if e is the last edge of the path P_w to the node w storing I (we argued this separately for $I \in \mathcal{R}_u \setminus \mathcal{L}_u$ and $I \in \mathcal{R}_u \cap \mathcal{L}_u$ above). This is also true for any $I \in \mathcal{R}_u^+$, since the premise is the same as for intervals from $\mathcal{R}_u \setminus \mathcal{L}_u$: we have $\text{beg}(I) > \text{end}_u \geq \text{beg}_u$ and $I \notin \mathcal{I}_u$ as \mathcal{R}_u separates \mathcal{I}_u from \mathcal{R}_u^+ . Therefore the colors of intervals in $\mathcal{R}_u \cup \mathcal{R}_u^+$ are permuted by μ relative to the colors induced by the permutations of the edges of T before the update. Hence $\mathcal{R}_u \cup \mathcal{R}_u^+$ is properly k -colored by induction, which concludes the proof. \blacktriangleleft

In order to bound the amortized runtime of one step when adding an interval I_{new} to the search tree T , we first determine the actual runtime.

► Lemma 24. *Let u be the node of T for which the update algorithm is run, let p_u be the number of nodes on the path P_u from the root of T to u , and let t_u be the number of nodes of the subtree T_u of T rooted at u . Then the update algorithm takes $\mathcal{O}(k(t_u + p_u) \log n)$ time.*

17:22 Recoloring Interval Graphs with Limited Recourse Budget

Proof. Finding the node v in which to store I_{new} and a node w on P_v for which T_w needs to be rebalanced is linear in the height of T , and can thus be done in $\mathcal{O}(\log n)$ time as T is α -balanced. If $n_w = |\mathcal{I}_w|$ denotes the number of intervals stored in T_w , it is known that rebalancing T_w can be done in $\mathcal{O}(n_w \log n_w)$ time [13] for step 1. Next we set $u = w$ or $u = v$ depending on whether some tree was rebalanced. As $|\mathcal{S}_x| \leq k$ for every node x of T , we have $n_u \leq kt_u$, and the time to rebalance can be bounded by $\mathcal{O}(kt_u \log n)$.

Retrieving beg_u and end_u in step 2 needs linear time in the height of the tree T_u , i.e., it can be done in $\mathcal{O}(\log n)$ time. If the number of nodes of P_u is denoted by p_u then the number of intervals stored in nodes of P_u is at most kp_u , by the observation that each set stored in a node forms a clique in a k -colorable graph. Thus retrieving \mathcal{L}_u and \mathcal{R}_u together with their colors takes $\mathcal{O}(kp_u)$ time if traversing P_u bottom up towards the root and in each step computing the composite permutation σ_e for each edge e of P_u from the permutation $\sigma_{e'}$ of the previous edge e' .

For step 3, also the set \mathcal{I}_u needs to be retrieved, which can be done in $\mathcal{O}(n_u)$ time given u . The runtime of the greedy algorithm [26] to color $\mathcal{I}_u \cup \mathcal{R}_u$ given the colors of \mathcal{L}_u is $\mathcal{O}((n_u + k) \log(n_u + k))$ as both \mathcal{L}_u and \mathcal{R}_u form a clique in a k -colorable graph. Finding the permutation μ takes $\mathcal{O}(k)$ time. As $n_u \leq kt_u$, the time spent for step 3 can be bounded by $\mathcal{O}(k(t_u + p_u) \log n)$.

To update the permutations on edges e of P_u and T_u in step 4, the algorithm can traverse P_u and T_u bottom up towards the root of T in order to first compute the composite permutations σ_e . Then it can traverse P_u and T_u top down from the root in order to compute σ'_e and τ_e given σ'_f of the parent edge f of e , as τ_e is uniquely defined by σ'_f in all four cases (a) to (d). Thus this takes $\mathcal{O}(k(t_u + p_u))$ time, which concludes the proof. \blacktriangleleft

To obtain the amortized runtime we give a proof using the potential function method [13].

Proof of Theorem 22. As for Lemma 24, let t_u be the number of nodes in T_u and p_u be the number of nodes of P_u . Given a potential function Φ , the amortized runtime is given by the sum of the actual runtime per update, which is $\mathcal{O}(k(t_u + p_u) \log n)$ by Lemma 24, and Δ_Φ , which is the difference between the potential after and before adding an interval I_{new} to T .

To define the potential, let $h = \mathcal{O}(\log n)$ be the maximum height of the α -balanced tree T , and for any node u let $m_u = \max\{n_u^-, n_u^+\}$, $s_u = |\mathcal{S}_u|$, and $a_u = \sum_{w \in V(P_u)} s_w$ be the number of intervals stored in nodes of P_u . Then define

$$\begin{aligned} \Gamma(u) &= \max \left\{ \frac{m_u - n_u/2}{\alpha - 1/2}, 0 \right\}, & \beta &= 4k^2h + 2k, \\ \Lambda(u) &= 2ks_u \cdot (kp_u - a_u), & \Phi(u) &= \beta \cdot \Gamma(u) + \Lambda(u). \end{aligned}$$

Note that each node of P_u stores at most k intervals so that $a_u \leq kp_u$ and thus $\Lambda(u) \geq 0$. Hence $\Phi(u) \geq 0$ and we can define a potential function $\Phi = C \log n \cdot \sum_{u \in V(T)} \Phi(u)$, where C is the constant hidden in the \mathcal{O} -notation of the actual runtime according to Lemma 24. Note that the change Δ_Φ is only influenced by the addition of the new interval I_{new} into node v , and the rebuilding of a subtree T_w in step 1 of the algorithm. That is, none of the steps 2 to 4 change any of the terms of Φ .

To bound the amortized runtime, we distinguish the cases when some subtree T_w is rebalanced and when not. For the former case, let us begin by determining Δ_Γ , i.e., the change in $\sum_{u \in V(T)} \Gamma(u)$ during an update. After I_{new} is inserted into v we have $m_w > \lceil \alpha n_w \rceil$ at the node w before T_w is being rebuilt in step 1. This means that before inserting I_{new} we had $m_w \geq \lceil \alpha n_w \rceil \geq \alpha n_w$, and thus $\Gamma(w) \geq n_w$. After rebuilding T_w it is perfectly balanced and we have $m_x \leq n_x/2$ for every node x of T_w , so that now $\Gamma(x) = 0$. In particular,

during the update, $\Gamma(w)$ decreased from at least n_w to 0. Note that compared to before the update, T_w may contain a different set of nodes after it is rebuilt. Nevertheless, the sum $\sum_{x \in V(T_w)} \Gamma(x)$ will decrease during the update, as afterwards $\Gamma(x) = 0$ for every node x of T_w . In the remaining tree T the value $\Gamma(u)$ can only increase by at most 1 for nodes u along the path P_w . Hence we get that $\Delta_\Gamma \leq p_w - n_w$.

We now determine Δ_Λ , i.e., the change in $\sum_{u \in V(T)} \Lambda(u)$ during an update, when a tree T_w is rebuilt in step 1. Note that $\Lambda(u)$ does not change for any node u of T that is not contained in T_w . As observed above, compared to before, T_w may contain a different set of nodes after it is rebuilt. However, the set of intervals \mathcal{I}_w stored in T_w remains the same. We therefore consider the contribution of each interval in \mathcal{S}_x towards $\Lambda(x)$ for any node x of T_w , before and after the update. Let us define $\Lambda'(u) = 2k(kp_u - a_u)$ for every node u , so that the contribution of every interval $I \in \mathcal{S}_u$ to $\Lambda(u)$ is $\Lambda'(u)$. For any node x of T_w , by definition of a_x and p_x we obtain

$$\Lambda'(x) = \sum_{y \in V(P_x)} 2k(k - s_y) = \Lambda'(w') + \sum_{y \in V(Q_x)} 2k(k - s_y),$$

where $Q_x \subseteq P_w$ is the path from x to w and w' is the parent of w (which exists since $w \neq r$). We may bound $\Lambda'(x)$ from below by $\Lambda'(w')$, and from above by $\Lambda'(w') + 2k^2p_x$. As $\Lambda'(w')$ is unchanged during the update, the contribution of each interval $I \in \mathcal{I}_w$ different from I_{new} changes by at most $2k^2p_x$, where x is the node of T_w storing I after the update. As I_{new} was not present in T_w before, its contribution adds $\Lambda'(w') + 2k^2p_x$ for the node x storing I_{new} after T_w is rebuilt. We may bound p_x by the height h of T after the update for any node x , and $\Lambda'(w')$ is at most $2k^2h$. Thus we get $\Delta_\Lambda \leq n_w \cdot 2k^2h + \Lambda'(w') \leq 2k^2h(n_w + 1)$, where n_w also counts I_{new} in \mathcal{I}_w .

Since $\beta = 4k^2h + 2k$ and $n_w \geq 1$, as a consequence of the above we obtain

$$\begin{aligned} \Delta_\Phi &= C(\beta\Delta_\Gamma + \Delta_\Lambda) \log n \leq C(\beta(p_w - n_w) + 2k^2h(n_w + 1)) \log n \\ &\leq C(\beta p_w - (4k^2h + 2k)n_w + 4k^2hn_w) \log n \leq C(\beta p_w - 2kn_w) \log n. \end{aligned}$$

We have that $t_w \leq 2n_w$, since we maintain the invariant that for every node u except the root of T , if $\mathcal{S}_u = \emptyset$ then u is a leaf of the complete binary tree T . Hence the actual runtime according to Lemma 24 can be upper bounded by $Ck(2n_w + p_w) \log n$, which means that the amortized runtime is $C(\beta + k)p_w \log n = \mathcal{O}(k^2 \log^3 n)$ in case a subtree T_w is rebalanced in step 1, since $\beta = \mathcal{O}(k^2 \log n)$ and $p_w \leq h = \mathcal{O}(\log n)$.

We now turn to the case when no subtree is rebalanced in step 1 and the only change of Φ is due to I_{new} being added to a node v of T . Note that $\Gamma(u)$ only changes along the nodes u of path P_v , where m_u may increase by 1. Thus $\Delta_\Gamma \leq \frac{p_v}{\alpha-1/2}$. To bound Δ_Λ we consider two cases: either v was an existing internal node of T , or v was a leaf and is then converted into an internal node. In the first case, a_u of every node u of T_v increases by 1 due to the new interval I_{new} stored in the ancestor v of u , and so $\Lambda(u)$ decreases by $2ks_u$. At the same time, $\Lambda(u)$ is unchanged for any node u not in T_v , and we get $\Delta_\Lambda \leq -2kn_v$. Hence in this case $\Delta_\Phi \leq C(\frac{\beta p_v}{\alpha-1/2} - 2kn_v) \log n$. As we have seen the actual runtime can be upper bounded by $Ck(2n_v + p_v) \log n$, and thus the amortized runtime becomes $Cp_v(\frac{\beta}{\alpha-1/2} + k) \log n = \mathcal{O}(k^2 \log^3 n)$.

Finally, if I_{new} is added to a leaf v of T , then v is converted into an internal node by adding two leaves to v . For any leaf x , $\Lambda(x) = 0$ as $s_x = 0$, and thus these new nodes do not contribute to Δ_Λ . However v was formerly a leaf and now contains I_{new} , so that its contribution to Δ_Λ is $2k(kp_v - a_v) \leq 2k^2p_v$. Hence we get $\Delta_\Phi \leq C(\frac{\beta p_v}{\alpha-1/2} + 2k^2p_v) \log n = \mathcal{O}(k^2 \log^3 n)$. The subtree T_v only stores I_{new} so that $n_v = 1$ and the actual runtime is $Ck(2n_v + p_v) \log n = \mathcal{O}(k \log^2 n)$. Thus in this case we also obtain an amortized runtime of $\mathcal{O}(k^2 \log^3 n)$, which concludes the proof. \blacktriangleleft

Optimal Randomized Group Testing Algorithm to Determine the Number of Defectives

Nader H. Bshouty

Department of Computer Science,
Technion, Haifa, Israel

Raghd Boulos

The Arab Orthodox College, Haifa, Israel

Jalal Nada

The Arab Orthodox College, Haifa, Israel

Yara Zaknoon

The Arab Orthodox College, Haifa, Israel

Catherine A. Haddad-Zaknoon

Department of Computer Science,
Technion, Haifa, Israel

Foad Moalem

The Arab Orthodox College, Haifa, Israel

Elias Noufi

The Arab Orthodox College, Haifa, Israel

Abstract

We study the problem of determining the exact number of defective items in an adaptive group testing by using a minimum number of tests. We improve the existing algorithm and prove a lower bound that shows that the number of tests in our algorithm is optimal up to small additive terms.

2012 ACM Subject Classification Mathematics of computing; Mathematics of computing → Discrete mathematics; Mathematics of computing → Probabilistic algorithms; Theory of computation → Probabilistic computation

Keywords and phrases Group Testing, Randomized Algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.18

1 Introduction

Let X be a set of *items* that contains some *defective items* $I \subseteq X$. In group testing, we *test* a subset $Q \subseteq X$ of items. An answer to the test is 1 if Q contains at least one defective item, i.e., $Q \cap I \neq \emptyset$, and 0 otherwise. Group testing was initially introduced as a potential approach to the economical mass blood testing, [15]. However, it has been proven to be applicable in a variety of problems, including DNA library screening, [26], quality control in product testing, [30], searching files in storage systems, [22], sequential screening of experimental variables, [24], efficient contention resolution algorithms for multiple-access communication, [22, 34], data compression, [20], and computation in the data stream model, [12]. See a brief history and other applications in [11, 16, 17, 21, 25, 26] and references therein.

Estimating or determining exactly the number of defective items is an important problem in biological and medical applications [4, 31]. For example it is used to estimate the proportion of organisms capable of transmitting the aster-yellows virus in a natural population of leafhoppers [32], estimating the infection rate of the yellow-fever virus in a mosquito population [33] and estimating the prevalence of a rare disease using grouped samples to preserve individual anonymity [23].

In *adaptive algorithms*, the tests can depend on the answers of the previous ones. In *non-adaptive algorithms*, they are independent of the previous ones and; therefore, all tests can be done in one parallel step.

In this paper, we study the problem of determining exactly the number of defective items with adaptive group testing algorithms. We first give an algorithm that improves the number of tests in the best-known algorithm by a factor of 4. Improving constant factors in Group testing algorithms is one of the utmost important challenges in group testing since, in many



© Nader H. Bshouty, Catherine A. Haddad-Zaknoon, Raghd Boulos, Foad Moalem, Jalal Nada, Elias Noufi, and Yara Zaknoon;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 18; pp. 18:1–18:12



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

applications, tests are incredibly costly and time-consuming [3, 6, 9, 17, 18, 27]. Moreover, we give a lower bound that shows that our algorithm is optimal up to a small additive term. To the best of our knowledge, this is the first non-trivial lower bound for this problem.

1.1 Previous and New Results

Let X be a set of n items with d defective items I . All the algorithms in this paper are *adaptive*. That is, the tests can depend on the answers to the previous ones. All the non-adaptive algorithms for determining exactly the number of defective items must ask at least $\Omega((d^2/\log d) \log n)$ queries and $\Omega(\log n/\log \log n)$ for estimating their number, [1, 13, 14]. In [2], Bshouty et al. show that any deterministic or Las Vegas adaptive algorithm must ask at least $\Omega(d \log(n/d))$ queries. Since the query complexity depends on the number of items n , which, for most applications, is extremely large, non-adaptive algorithms and Las Vegas (and deterministic) algorithms are not desirable for solving this problem.

In [5], Cheng gave a randomized Monte Carlo adaptive algorithm that for any constant c , asks $4dc \log d$ queries¹ and, with probability at least $1 - \delta = 1 - 1/d^{c-1}$, determines exactly the number of defective items. His algorithm, with the technique used in this paper², gives a randomized Monte Carlo algorithm that asks $4d \log(d/\delta)$ queries with success probability at least $1 - \delta$ for any δ .

In this paper, we first give lower bounds for the number of queries. The first lower bound is $d \log(1/d\delta)$ for any n , d and $\delta > 1/(2(n - d + 1))$. See Theorem 4 in Section 3. This shows that Cheng's algorithm is almost optimal (up to a multiplicative factor of 4 and an additive term of $4d \log d$). For $\delta < 1/(2(n - d + 1))$, we give the tight bound $d \log(n/d)$, which, in particular, is the number of tests required for any deterministic algorithm. This bound matches the tight bound for *finding* all the defective items. We also give better lower bound of $d \log(1/\delta)$ for any large enough³ n . See Theorem 5 in Section 3.

Moreover, we give a new randomized Monte Carlo algorithm that asks $d \log(d/\delta)$ queries. See Theorem 7 in Section 4. Our algorithm improves Cheng algorithm by a multiplicative factor of 4 and is optimal up to an additive term of $d \log d$. Notice that, for $\delta = 1/d^{\omega(1)}$ (especially when δ depends on n), our algorithm is optimal up to a small additive term.

Estimating the number of defective items is studied in [2, 10, 13, 14, 19, 28]. The problem is to find an integer D such that $d \leq D \leq (1 + \epsilon)d$. In [2], Bshouty et al. modified Falhatgar et al. algorithm, [19], and gave a randomized algorithm that makes expected number of $(1 - \delta) \log \log d + O((1/\epsilon^2) \log(1/\delta))$ tests. They also prove the lower bound $(1 - \delta) \log \log d + \Omega((1/\epsilon) \log(1/\delta))$.

2 Definitions and Preliminary Results

In this section we give some notations, definitions, the type of algorithms that are used in the literature and some preliminary results.

¹ All the log in this paper are \log_2 and all the complexities in this introduction are multiplied by $1 + o(1)$ where the $o(1)$ is with respect to d .

² First estimate d using the algorithm in this paper. Then determine c to get success δ and run his algorithm

³ $n \geq d^{\omega(1)}$ where $\omega(1)$ is with respect to d for example $\log^* d$

2.1 Notations and Definitions

Let $X = [n] := \{1, 2, 3, \dots, n\}$ be a set of *items* with some *defective* items $I \subseteq [n]$. In group testing, we *query* a subset $Q \subseteq X$ of items and the answer to the query is $Q(I) := 1$ if Q contains at least one defective item, i.e., $Q \cap I \neq \emptyset$, and $Q(I) := 0$, otherwise.

Let $I \subseteq [n]$ be the set of defective items. Let \mathcal{O}_I be an *oracle* that for a query $Q \subseteq [n]$ returns $Q(I)$. Let A be an algorithm that has access to an oracle \mathcal{O}_I . The output of the algorithm with an access to an oracle \mathcal{O}_I is denoted by $A(\mathcal{O}_I)$. When the algorithm is randomized, then we add the random seed r , and then the output of the algorithm is a random variable $A(\mathcal{O}_I, r)$ in $[n]$. When I is known from the context, we just write $A(r)$. Let A be a randomized algorithm and let r_0 be a fixed seed. Then $A(r_0)$ is a deterministic algorithm that is equivalent to the algorithm A with the fixed seed r_0 . We denote by $\mathcal{Q}(A, \mathcal{O}_I)$ (or $\mathcal{Q}(A, \mathcal{O}_I, r)$) the set of queries that A asks with oracle \mathcal{O}_I (and a seed r). We say that the algorithm determines $|I| = d$ exactly if $A(\mathcal{O}_I, r) = |I|$.

2.2 Types of Algorithms

In this paper we consider four types of algorithms whose running time is polynomial in n .

1. The *deterministic* algorithm A with an oracle \mathcal{O}_I , $I \subseteq X$. The query complexity of a deterministic algorithm A is the *worst case complexity*, i.e., $\max_{|I|=d} |\mathcal{Q}(A, \mathcal{O}_I)|$.
2. The *randomized Las Vegas algorithm*. We say that a randomized algorithm $A(r)$ is a *randomized Las Vegas algorithm that has expected query complexity $g(d)$* if for any $I \subseteq X$, $A(r)$ with an oracle \mathcal{O}_I asks $g(|I|)$ expected number of queries and with probability 1 outputs $|I|$.
3. The *randomized Monte Carlo algorithm*. We say that a randomized algorithm $A(r)$ is a *randomized Monte Carlo algorithm that has query complexity $g(d, \delta)$* if for any $I \subseteq X$, A with an oracle \mathcal{O}_I asks at most $g(|I|, \delta)$ queries and with probability at least $1 - \delta$ outputs $|I|$.
4. The *randomized Monte Carlo algorithm with average case complexity*. We say that a randomized algorithm $A(r)$ is *Monte Carlo algorithm with average case complexity that has expected query complexity $g(d, \delta)$* if for any $I \subseteq X$, A asks $g(|I|, \delta)$ expected number of queries and with probability at least $1 - \delta$ outputs $|I|$.

2.3 Preliminary Results

We now prove a few results that will be used throughout the paper.

Let $s \in \cup_{i=0}^{\infty} \{0, 1\}^i$ be a *string* over $\{0, 1\}$ (including the empty string $\lambda \in \{0, 1\}^0$). We denote by $|s|$ the *length* of s , i.e., the integer m such that $s \in \{0, 1\}^m$. Let $s_1, s_2 \in \cup_{n=0}^{\infty} \{0, 1\}^n$ be two strings over $\{0, 1\}$ of length m_1 and m_2 , respectively. We say that s_1 is a *prefix* of s_2 if $m_1 \leq m_2$ and $s_{1,i} = s_{2,i}$ for all $i = 1, \dots, m_1$. We denote by $s_1 \cdot s_2$ the *concatenation* of the two strings.

The following lemma is proved in [1].

► **Lemma 1.** *Let $S = \{s_1, \dots, s_N\}$ be a set of N distinct strings such that no string is a prefix of another. Then*

$$\max_{s \in S} |s| \geq E(S) := \mathbf{E}_{s \in S}[|s|] \geq \log N.$$

► **Lemma 2.** *Let A be a deterministic adaptive algorithm that asks queries. If $A(\mathcal{O}_I) \neq A(\mathcal{O}_J)$ then there is $Q_0 \in \mathcal{Q}(A, \mathcal{O}_I) \cap \mathcal{Q}(A, \mathcal{O}_J)$ such that $Q_0(I) \neq Q_0(J)$.*

In particular, if, in addition, $I \subseteq J$ then $Q_0(I) = 0$ and $Q_0(J) = 1$.

18:4 Algorithm to Determine the Number of Defectives

Proof. Since algorithm A is deterministic, in the execution of A with \mathcal{O}_I and \mathcal{O}_J , A asks the same queries as long as it gets the same answers to the queries. Since $A(\mathcal{O}_I) \neq A(\mathcal{O}_J)$, there must be a query Q_0 that is asked to both \mathcal{O}_I and \mathcal{O}_J for which $Q_0(I) \neq Q_0(J)$. ◀

► **Lemma 3.** *Let A be a deterministic adaptive algorithm that asks queries. Let $C \subseteq 2^{[n]} = \{I \mid I \subseteq [n]\}$. If for every two distinct I_1 and I_2 in C there is a query $Q \in \mathcal{Q}(A, \mathcal{O}_{I_1})$ such that $Q(I_1) \neq Q(I_2)$ then*

$$\max_{I \in C} |\mathcal{Q}(A, \mathcal{O}_I)| \geq \mathbf{E}_{I \in C} |\mathcal{Q}(A, \mathcal{O}_I)| \geq \log |C|.$$

That is, the query complexity of A is at least $\log |C|$.

Proof. For $I \in C$ consider the sequence of the queries that A with the oracle \mathcal{O}_I asks and let $s(I) \in \cup_{n=0}^{\infty} \{0, 1\}^n$ be the sequence of answers. The query complexity and average-case complexity of A is $s(C) := \max_{I \in C} |s(I)|$ and $\bar{s}(C) := \mathbf{E}_{I \in C} |s(I)|$ where $|s(I)|$ is the length of $s(I)$. We show that for every two distinct I_1 and I_2 in C , $s(I_1) \neq s(I_2)$ and $s(I_1)$ is not a prefix of $s(I_2)$. This implies that $\{s(I) \mid I \in C\}$ contains $|C|$ distinct strings such that no string is a prefix of another. Then by Lemma 1, the result follows.

Consider two distinct sets $I_1, I_2 \subseteq [n]$. There is a query $Q_0 \in \mathcal{Q}(A, \mathcal{O}_{I_1})$ such that $Q_0(I_1) \neq Q_0(I_2)$. Consider the execution of algorithm A with both \mathcal{O}_{I_1} and \mathcal{O}_{I_2} , respectively. Since A is deterministic, as long as the answers of the queries are the same both (A with \mathcal{O}_{I_1} and A with \mathcal{O}_{I_2}) continue to ask the same query. Then, either we get to the query Q_0 in both execution and then $Q_0(I_1) \neq Q_0(I_2)$, or we reach some other query Q' , that is asked before Q_0 , satisfies $Q'(I_1) \neq Q'(I_2)$. In both cases, $s(I_1) \neq s(I_2)$ and $s(I_1)$ is not a prefix of $s(I_2)$. ◀

3 Lower Bounds

In this section, we prove some lower bounds for the number of queries that are needed to determine exactly the number of defective items with a Monte Carlo algorithm.

► **Theorem 4.** *Let $\delta \geq 1/(2(n - d + 1))$. Let A be a randomized Monte Carlo adaptive algorithm that for any set of defective items I of size $|I| \in \{d, d + 1\}$, with probability at least $1 - \delta$, determines exactly the number of defective items $|I|$. Algorithm A must ask at least*

$$d \log \frac{1}{2d\delta} - 1$$

queries.

In particular, when $\delta \leq 1/(2(n - d + 1))$ then A must ask at least $d \log(n/d) - O(d)$ queries which is the query complexity (up to additive term $O(d)$) of finding the defective items (and therefore, in particular, finding $|I|$) with $\delta = 0$ error.

Proof. Let $A(\mathcal{O}_I, r)$ be a randomized Monte Carlo algorithm that for $|I| \in \{d, d + 1\}$, determines $|I|$ with probability at least $1 - \delta$ where r is the random seed of the algorithm. Let $X(I, r)$ be a random variable that is equal to 1 if $A(\mathcal{O}_I, r) \neq |I|$ and 0 otherwise. Then, for any $I \subseteq [n]$, $\mathbf{E}_r[X(I, r)] \leq \delta$. Let $m = \lfloor 1/(2\delta) \rfloor + d - 1 \leq n$. Consider any $J \subseteq [m]$, $|J| = d$. For any such J , let

$$Y_J(r) = X(J, r) + \sum_{i \in [m] \setminus J} X(J \cup \{i\}, r).$$

Then, for every $J \subseteq [m]$ of size d , $\mathbf{E}_r [Y_J(r)] \leq (m - d + 1)\delta \leq \frac{1}{2}$. Therefore, for a random uniform $J \subseteq [m]$ of size d , we have $\mathbf{E}_r[\mathbf{E}_J[Y_J(r)]] = \mathbf{E}_J[\mathbf{E}_r[Y_J(r)]] \leq 1/2$. Thus, there is r_0 such that for at least half of the sets $J \subseteq [m]$, of size d , $Y_J(r_0) = 0$. Let C be the set of all $J \subseteq [m]$, of size d , such that $Y_J(r_0) = 0$. Then

$$|C| \geq \frac{1}{2} \binom{m}{d} = \frac{1}{2} \binom{\lfloor 1/(2\delta) \rfloor + d - 1}{d}.$$

Consider the deterministic algorithm $A(r_0)$. We now claim that for every two distinct $J_1, J_2 \in C$, there is a query $Q_0 \in \mathcal{Q}(A(r_0), \mathcal{O}_{J_1})$ such that $Q_0(J_1) \neq Q_0(J_2)$. If this is true then, by Lemma 3, the query complexity of $A(r_0)$ is at least

$$\log |C| \geq \log \frac{1}{2} \binom{\lfloor 1/(2\delta) \rfloor + d - 1}{d} \geq d \log \frac{1}{2d\delta} - 1.$$

Proof. We now prove the claim. Consider two distinct $J_1, J_2 \in C$. There is w.l.o.g $j \in J_2 \setminus J_1$. Since $Y_{J_1}(r_0) = 0$, we have $X(J_1, r_0) = 0$ and $X(J_1 \cup \{j\}, r_0) = 0$ and, therefore, $A(\mathcal{O}_{J_1}, r_0) = d$ and $A(\mathcal{O}_{J_1 \cup \{j\}}, r_0) = d + 1$. Thus, by Lemma 2, there is a query $Q_0 \in \mathcal{Q}(A(r_0), \mathcal{O}_{J_1}) \cap \mathcal{Q}(A(r_0), \mathcal{O}_{J_1 \cup \{j\}})$ for which $Q_0(J_1) = 0$ and $Q_0(J_1 \cup \{j\}) = 1$. Therefore, $Q_0(\{j\}) = 1$ and then $Q_0(J_1) = 0$ and $Q_0(J_2) = 1$. \triangleleft

In the Appendix, we prove this lower bound for any randomized Monte Carlo algorithm with average-case complexity.

For large enough⁴ n , $n = d^{\omega(1)}$, the following result gives a better lower bound.

► **Theorem 5.** *Any randomized Monte Carlo adaptive algorithm that with probability at least $1 - \delta$ determines the number of defectives must ask at least*

$$\left(1 - \frac{\log d + \log(1/\delta) + 1}{\log n + \log(1/\delta)}\right) d \log \frac{1}{2\delta}$$

queries.

In particular, when $n = d^{\omega(1)}$ then the number of queries is at least

$$(1 - o(1)) d \log \frac{1}{2\delta}.$$

Proof. Let $A(r)$ be a randomized Monte Carlo algorithm that determines the number of defective items with probability at least $1 - \delta$ where r is the random seed of the algorithm. Let $X'(I, r)$ be a random variable that is equal to 1 if $A(\mathcal{O}_I, r) \neq |I|$ and 0 otherwise. Then, for every I , $\mathbf{E}_r[X'(I, r)] \leq \delta$. For every set I and $i \in [n] \setminus I$, let $X(I, i, r) = X'(I, r) + X'(I \cup \{i\}, r)$. Then, for every $I \subseteq [n]$ and $i \in [n] \setminus I$, $\mathbf{E}_r[X(I, i, r)] \leq 2\delta$. For I of size d chosen uniformly at random and $i \in [n] \setminus I$ chosen uniformly at random, we have $\mathbf{E}_r \mathbf{E}_I \mathbf{E}_i[X(I, i, r)] = \mathbf{E}_I \mathbf{E}_i \mathbf{E}_r[X(I, i, r)] \leq 2\delta$. Therefore, there exists a seed r_0 such that $\mathbf{E}_I \mathbf{E}_i[X(I, i, r_0)] \leq 2\delta$. We now choose q permutations $\phi_1, \dots, \phi_q : [n] \rightarrow [n]$ uniformly and independently at random where

$$q = \left\lceil \frac{1 + \log n}{\log \frac{1}{2\delta}} \right\rceil.$$

⁴ The $\omega(1)$ is with respect to d . For example, $n > d^{\log^* d}$.

18:6 Algorithm to Determine the Number of Defectives

Then, for any I and $i \in [n] \setminus I$, $\phi_1(I), \dots, \phi_q(I)$ are uniform and independent random sets of size d and $\phi_1(i), \dots, \phi_q(i)$ are uniform and independent random integers where $\phi_j(i) \notin \phi_j(I)$ for all $j \in [q]$. Hence,

$$\mathbf{E}_{\{\phi_j\}_j} \left[\prod_{j=1}^q X(\phi_j(I), \phi_j(i), r_0) \right] = \prod_{j=1}^q \mathbf{E}_{\phi_j} [X(\phi_j(I), \phi_j(i), r_0)] \leq (2\delta)^q$$

and,

$$\begin{aligned} \mathbf{E}_{\{\phi_j\}_j} \mathbf{E}_I \mathbf{E}_i \left[\prod_{j=1}^q X(\phi_j(I), \phi_j(i), r_0) \right] &= \mathbf{E}_I \mathbf{E}_i \mathbf{E}_{\{\phi_j\}_j} \left[\prod_{j=1}^q X(\phi_j(I), \phi_j(i), r_0) \right] \\ &\leq (2\delta)^q. \end{aligned}$$

Therefore,

$$\mathbf{E}_{\{\phi_j\}_j} \mathbf{E}_I \left[\sum_{i \in [n] \setminus I} \prod_{j=1}^q X(\phi_j(I), \phi_j(i), r_0) \right] \leq (n-d)(2\delta)^q < \frac{1}{2}.$$

Thus, there are permutations $\{\phi'_j\}_{j \in [q]}$ such that

$$\mathbf{E}_I \left[\sum_{i \in [n] \setminus I} \prod_{j=1}^q X(\phi'_j(I), \phi'_j(i), r_0) \right] < \frac{1}{2}.$$

Since X takes values in $\{0, 1\}$, this implies that for at least half of the sets $I \subseteq [n]$, of size d , and all $i \in [n] \setminus I$, there exists $j \in [q]$ such that $X(\phi'_j(I), \phi'_j(i), r_0) = 0$. Let C be the class of such sets I for which the later statement is true. Then,

$$|C| \geq \frac{1}{2} \binom{n}{d}$$

and

$$(\forall I \in C)(\forall i \in [n] \setminus I)(\exists j \in [q]) X(\phi'_j(I), \phi'_j(i), r_0) = 0. \quad (1)$$

Consider the following deterministic algorithm A^* :

Algorithm A^*
 For $j = 1, \dots, q$
 Run $A(r_0)$ with oracle \mathcal{O}_I
 If $A(r_0)$ asks Q then ask the query $\phi_j'^{-1}(Q)$.

First notice that, if algorithm $A(r_0)$ has query complexity M , then A^* has query complexity at most qM .

Since $\phi_j'^{-1}(Q) \cap I = \emptyset$ if and only if $Q \cap \phi'_j(I) = \emptyset$, at iteration j , the algorithm $A(r_0)$ runs as if the defective items are $\phi'_j(I)$. Therefore, at iteration j , the queries that are asked by $A(r_0)$ are $\mathcal{Q}(A(r_0), \mathcal{O}_{\phi'_j(I)})$ and the queries that are asked by A^* are $\phi_j'^{-1}(\mathcal{Q}(A(r_0), \mathcal{O}_{\phi'_j(I)}))$. Hence,

$$\mathcal{Q}(A^*, \mathcal{O}_I) = \bigcup_{j=1}^q \phi_j'^{-1}(\mathcal{Q}(A(r_0), \mathcal{O}_{\phi'_j(I)})). \quad (2)$$

We now show that,

▷ **Claim 6.** For every two distinct sets $I_1, I_2 \in C$ there is a query $Q' \in \mathcal{Q}(A^*, \mathcal{O}_{I_1})$ that gives different answers for I_1 and I_2 .

If this Claim is true then, by Lemma 3, the query complexity qM of A^* is at least $\log |C|$ and then, since $\binom{n}{d} \geq (n/d)^d$,

$$\begin{aligned} M &\geq \frac{\log |C|}{q} \geq \frac{\log \frac{1}{2} \binom{n}{d}}{\frac{1+\log n}{\log(1/(2\delta))} + 1} \\ &\geq \frac{\log n - \log d - 1}{\log n + \log \frac{1}{\delta}} \cdot d \log \frac{1}{2\delta} \\ &= \left(1 - \frac{\log d + \log(1/\delta) + 1}{\log n + \log(1/\delta)}\right) d \log \frac{1}{2\delta}. \end{aligned}$$

Proof. We now prove the claim. Let $I_1, I_2 \in C$ be two distinct sets of size d . Then there is $i_0 \in I_2 \setminus I_1$. By (1) there is $j_0 \in [q]$ such that for $\phi := \phi'_{j_0}$, $X(\phi(I_1), \phi(i_0), r_0) = 0$. Therefore $A(\mathcal{O}_{\phi(I_1)}, r_0) = |\phi(I_1)| = |I_1|$ and

$$A(\mathcal{O}_{\phi(I_1) \cup \{\phi(i_0)\}}, r_0) = |\phi(I_1) \cup \{\phi(i_0)\}| = |I_1| + 1.$$

Therefore, by Lemma 2, there exists a query $Q_0 \in \mathcal{Q}(A(r_0), \mathcal{O}_{\phi(I_1)})$ that satisfies $Q_0(\phi(I_1)) = 0$ and $Q_0(\phi(I_1) \cup \{\phi(i_0)\}) = 1$. That is, $Q_0 \cap \phi(I_1) = \emptyset$ and $Q_0 \cap (\phi(I_1) \cup \{\phi(i_0)\}) \neq \emptyset$. This implies that $\phi(i_0) \in Q_0$. Since $\phi(i_0) \in \phi(I_2)$, we get that $Q_0 \cap \phi(I_1) = \emptyset$ and $Q_0 \cap \phi(I_2) \neq \emptyset$. Thus, $\phi^{-1}(Q_0) \cap I_1 = \emptyset$ and $\phi^{-1}(Q_0) \cap I_2 \neq \emptyset$. That is, the query $Q' := \phi^{-1}(Q_0)$ satisfies

$$Q'(I_1) \neq Q'(I_2).$$

Since $Q_0 \in \mathcal{Q}(A(r_0), \mathcal{O}_{\phi(I_1)})$, by (2), we have (recall that $\phi := \phi'_{j_0}$)

$$Q' = \phi^{-1}(Q_0) \in \phi^{-1}(\mathcal{Q}(A(r_0), \mathcal{O}_{\phi(I_1)})) \subseteq \mathcal{Q}(A^*, \mathcal{O}_I)$$

and therefore, $Q' \in \mathcal{Q}(A^*, \mathcal{O}_I)$. This completes the proof of the claim. ◀

◀

4 Upper Bound

In this section we prove:

▶ **Theorem 7.** *There is a Monte Carlo adaptive algorithm that asks*

$$d \log \frac{d}{\delta} + O\left(d + \log d \log \frac{1}{\delta}\right) = (1 + o(1))d \log \frac{d}{\delta}$$

queries and, with probability at least $1 - \delta$, finds the number of defective items.

This improves the bound $4d \log(d/\delta)$ achieved in [5]. By Theorem 5, this bound is optimal up to the additive term $(1 + o(1))d \log d$.

We will use the following.

▶ **Lemma 8** ([7, 8, 29]). *There is a deterministic algorithm, **Find-Defectives**, that without knowing d , asks $d \log(n/d) + O(d)$ queries and finds the defective items.*

Our algorithm, at the first stage, calls the procedure **Estimate** that, with probability at least $1 - \delta/2$ finds an estimate D of the number defective items where $d \leq D \leq 8d$. This procedure makes $O(d + \log d \log(1/\delta))$ queries.

18:8 Algorithm to Determine the Number of Defectives

At the second stage, it uniformly at random partitions the set of items $[n]$ into $t = D^2/\delta$ disjoint sets B_1, B_2, \dots, B_t . We show that, with probability at least $1 - \delta/2$, each set contains at most one defective item. We call a set that contains a defective item a *defective set*. Therefore, with probability at least $1 - \delta/2$, the number of defective items is equal to the number of defective sets. Then, we treat each set B_i as one item i and call the algorithm **Find-Defectives** in Lemma 8 on t items to find the defective sets. To do that, each test $Q \subseteq [t]$ in **Find-Defectives** is simulated by the query $S(Q) := \cup_{i \in Q} B_i$ in our algorithm. Obviously, the set Q contains an index of a defective set if and only if $S(Q)$ contains a defective item. Therefore, the algorithm will return the number of defective sets which, with probability at least $1 - \delta/2$, is equal to the number of defective items. By Lemma 8, the number of queries asked in the second stage is

$$d \log \frac{t}{d} + O(d) = d \log \frac{D^2/\delta}{d} + O(d) = d \log \frac{d}{\delta} + O(d).$$

We now prove:

► **Lemma 9.** *There is a Monte Carlo adaptive algorithm **Estimate** that asks*

$$O\left(d + \log d \log \frac{1}{\delta}\right)$$

queries and returns an integer D that, with probability at least $1 - \delta$, $D \geq d$ and, with probability 1, $D \leq 8d$.

Proof. The algorithm is in Figure 1. For each $k = 2^i$, the algorithm does the following $t = \lceil 2 \log(1/\delta)/k \rceil$ times: uniformly at random divides the items into k mutually disjoint sets and then tests each set and counts (in the variable “count”) the number of sets that contain at least one defective item. First, notice that, for $k \leq d$,

$$\Pr[\text{count} < k/4] \leq \binom{k}{k/4} \left(\frac{1}{4}\right)^d \leq 2^k \left(\frac{1}{4}\right)^d \leq 2^{-d}.$$

The probability that the algorithm outputs $k < d$ is the probability that the event “count $< k/4$ ” happens $t = \lceil 2 \log(1/\delta)/k \rceil$ times for some $k = 2^i < d$. This is at most

$$\sum_{i=1}^{\lfloor \log d \rfloor} (2^{-d})^{\lceil 2 \log(1/\delta)/2^i \rceil} \leq \sum_{i=1}^{\lfloor \log d \rfloor} \delta^{2d/2^i} \leq \delta.$$

Therefore, with probability at least $1 - \delta$, the output is greater or equal to d . Since “count” is always less than or equal to the number of defective items d , when $4d < k \leq 8d$, we have $\text{count} \leq d < k/4$ and the algorithm halts. Therefore, the output cannot be greater than $8d$.

The number of queries that the algorithm asks is at most

$$\sum_{i=1}^{\lceil \log 8d \rceil} 2^i \left\lceil \frac{2 \log(1/\delta)}{2^i} \right\rceil \leq \sum_{i=1}^{\lceil \log 8d \rceil} 2 \log(1/\delta) + 2^i = O\left(d + \log d \log \frac{1}{\delta}\right). \quad \blacktriangleleft$$

We now prove Theorem 7.

Proof. The algorithm is in Figure 2. First, we run the algorithm in Figure 1 that estimates d and returns D such that, with probability at least $1 - \delta/2$, $d \leq D \leq 8d$. Then, the algorithm chooses a function $f : [n] \rightarrow [N]$ uniformly at random where $N = \lceil D^2/\delta \rceil$. This is equivalent

Estimate(n, δ)

- 1) For $k = 2^i, i = 1, 2, 3, \dots$ do
- 2) For $m = 1$ to $t := \lceil \frac{2 \log(1/\delta)}{k} \rceil$
- 3) Choose a function $f_m : [n] \rightarrow [k]$ uniformly at random
- 4) count $\leftarrow 0$
- 5) For $j = 1$ to k
- 6) ask the query $Q := f_m^{-1}(j)$
- 7) If the answer is 1 then count \leftarrow count+1
- 8) EndFor
- 9) If (count $\geq k/4$) Break (leave the For loop)
- 10) EndFor
- 11) If (count $\leq k/4$) Output(k) and halt
- 12) EndFor

■ **Figure 1** An algorithm that estimates d .

to uniformly at random divide the items into N mutually disjoint sets $Q_i, i = 1, \dots, N$. The probability that some Q_i contains two defective items is

$$\begin{aligned} \Pr[(\exists i) Q_i \text{ contains two defective items}] &= 1 - \prod_{i=1}^{d-1} \left(1 - \frac{i}{N}\right) \\ &\leq \sum_{i=1}^{d-1} \frac{i}{N} \leq \frac{d^2}{2N} \leq \frac{\delta}{2}. \end{aligned}$$

Then, the algorithm runs **Find-Defectives** in Lemma 8 on the N disjoint sets Q_1, \dots, Q_N to find the number of sets that contain a defective item. This number is, with probability at least $1 - \delta/2$, equal to the number of defective items. Therefore, with probability at least $1 - \delta$, **Find-Defectives** finds d . This completes the proof of correctness.

The query complexity is the query complexity of **Estimate**($n, \delta/2$) and **Find-Defectives** with N items. This, by Lemma 8 and 9, is equal to

$$d \log \frac{N}{d} + O(d) + O\left(d + \log d \log \frac{1}{\delta}\right) = d \log \frac{d}{\delta} + O\left(d + \log d \log \frac{1}{\delta}\right). \quad \blacktriangleleft$$

Find-d(n, δ)

- 1) $D \leftarrow$ **Estimate**($n, \delta/2$).
- 3) $N \leftarrow \lceil D^2/\delta \rceil$.
- 2) Choose a function $f : [n] \rightarrow [N]$ uniformly at random.
- 3) Define $Q_i = f^{-1}(i)$ for $i = 1, 2, \dots, N$.
- 4) Run **Find-Defectives** with N items.
- 5) and for each query Q ask $\cup_{j \in Q} Q_j$.
- 6) Let Δ be the output of **Find-Defectives**.
- 7) Output Δ .

■ **Figure 2** An algorithm that finds d .

References

- 1 Nader H. Bshouty. Lower bound for non-adaptive estimate the number of defective items. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:53, 2018. URL: <https://eccc.weizmann.ac.il/report/2018/053>.
- 2 Nader H. Bshouty, Vivian E. Bshouty-Hurani, George Haddad, Thomas Hashem, Fadi Khoury, and Omar Sharafy. Adaptive group testing algorithms to estimate the number of defectives. In *Algorithmic Learning Theory, ALT 2018, 7-9 April 2018, Lanzarote, Canary Islands, Spain*, pages 93–110, 2018. URL: <http://proceedings.mlr.press/v83/bshouty18a.html>.
- 3 Nader H. Bshouty, Nuha Diab, Shada R. Kawar, and Robert J. Shahla. Non-adaptive randomized algorithm for group testing. In *International Conference on Algorithmic Learning Theory, ALT 2017, 15-17 October 2017, Kyoto University, Kyoto, Japan*, pages 109–128, 2017.
- 4 Chao L. Chen and William H. Swallow. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics.*, 46(4):1035–1046, 1990.
- 5 Yongxi Cheng. An efficient randomized group testing procedure to determine the number of defectives. *Oper. Res. Lett.*, 39(5):352–354, 2011. doi:10.1016/j.orl.2011.07.001.
- 6 Yongxi Cheng and Ding-Zhu Du. New constructions of one- and two-stage pooling designs. *Journal Of Computational Biology*, 2008.
- 7 Yongxi Cheng, Ding-Zhu Du, and Yinfeng Xu. A zig-zag approach for competitive group testing. *INFORMS Journal on Computing*, 26(4):677–689, 2014. doi:10.1287/ijoc.2014.0591.
- 8 Yongxi Cheng, Ding-Zhu Du, and Feifeng Zheng. A new strongly competitive group testing algorithm with small sequentiality. *Annals OR*, 229(1):265–286, 2015. doi:10.1007/s10479-014-1766-4.
- 9 Yongxi Cheng, Dingzhu Du, and Guohui Lin. On the upper bounds of the minimum number of rows of disjunct matrices. *Optimization Letters*, 3:297–302, 2009.
- 10 Yongxi Cheng and Yinfeng Xu. An efficient FPRAS type group testing procedure to approximate the number of defectives. *J. Comb. Optim.*, 27(2):302–314, 2014. doi:10.1007/s10878-012-9516-5.
- 11 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013. doi:10.1007/978-3-642-17327-1.
- 12 Graham Cormode and S. Muthukrishnan. What’s hot and what’s not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005. doi:10.1145/1061318.1061325.
- 13 Peter Damaschke and Azam Sheikh Muhammad. Bounds for nonadaptive group tests to estimate the amount of defectives. In *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Kailua-Kona, HI, USA, December 18-20, 2010, Proceedings, Part II*, pages 117–130, 2010. doi:10.1007/978-3-642-17461-2_10.
- 14 Peter Damaschke and Azam Sheikh Muhammad. Competitive group testing and learning hidden vertex covers with minimum adaptivity. *Discrete Math., Alg. and Appl.*, 2(3):291–312, 2010. doi:10.1142/S179383091000067X.
- 15 R. Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, pages 436–440, 1943.
- 16 D. Du and F. K Hwang. Combinatorial group testing and its applications. *World Scientific Publishing Company.*, 2000.
- 17 D. Du and F. K Hwang. Pooling design and nonadaptive group testing: important tools for dna sequencing. *World Scientific Publishing Company.*, 2006.
- 18 Ding-Zhu Du, Frank K. Hwang, Weili Wu, and Taieb Znati. New construction for transversal design. *Journal of computational biology : a journal of computational molecular cell biology*, 13:990–5, June 2006. doi:10.1089/cmb.2006.13.990.
- 19 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Estimating the number of defectives with group testing. In

- IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 1376–1380, 2016. doi:10.1109/ISIT.2016.7541524.
- 20 Edwin S. Hong and Richard E. Ladner. Group testing for image compression. *IEEE Trans. Image Processing*, 11(8):901–911, 2002. doi:10.1109/TIP.2002.801124.
 - 21 F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67:605–608, 1972.
 - 22 William H. Kautz and Richard C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Information Theory*, 10(4):363–377, 1964. doi:10.1109/TIT.1964.1053689.
 - 23 Joseph L. Gastwirth and Patricia A. Hammick. Estimation of the prevalence of a rare disease, preserving the anonymity of the subjects by group testing: application to estimating the prevalence of aids antibodies in blood donors. *Journal of Statistical Planning and Inference.*, 22(1):15–27, 1989.
 - 24 C. H. Li. A sequential method for screening experimental variables. *J. Amer. Statist. Assoc.*, 57:455–477, 1962.
 - 25 Anthony J. Macula and Leonard J. Popyack. A group testing method for finding patterns in data. *Discrete Applied Mathematics*, 144(1-2):149–157, 2004. doi:10.1016/j.dam.2003.07.009.
 - 26 Hung Q. Ngo and Ding-Zhu Du. A survey on combinatorial group testing algorithms with applications to DNA library screening. In *Discrete Mathematical Problems with Medical Applications, Proceedings of a DIMACS Workshop, December 8-10, 1999*, pages 171–182, 1999. doi:10.1090/dimacs/055/13.
 - 27 Ely Porat and Amir Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Trans. Information Theory*, 57(12):7982–7989, 2011. doi:10.1109/TIT.2011.2163296.
 - 28 Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *CoRR*, abs/1404.5568, 2014. URL: <http://arxiv.org/abs/1404.5568>, arXiv:1404.5568.
 - 29 Jens Schlaghoff and Eberhard Triesch. Improved results for competitive group testing. *Combinatorics, Probability & Computing*, 14(1-2):191–202, 2005. doi:10.1017/S0963548304006649.
 - 30 M. Sobel and P. A. Groll. Group testing to eliminate efficiently all defectives in a binomial sample. *Bell System Tech. J.*, 38:1179–1252, 1959.
 - 31 William H. Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology*, 1985.
 - 32 Keith H. Thompson. Estimation of the proportion of vectors in a natural population of insects. *Biometrics*, 18(4):568–578, 1962.
 - 33 S. D. Walter, S. W. Hildreth, and B. J. Beaty. Estimation of infection rates in population of organisms using pools of variable size. *Am J Epidemiol.*, 112(1):124–128, 1980.
 - 34 Jack K. Wolf. Born again group testing: Multiaccess communications. *IEEE Trans. Information Theory*, 31(2):185–191, 1985. doi:10.1109/TIT.1985.1057026.

5 Appendix

► **Theorem 10.** Let $1/d^{\omega(1)} \geq \delta \geq 1/(2(n-d+1))$. Let A be a randomized adaptive algorithm that for any set of defective items I of size d or $d+1$, with probability at least $1-\delta$, exactly determines the number of defective items $|I|$. Algorithm A must ask at least

$$(1-o(1))d \log \frac{1}{\delta}$$

expected number of queries.

When $\delta \leq 1/(2(n-d+1))$ then A must ask at least $(1-o(1))d \log n$ queries which is, asymptotically, the query complexity of finding the defective items with $\delta=0$ error.

Proof. Let $A(r)$ be a randomized algorithm that for $I \subseteq [n]$, $|I| \in \{d, d+1\}$ and oracle \mathcal{O}_I , determines $|I|$ with probability at least $1-\delta$ where r is the random seed of the algorithm.

18:12 Algorithm to Determine the Number of Defectives

Let $X(I, r)$ be a random variable that is equal to 1 if $A(r, \mathcal{O}_I) \neq |I|$ and 0 otherwise. Then for any $I \subseteq [m]$, $\mathbf{E}_r[X(I, r)] \leq \delta$. Let $m = \lceil \tau/\delta \rceil + d - 1 \leq n$ where $\tau = 1/(d \log(1/(d\delta)))$. Consider any $J \subseteq [m]$, $|J| = d$. For any such J let

$$Y_J(r) = X(J, r) + \sum_{i \in [m] \setminus J} X(J \cup \{i\}, r).$$

Then for every $J \subseteq [m]$ of size d , $\mathbf{E}_r[Y_J(r)] \leq (m - d + 1)\delta \leq \tau$. Therefore for a random uniform $J \subseteq [m]$ of size d we have $\mathbf{E}_r[\mathbf{E}_J[Y_J(r)]] = \mathbf{E}_J[\mathbf{E}_r[Y_J(r)]] \leq \tau$. Therefore, by Markov's inequality, for $\eta = 1/\log(1/(d\delta))$,

$$\mathbf{Pr}_r[\mathbf{E}_J[Y_J(r)] > \eta] \leq \frac{\tau}{\eta} = \frac{1}{d}.$$

That is, for random r , with probability at least $1 - 1/d$, at least $1 - \eta$ fraction of the sets $J \subseteq [m]$ of size d satisfies $Y_J(r) = 0$. Let R be the set of seeds r such that at least $1 - \eta$ fraction of the sets $J \subseteq [m]$ of size d satisfies $Y_J(r) = 0$. Then $\mathbf{Pr}_r[R] \geq 1 - 1/d$. Let $r_0 \in R$. Let C_{r_0} be the set of all $J \subseteq [m]$ of size d such that $Y_J(r_0) = 0$. Then

$$|C_{r_0}| \geq (1 - \eta) \binom{m}{d} = (1 - \eta) \binom{\lceil \tau/\delta \rceil + d - 1}{d}.$$

Consider the deterministic algorithm $A(r_0)$. As in Theorem 4, for every two distinct $J_1, J_2 \in C_{r_0}$, there is a query $Q \in \mathcal{Q}(A(r_0), \mathcal{O}_{J_1})$ such that $Q(J_1) \neq Q(J_2)$. Then by Lemma 3, the average-case query complexity of $A(r_0)$ is at least

$$\log |C_{r_0}| \geq \log(1 - \eta) \binom{\lceil \tau/\delta \rceil + d - 1}{d} \geq d \log \frac{\tau}{d\delta} - \log \frac{1}{1 - \eta}.$$

Let $Z(\mathcal{O}_I, r) = |\mathcal{Q}(A(r), \mathcal{O}_I)|$. We have shown that for every $r \in R$,

$$\mathbf{E}_{I \in C_r}[Z(\mathcal{O}_I, r)] \geq d \log \frac{\tau}{d\delta} - \log \frac{1}{1 - \eta}.$$

Therefore for every $r \in R$,

$$\begin{aligned} \mathbf{E}_I[Z(\mathcal{O}_I, r)] &\geq \mathbf{E}_I[Z(\mathcal{O}_I, r) | I \in C_r] \mathbf{Pr}[I \in C_r] \\ &\geq (1 - \eta) \left(d \log \frac{\tau}{d\delta} - \log \frac{1}{1 - \eta} \right). \end{aligned}$$

Therefore

$$\begin{aligned} \mathbf{E}_I \mathbf{E}_r[Z(\mathcal{O}_I, r)] &= \mathbf{E}_r \mathbf{E}_I[Z(\mathcal{O}_I, r)] \\ &\geq \mathbf{E}_r[\mathbf{E}_I[Z(\mathcal{O}_I, r) | r \in R] \mathbf{Pr}[r \in R]] \\ &\geq \left(1 - \frac{1}{d}\right) (1 - \eta) \left(d \log \frac{\tau}{d\delta} - \log \frac{1}{1 - \eta} \right). \end{aligned}$$

Therefore there is I such that

$$\mathbf{E}_r[Z(\mathcal{O}_I, r)] \geq \left(1 - \frac{1}{d}\right) (1 - \eta) \left(d \log \frac{\tau}{d\delta} - \log \frac{1}{1 - \eta} \right)$$

and then

$$\mathbf{E}_r[Z(\mathcal{O}_I, r)] \geq (1 - o(1)) d \log \frac{1}{\delta}. \quad \blacktriangleleft$$

On the Hardness of Computing an Average Curve

Kevin Buchin 

Department of Mathematics and Computing Science, TU Eindhoven, The Netherlands
k.a.buchin@tue.nl

Anne Driemel

University of Bonn, Hausdorff Center for Mathematics, Bonn, Germany
driemel@cs.uni-bonn.de

Martijn Struijs 

Department of Mathematics and Computing Science, TU Eindhoven, The Netherlands
m.a.c.struijs@tue.nl

Abstract

We study the complexity of clustering curves under k -median and k -center objectives in the metric space of the Fréchet distance and related distance measures. Building upon recent hardness results for the minimum-enclosing-ball problem under the Fréchet distance, we show that also the 1-median problem is NP-hard. Furthermore, we show that the 1-median problem is W[1]-hard with the number of curves as parameter. We show this under the discrete and continuous Fréchet and Dynamic Time Warping (DTW) distance. This yields an independent proof of an earlier result by Bulteau et al. from 2018 for a variant of DTW that uses squared distances, where the new proof is both simpler and more general. On the positive side, we give approximation algorithms for problem variants where the center curve may have complexity at most ℓ under the discrete Fréchet distance. In particular, for fixed k, ℓ and ε , we give $(1 + \varepsilon)$ -approximation algorithms for the (k, ℓ) -median and (k, ℓ) -center objectives and a polynomial-time exact algorithm for the (k, ℓ) -center objective.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Curves, Clustering, Algorithms, Hardness, Approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.19

1 Introduction

Clustering is an important tool in data analysis, used to split data into groups of similar objects. Their dissimilarity is often based on distance between points in Euclidean space. However, the dissimilarity of polygonal curves is more accurately measured by specialised measures: Dynamic Time Warping (DTW) [23], continuous and discrete Fréchet distance [1, 13].

We focus on *centroid-based clustering*, where each cluster has a center curve and the quality of the clustering is based on the similarity between the center and the elements inside the cluster. In particular, given a distance measure δ , we consider the following problems:

► **Problem 1 (k -median for curves with distance δ).** Given a set $\mathcal{G} = \{g_1, \dots, g_m\}$ of polygonal curves, find a set $\mathcal{C} = \{c_1, \dots, c_k\}$ of polygonal curves with at most n vertices each that minimizes $\sum_{g \in \mathcal{G}} \min_{i=1}^k \delta(c_i, g)$.

► **Problem 2 (k -center for curves with distance δ).** Given a set $\mathcal{G} = \{g_1, \dots, g_m\}$ of polygonal curves, find a set $\mathcal{C} = \{c_1, \dots, c_k\}$ of polygonal curves with at most n vertices each that minimizes $\max_{g \in \mathcal{G}} \min_{i=1}^k \delta(c_i, g)$.

For points in Euclidean space, the most widely-used centroid-based clustering problem is k -means, in which the distance measure δ is the squared Euclidean distance. But also for general metric spaces the k -median problem is well studied, often in the context of the



© Kevin Buchin, Anne Driemel, and Martijn Struijs;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 19; pp. 19:1–19:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

closely related facility location problem [20]. In general metric spaces usually, the *discrete k -median problem* is studied, where the centers must be selected from a finite set F , and are called facilities.

For clustering curves, limiting the possible centers to a finite set of “facilities” is unnecessarily restrictive. In this paper, we are therefore interested in the *unconstrained k -median problem*, where a center can be any element of the metric space (as in the case of k -means). Often, we will simply write k -median problem to denote the unconstrained version. In this paper, we are in particular interested in the complexity of the 1-median problem, which we refer to as *average curve problem*.

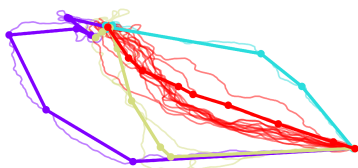
Hardness of the average curve problem

While clustering on points for general k in the plane or higher dimension is often NP-hard [22], many point clustering problems can be solved efficiently when $k = 1$ in low dimension. For instance, the 1-center problem in the plane can be solved in linear time [21], and there are practical algorithms for higher dimensional Euclidean space [15]. In contrast, the 1-center problem (i.e., the minimum enclosing ball problem) for curves under the discrete and continuous Fréchet distance is already NP-hard in 1D [6]. In this paper, we show that also the average curve problem, i.e. the 1-median problem, is NP-hard. We show this for the discrete and for the continuous Fréchet distance, and for the dynamic time-warping (DTW) distance. Variants of the DTW distance differ in the norm used for comparing pairs of points, and how that norm is used, see Section 1.1 for details. Our results apply to a large class of variants of DTW. For the frequently used variant of DTW using the squared Euclidean distance, Bulteau et al. [8] recently showed that the average curve problem is NP-hard and even W[1]-hard when parametrized in the number of input curves m and there exists no $f(m) \cdot n^{o(m)}$ -time algorithm unless the Exponential Time Hypothesis (ETH) fails¹. Because of its importance in time series clustering, there are many heuristics for the average curve problem under DTW [18, 23]. Brill et al. [4] showed that dynamic programming yields an exponential-time exact algorithm and additionally show the problem can be solved in polynomial time when both the input curves and center curve use only vertices from $\{0, 1\}$.

Approximation algorithms

Since both the k -center and the k -median problem for curves are already NP-hard for $k = 1$ in 1D, we further study efficient approximation algorithms for these problems. For approximation in metric spaces, the discrete and unconstrained k -median (likewise for k -center) are closely related: any set of curves that realises an α -approximation for the discrete k -median problem realises an 2α -approximation for the unconstrained k -median problem. There is an elegant $O(kn)$ time 2-approximation algorithm for the k -center problem in metric spaces [17]. This approximation factor is tight for clustering curves under the discrete Fréchet distance [6]. Finding approximate solutions for k -median is more challenging: the best known polynomial-time approximation algorithm for discrete k -median in general metric space achieves a factor of $3 + \varepsilon$ for any $\varepsilon > 0$ [2] and it is NP-hard to achieve an approximation factor of $1 + 2/e$ [19]. Unconstrained clustering of curves may result in centers of high complexity. To avoid overfitting and to obtain a compact representation of the data, we look at a variant of the clustering problems with center curves of at most a fixed complexity, denoted by ℓ . More formally, the (k, ℓ) -center problem is to find a set of curves

¹ See e.g. [11] for background on parametrized complexity



■ **Figure 1** (k, ℓ) -center clustering of pigeon flight paths computed by the algorithm of [7].

$\mathcal{C} = \{c_1, \dots, c_k\}$, each of complexity at most ℓ , that minimizes $\max_{g \in \mathcal{G}} \min_{i=1}^k \delta(c_i, g)$. The (k, ℓ) -median problem is defined analogously. Although the general case for this variant is still NP-hard, we can find efficient algorithms when k and ℓ are fixed. The (k, ℓ) -center and (k, ℓ) -median problems were introduced by Driemel et al. [12], who obtained an $\tilde{O}(mn)$ -time $(1 + \varepsilon)$ -approximation algorithm for the (k, ℓ) -center and (k, ℓ) -median problem under the Fréchet distance for curves in 1D, assuming k, ℓ, ε are constant. In [6], Buchin et al. gave polynomial-time constant-factor approximation algorithms for the (k, ℓ) -center problem under the discrete and continuous Fréchet distance for curves in arbitrary dimension. These approximation algorithms have led to efficient implementations of heuristics for the center version showing that the considered clustering formulations are useful in practice [7]. See Figure 1 for an example of a computed clustering. This encourages further study of the median variants of the problem.

1.1 Definitions of distance measures

Let x be a polygonal curve, defined by a sequence of vertices x_1, \dots, x_n from \mathbb{R}^d where consecutive vertices are connected by straight line segments. We call the number of vertices of x the *complexity*, denoted by $|x|$. Given a pair of polygonal curves x, y , a *warping path* between them is a sequence $W = \langle w_1, \dots, w_L \rangle$ of index pairs $w_l = (i_l, j_l)$ from $\{1, \dots, |x|\} \times \{1, \dots, |y|\}$ such that $w_1 = (1, 1)$, $w_L = (|x|, |y|)$, and $(i_{l+1} - i_l, j_{l+1} - j_l) \in \{(0, 1), (1, 0), (1, 1)\}$ for all $1 \leq l < L$. We say two vertices x_i, y_j are *matched* if $(i, j) \in W$.

Denote the set of all warping paths between curves x and y by $\mathcal{W}_{x,y}$. For any integers $p, q \geq 1$, we define the Dynamic Time Warping Distance between x and y as

$$\text{DTW}_p^q(x, y) := \left(\min_{W \in \mathcal{W}_{x,y}} \sum_{(i,j) \in W} \|x_i - y_j\|^p \right)^{q/p},$$

where $\|\cdot\|$ denotes the Euclidean norm. In text, we refer to DTW_p^q also as (p, q) -DTW. Similarly, define the discrete Fréchet distance between x, y as

$$d_{dF}(x, y) := \min_{W \in \mathcal{W}_{x,y}} \max_{(i,j) \in W} \|x_i - y_j\|.$$

The *continuous* Fréchet distance is defined with a reparametrization $f : [0, 1] \rightarrow [0, 1]$, which is a continuous injective function with $f(0) = 0$ and $f(1) = 1$. We say two points on x and y are *matched* if $f(i) = j$. Denote the set of all reparametrizations by \mathcal{F} , then the continuous Fréchet distance is given by

$$d_F(x, y) := \inf_{f \in \mathcal{F}} \max_{\alpha \in [0,1]} \|x(f(\alpha)) - y(\alpha)\|.$$

19:4 On the Hardness of Computing an Average Curve

■ **Table 1** Overview of results. In these tables, n denotes the length of the input curves, m denotes the number of input curves and d denotes the ambient dimension of the curves.

(a) Results on exact computation.

Problem	Result	Restrictions	Reference
1-median, DTW_p^q	$O(n^{2m+1}2^m m)$	$d = 1$	Brill et al. [4]
	$O(mn^{1.87})$	Binary	Schaar et al. [28]
	NP-hard W[1]-hard in m	$p = q = 2$	Bulteau et al. [8]
	NP-hard W[1]-hard in m	$p, q \in \mathbb{N}$	Theorem 7
1-median, Fréchet	NP-hard W[1]-hard in m		Theorem 4
1-center, discrete Fréchet	NP-hard		Buchin et al. [6]
(k, ℓ) -center, discrete Fréchet	$O((mn)^{2k\ell+1} k\ell \log(mn))$	$d \leq 2$	Theorem 13

(b) Approximation algorithms. (In stating the running times we assume k , ℓ , and ε are constants independent of n and m .)

Problem	Result	Approx factor	Restrictions	Reference
(k, ℓ) -median, continuous Fréchet	$\tilde{O}(nm)$	$(1 + \varepsilon)$	$d = 1$	Driemel et al. [12]
(k, ℓ) -median, discrete Fréchet	$\tilde{O}(nm)$	65		Driemel et al. [12]
	$\tilde{O}(m^2(m+n))$	12		Theorem 10
	$\tilde{O}(nm)$	$(1 + \varepsilon)$	$k = 1$	Theorem 12
	$\tilde{O}(nm^{dk\ell+1})$	$(1 + \varepsilon)$	$k > 1$	Theorem 12
(k, ℓ) -center, discrete Fréchet	$\tilde{O}(nm)$	3		Buchin et al. [6]
	$\tilde{O}(nm)$	$(1 + \varepsilon)$		Theorem 9

1.2 Results

We show that the average curve problem for discrete and continuous Fréchet distance in 1D is NP-complete, W[1]-hard when parametrized in the number of curves m , and admits no $f(m) \cdot n^{o(m)}$ -time algorithm unless ETH fails. In addition, we prove the same hardness results of the average curve problem for the (p, q) -DTW distance for any $p, q \in \mathbb{N}$.

This is an independent proof that is simpler and more general than the result by Bulteau et al. [8]. Their hardness result holds for the case of the $(2, 2)$ -DTW distance, which is widely-used. Other common variants, covered by our proof, are $(1, 1)$ -DTW, i.e., (non-squared) Euclidean distance and Manhattan distance in 1D [16], $(2, 1)$ -DTW, and more generally $(p, 1)$ -DTW [26, 27]. Note that, while we define $(p, 1)$ -DTW in terms of the p th power of the Euclidean norm, our hardness results also apply to the p th power of the L_p -norm, since these norms are equal in 1D. Another difference is that hardness construction by Bulteau et al. [8] uses binary input curves and a center curve that is not restricted to a bounded set of vertices, while in our construction both the input curves and the center curve use only vertices from $\{-1, 0, 1\}$. This means we answer a question by Brill et al. [4], who asked whether their result can be extended to obtain a polynomial time algorithm when all curves are restricted to sets of 3 vertices, in the negative.

Since our and other hardness results exclude efficient algorithms for the (k, ℓ) -center or ℓ -median clustering without further assumptions, we investigate other approaches with provable guarantees. In particular, we give a $(1 + \varepsilon)$ -approximation algorithm that runs in $\tilde{O}(mn)$ time and a polynomial-time exact algorithm to solve the (k, ℓ) -center problem for the discrete Fréchet distance, when k, ℓ , and ε are fixed. For the (k, ℓ) -median problem under the discrete Fréchet distance, we give a polynomial time 12-approximation algorithm, and an $(1 + \varepsilon)$ -approximation algorithm that runs in polynomial time when k, ℓ , and ε are fixed. Table 1 gives an overview of our results.

2 Hardness of the average curve problem for discrete and continuous Fréchet

In this section, we will show that the 1-median problem (or average curve problem) is NP-hard for the discrete and continuous Fréchet distance. The average curve problem for the discrete Fréchet distance is as follows: given a set of curves \mathcal{G} and an integer r , determine whether there exists a center curve c such that $\sum_{g \in \mathcal{G}} d_{dF}(c, g) \leq r$. We will show that this problem is NP-hard. To find a reasonable algorithm, we can look at a parametrized version of the problem. A natural parameter is the number of input curves, which we will denote by m . However, we will show that this parametrized problem is W[1]-hard, which rules out any $f(m) \cdot n^{O(1)}$ -time algorithm, unless FPT = W[1]. To achieve these reductions, we create a reduction from a variant of the shortest common supersequence (SCS) problem.

2.1 The FCCS problem

To show the hardness of the average curve problem for the Fréchet and DTW distance, we reduce from a variant of the *Shortest Common Supersequence* (SCS) problem, which we will call the *Fixed Character Common Supersequence* (FCCS) problem. If s is a string and x is a character, $\#_x(s)$ denotes the number of occurrences of x in s .

► **Problem 3 (Shortest Common Supersequence (SCS)).** *Given a set S of m strings with length at most n over the alphabet Σ and an integer t , does there exist a string s^* of length t that is a supersequence of each string $s \in S$?*

► **Problem 4 (Fixed Character Common Supersequence (FCCS)).** *Given a set S of m strings with length at most n over the alphabet $\Sigma = \{A, B\}$ and $i, j \in \mathbb{N}$, does there exist a string s^* with $\#_A(s^*) = i$ and $\#_B(s^*) = j$ that is a supersequence of each string $s \in S$?*

The SCS problem with a binary alphabet is known to be NP-hard [25] and W[1]-hard [24]. The same holds for FCCS:

► **Lemma 1.** *The FCCS problem is NP-hard. The FCCS problem with m as parameter is W[1]-hard. There exists no $f(m) \cdot n^{o(m)}$ time algorithm for FCCS unless ETH fails.*

Proof. We reduce from SCS with the binary alphabet $\{A, B\}$ to FCCS. Given an instance (S, t) of SCS, construct $S' = \{s + AB^{2t}A + c(s) \mid s \in S\}$, where $c(s)$ denotes the string constructed by replacing all A characters in s by B and vice versa, and $+$ denotes string concatenation. We reduce to the instance $(S', t + 2, 3t)$ of FCCS and claim that (S, t) is a true instance of SCS if and only if $(S', t + 2, 3t)$ is a true instance of FCCS.

If (S, t) is a true instance of SCS, then there exists a string q of length t that is a supersequence of each string in S . Therefore, the string $q' = q + AB^{2t}A + c(q)$ is a supersequence of all strings in S' . Since $\#_A(q') = 2 + \#_A(q + c(q)) = 2 + t$ and $\#_B(q') = 2t + \#_B(q + c(q)) = 3t$, $(S', t + 2, 3t)$ is a true instance of FCCS.

If $(S', t + 2, 3t)$ is a true instance of FCCS, there is string q' with $\#_A(q') = t + 2$ and $\#_B(q') = 3t$ that is a supersequence of each string $s' \in S'$. Consider a pair of strings $s'_1 = s_1 + AB^{2t}A + c(s_1)$ and $s'_2 = s_2 + AB^{2t}A + c(s_2)$ from S' . If there is no matching such that the first character of the $AB^{2t}A$ substring in s'_1 is matched to the same character of q' as the first character of that substring in s'_2 , then q' is a supersequence of $AB^{2t}AB^{2t}A$ and so $\#_B(q') > 3t$, a contradiction. By symmetry, the same holds for the last character of the substring $AB^{2t}A$ and therefore $q = q_1 + q_2 + q_3$, where q_1 is a supersequence of S , q_2 is a supersequence of $AB^{2t}A$ and q_3 is a supersequence of $\{c(s) \mid s \in S\}$. Note that $c(q_3)$ is a supersequence of S . Also, $\#_A(q_1) + \#_A(c(q_3)) = \#_A(q) - \#_A(q_2) \leq t$ and $\#_B(q_1) + \#_B(c(q_3)) = \#_B(q) - \#_B(q_2) \leq t$. So, $|q_1| + |c(q_3)| \leq 2t$, which means that $|q_1| \leq t$ or $|c(q_3)| \leq t$ and thus (S, t) is a true instance of SCS.

Note that this reduction is both a polynomial-time reduction and a parametrized reduction in the parameter m . Since the SCS problem over the binary alphabet $\{A, B\}$ is NP-hard [25] and W[1]-hard when parametrized with the number of strings m [24], the first two parts of the claim follow. The final part of the claim follows from the fact that this reduction

Together with the reduction from [24], we have a parametrized reduction from CLIQUE with a linear bound on the parameter, so the final part of the claim follows [11, Obs. 14.22]. ◀

2.2 Complexity of the average curve problem under the discrete and continuous Fréchet distance

We will show the hardness of finding the average curve under the discrete and continuous Fréchet distance via the following reduction from FCCS. Given an instance (S, i, j) of FCCS, we construct a set of curves using the following vertices in \mathbb{R} : $g_a = -1$, $g_b = 1$, $g_A = -3$, and $g_B = 3$. For each string $s \in S$, we map each character to a subcurve in \mathbb{R} :

$$A \rightarrow (g_a g_b)^{i+j} g_A (g_b g_a)^{i+j} \quad B \rightarrow (g_b g_a)^{i+j} g_B (g_a g_b)^{i+j}.$$

The curve $\gamma(s)$ is constructed by concatenating the subcurves resulting from this mapping, $G = \{\gamma(s) \mid s \in S\}$ denotes the set of these curves. Additionally, we use the curves

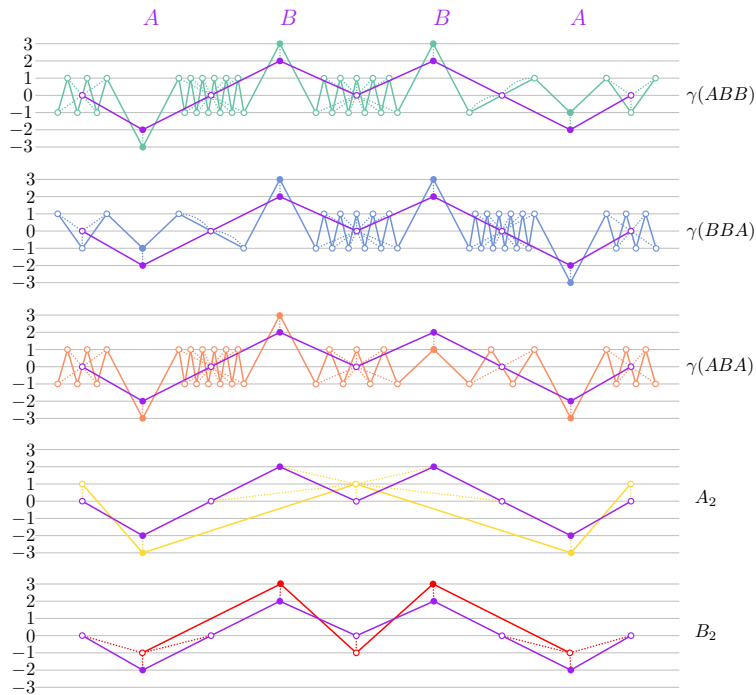
$$A_i = g_b (g_A g_b)^i \quad B_j = g_a (g_B g_a)^j.$$

We will call subcurves containing only g_A or g_B vertices *letter gadgets* and subcurves containing only g_a or g_b vertices *buffer gadgets*. Let $R_{i,j} = \{A_i, B_j\}$. We reduce to the instance $(G \cup R_{i,j}, r)$ of the average curve problem, where $r = |S| + 2$. We use the same construction for the discrete and continuous case. We call the interval of points p on a subcurve $g_b g_A g_b$ with $p < -1$ an *A-peak*, and the interval of points p on a subcurve $g_a g_B g_a$ with $p > 1$ a *B-peak*. A curve $\gamma(s)$ has exactly one peak for every letter in s .

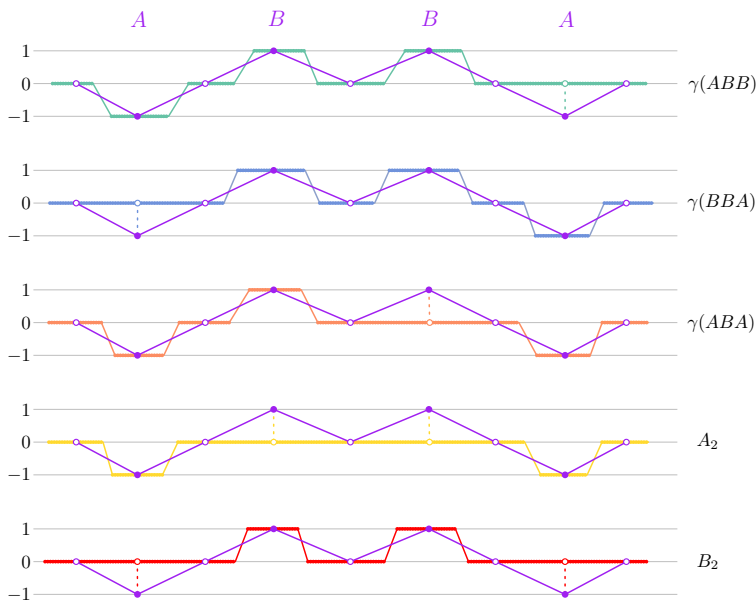
For an example of this construction, take $S = \{ABB, BBA, ABA\}$, $i = 2$, $j = 2$. Then $ABBA$ is a supersequence of S with the correct number of characters. Note that the curve with vertices $0g_A 0g_B 0g_B 0g_A 0$ has a (discrete) Fréchet distance of at most 1 to the curves in $G \cup R_{i,j}$, see Figure 2, so the sum of those distances is at most $|S| + 2 = r$.

► **Lemma 2.** *If (S, i, j) is a true instance of FCCS, then $(G \cup R_{i,j}, r)$ is a true instance of the average curve problem for discrete and continuous Fréchet.*

Proof. We will show the proof for the discrete Fréchet distance. Since the discrete Fréchet distance is an upper bound of the continuous version, this proves the continuous case as well.



■ **Figure 2** Five curves from $G \cup R_{i,j}$ in the reduction for the Fréchet average curve problem and a center curve constructed from $ABBA$ (purple) as in Lemma 2. Matchings are indicated by dotted lines. Note that each of these matchings achieves a (discrete) Fréchet distance of 1.



■ **Figure 3** Five curves from $G \cup R_{i,j}$ in the reduction for the DTW average curve problem and a center curve constructed from the string $ABBA$ (purple) as in Lemma 5. Fat horizontal lines indicate β consecutive vertices. Vertices that match at distance 0 touch, vertices that match at distance 1 are indicated by dotted lines. The center has 1 mismatch with the first 3 curves and 2 with the final two, so the total cost here is $3 \cdot (1^p)^{q/p} + 2\alpha \cdot (2 \cdot 1^p)^{q/p} = 3 + 2\alpha \cdot 2^q$.

19:8 On the Hardness of Computing an Average Curve

Since (S, i, j) is a true instance of FCCS, there exists a common supersequence s^* of S with $\#_A(s^*) = i$ and $\#_B(s^*) = j$. Construct the curve c of complexity $2|s^*| + 1$, given by

$$c_l = \begin{cases} 0 & \text{if } l \text{ is odd} \\ -2 & \text{if } l \text{ is even and } s_{l/2}^* = A, \\ 2 & \text{if } l \text{ is even and } s_{l/2}^* = B \end{cases}$$

for each $l \in \{1, \dots, 2|s^*| + 1\}$. Let $s \in S$, then note that the sequence of letter gadgets in $\gamma(s)$ is a subsequence of the letter gadgets in c , because s is a subsequence of s^* . So, all letter gadgets in $\gamma(s)$ can be matched with a letter gadget in c , the remaining letter gadgets in c with a buffer gadget in $\gamma(s)$ and all remaining buffer gadgets with another buffer gadget, such that $d_{dF}(c, \gamma(s)) \leq 1$. For the matching with A_i , note that c has exactly i g_A vertices, so these can be matched with the i g_A vertices in A_i . All other vertices in c have distance 1 to the remaining buffer gadgets in A_i , so $d_{dF}(c, A_i) \leq 1$. Analogously, $d_{dF}(c, B_j) \leq 1$. So, we get $\sum_{g \in G \cup R_{i,j}} d_{dF}(c, g) = \sum_{s \in S} d_{dF}(c, \gamma(s)) + d_{dF}(c, A_i) + d_{dF}(c, B_j) \leq |S| + 2 = r$, and $(G \cup R_{i,j}, r)$ is a true instance of average curve for discrete Fréchet. \blacktriangleleft

► **Lemma 3.** *If $(G \cup R_{i,j}, r)$ is a true instance of the average curve problem for discrete and continuous Fréchet, then (S, i, j) is a true instance of FCCS.*

We give a sketch of the proof, see Appendix A for the full proof. Since $(G \cup R_{i,j}, r)$ is a true instance of the average curve problem for continuous Fréchet, there exists a curve c^* such that $\sum_{g \in G \cup R_{i,j}} d_F(c^*, g) \leq r = |S| + 2\alpha$. We show $d_{dF}(c, g) = 1$ for all $g \in G \cup R_{i,j}$ and any center curve c that exhibits this bound. It remains to show that such a center curve encodes a solution to the initial FCCS instance. Note that such a center curve is also a solution to the 1-center problem for this set of curves. We can now apply the proof of Lemma 33 from [5, 6], where the same gadgets were used in the reduction to the 1-center problem. \blacktriangleleft

► **Theorem 4.** *The average curve problem for discrete and continuous Fréchet distance is NP-hard. When parametrized in the number of input curves m , this problem is $W[1]$ -hard. There exists no $f(m) \cdot n^{o(m)}$ time algorithm for this problem unless ETH fails.*

Proof. By Lemmas 2 and 3, we have a valid reduction from FCCS to the average curve problem. Since this reduction runs in polynomial time and FCCS is NP-hard (Lemma 1), the average curve problem for discrete and continuous Fréchet is NP-hard. Note that the number of curves in the reduced average curve instance is $k + 2$, where k is the number of input sequences of the FCCS instance. So, together with the reduction from Lemma 1, this reduction is also a parametrized reduction from CLIQUE with a linear bound on the parameter to the average curve problem for discrete and continuous Fréchet with the number of curves as a parameter, which implies the remainder of the theorem [11, Obs. 14.22]. \blacktriangleleft

3 Hardness of the average curve problem for (p, q) -DTW

We will show that the average curve problem under the (p, q) -DTW distance is NP-hard for all $p, q \in \mathbb{N}$. This generalises the result of [8], who use different methods to achieve the same hardness results for the $(2, 2)$ -DTW average curve problem only. We again reduce from FCCS instance (S, i, j) . Given a string $s \in S$ over the binary alphabet $\{A, B\}$, we map each character to a subcurve in \mathbb{R} :

$$A \rightarrow g_0^\beta g_a^\beta g_0^\beta \quad B \rightarrow g_0^\beta g_b^\beta g_0^\beta,$$

where $g_0 = 0, g_a = -1, g_b = 1$ as before and β is a large constant that will be determined later. The curve $\gamma(s)$ is constructed by concatenating these subcurves and $G = \{\gamma(s) \mid s \in S\}$. We additionally use the curves

$$A_i = g_0^\beta (g_a^\beta g_0^\beta)^i \quad B_j = g_0^\beta (g_b^\beta g_0^\beta)^j.$$

Call any subcurve consisting of g_a or g_b vertices a letter gadget and any subcurve consisting of g_0 a buffer gadget. Let $R_{i,j}$ contain curves A_i and B_j , both with multiplicity α . We reduce to the instance $(G \cup R_{i,j}, r)$ of (p, q) -DTW average curve, where $r = \sum_{s \in S} (i + j - |s|)^{q/p} + \alpha(i^{q/p} + j^{q/p})$, $\beta = \lceil r/\varepsilon^q \rceil + 1$, $\alpha = |S|$ and $\varepsilon = 1 - (1 - \min_{x \in \{i,j\}} \frac{(x+1)^{q/p} - x^{q/p}}{4(i+j)^{q/p}})^{1/q}$.² See Figure 3 for an example of this construction with $S = \{ABB, BBA, ABA\}$ and $i = j = 2$.

The following definitions are used to prove Lemma 6. Take a vertex p on some center curve c^* . If $|p - g_a| < \varepsilon$, we call p an *A-signal vertex*. If $|p - g_b| < \varepsilon$ we call p an *B-signal vertex*. If p is not a signal vertex, then we call p a *buffer vertex*. Note that ε is chosen small enough such that no vertex is both an A- and B-signal vertex. We will show that the sequence of signal vertices in the curve satisfying $(G \cup R_{i,j}, r)$ is a supersequence satisfying (S, i, j) .

► **Lemma 5.** *If (S, i, j) is a true instance of FCCS, then $(G \cup R_{i,j}, r)$ is a true instance of (p, q) -DTW average curve.*

Proof. If (S, i, j) is a true instance of FCCS, then there exists a string s^* that is a supersequence of S , with $\#_A(s^*) = i$ and $\#_B(s^*) = j$. Construct the curve c of length $2(i + j) + 1$:

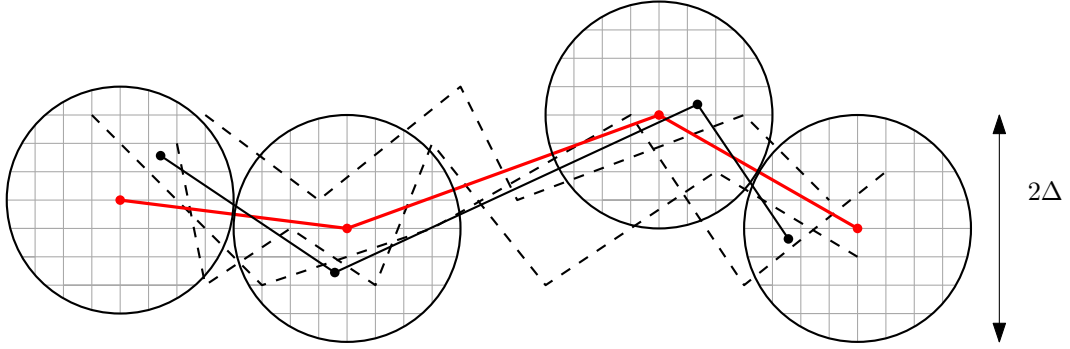
$$c_l = \begin{cases} 0 & \text{if } l \text{ is odd} \\ g_a & \text{if } l \text{ is even and } s_{l/2}^* = A, \\ g_b & \text{if } l \text{ is even and } s_{l/2}^* = B \end{cases}$$

for each $l \in \{1, \dots, 2(i + j) + 1\}$. Analogously to Lemma 2, we can match the letter gadgets from $\gamma(s)$ to g_A or g_B in c as s^* is a supersequence of s , the letter gadgets of A_i, B_j to g_A, g_B in c as the number of curves match, and g_0 vertices to buffer gadgets. This gives a matching such that $\sum_{g \in G \cup R_{i,j}} \text{DTW}_p^q(c, g) \leq r$. ◀

► **Lemma 6.** *If $(G \cup R_{i,j}, r)$ is a true instance of (p, q) -DTW average curve, then (S, i, j) is a true instance of FCCS.*

We give a sketch of the proof, for the full proof, see Appendix A. Let c^* be a center curve such that $\sum_{g \in G \cup R_{i,j}} \text{DTW}_p^q(c^*, g) \leq r$. Since $\varepsilon^q \cdot \beta > r$, each letter gadget must be matched to a signal vertex. Additionally, each signal vertex can only be matched to at most one letter gadget, because matching the buffer separating two gadgets costs at least $(1 - \varepsilon)^q \cdot \beta > r$. This means that the sequence of letter gadgets in $\gamma(s)$ is a subsequence of the sequence of signal vertices in c^* , so the sequence of signal vertices in c^* induces a supersequence s' of S . What remains to be proven is that s' doesn't use too many characters, i.e. that there are no

² Computing the values r, β, ε requires computing higher order roots. For simplicity, we assume that we can compute the exact values in polynomial time. However, this assumption is not necessary, as the construction also works if we use corresponding approximate values $\tilde{r}, \tilde{\varepsilon}, \tilde{\beta}$, as long as $\tilde{r} \in [r, r + \frac{1}{4} \min_{x \in \{i,j\}} (x+1)^{q/p} - x^{q/p}]$, $\tilde{\varepsilon} \leq \varepsilon$, and $\tilde{\beta} \geq \beta$. So, we are allowed to make an error of at least $\Omega((i+j)^{-1}) = \Omega(n^{-1})$, which we can do in polynomial time.



■ **Figure 4** Given an approximate $(1, \ell)$ -center curve (red) for a set of curves (dashed), the vertices of the optimal center curve (black) will be close to the hypercube grids around the vertices of the approximate center.

more than i A-signal vertices and j B-signal vertices on c^* . We prove this by deriving an upper bound on $\text{DTW}_p^q(c^*, A_i) + \text{DTW}_p^q(c^*, B_j)$ that cannot be achieved if c^* has too many signal vertices. ◀

► **Theorem 7.** *The average curve problem for the (p, q) -DTW distance is NP-hard, for any $p, q \in \mathbb{N}$. When parametrized in the number of input curves m , this problem is W[1]-hard. There exists no $f(m) \cdot n^{o(m)}$ time algorithm for this problem unless ETH fails.*

Proof. By Lemmas 5 and 6, we have a valid reduction from FCCS to the average curve problem. Since this reduction runs in polynomial time and FCCS is NP-hard (Lemma 1), the average curve problem for discrete and continuous Fréchet is NP-hard. Since the reduction runs in polynomial time (note that $1/\varepsilon$ can be bounded by a polynomial function in n , since p, q are constants, so β can be polynomially bounded) and the number of input curves is bounded by a linear function in $|S|$, the claim follows. ◀

4 Algorithms for (k, ℓ) -center and -median curve clustering

4.1 $(1 + \varepsilon)$ -approximation for (k, ℓ) -center clustering for discrete Fréchet distance in \mathbb{R}^d

In this section, we develop a $(1 + \varepsilon)$ -approximation algorithm for the (k, ℓ) -center problem under the discrete Fréchet distance that runs in $O(mn \log(n))$ time for fixed k, ℓ, ε . In this algorithm, we use hypercube grids $L_v(a, b)$ around a vertex v of width a and resolution b : take the axis-parallel d -dimensional hypercube centered at v of side-length a . Divide this hypercube into smaller hypercubes of side-length at most b . The grid $L_v(a, b)$ is the set of all vertices of the smaller hypercubes that intersect the ball of diameter a around v . See Figure 4 for an example. The algorithm is as follows: First, we compute a set of curves $\mathcal{C} = \{c_1, \dots, c_k\}$ that forms a 3-approximation for the (k, ℓ) -center problem, using the algorithm by Buchin et al. [6]. Let Δ be the cost of \mathcal{C} . Let V be the union of the hypercube grids $L_v(4\Delta, \frac{2\Delta\varepsilon}{3\sqrt{d}})$ over all vertices v of curves in \mathcal{C} . For every set of k center curves with complexity ℓ using only vertices from V , compute the clustering and cost as centers for G , and return the set with minimal cost.

In order to show this algorithm gives an $(1 + \varepsilon)$ -approximation, we use the following lemma to show that there is a set of k center curves that is close enough to the optimal solution:

► **Lemma 8.** *Let $k, \ell \in \mathbb{N}$, $\delta \in \mathbb{R}$ and $X > 0$. Suppose there are two sets $\mathcal{C} = \{c_1, \dots, c_k\}$ and $\mathcal{C}^* = \{c_1^*, \dots, c_k^*\}$, both containing k curves in \mathbb{R}^d of complexity ℓ . Additionally, suppose that for all curves $c^* \in \mathcal{C}^*$, there exists a curve $c \in \mathcal{C}$ such that $d_{dF}(c, c^*) \leq \delta$. Let $V = \{L_v(2\delta, 2\frac{X}{\sqrt{d}}) \mid v \text{ is a vertex of a curve in } \mathcal{C}\}$. Then there is a set of curves $\tilde{\mathcal{C}} = \{\tilde{c}_1, \dots, \tilde{c}_k\}$, using only vertices from V , such that $d_{dF}(c_i^*, \tilde{c}_i) \leq X$, $|\tilde{c}_i| = \ell$, for all $1 \leq i \leq k$.*

Proof. Let v be a vertex of a curve in \mathcal{C} , and let p be a point such that $\|p - v\| \leq \delta$. Then p lies inside one of the small hypercubes and so there is a vertex $p' \in L_v(2\delta, 2\frac{X}{\sqrt{d}})$ (a vertex of that small hypercube) such that $\|p - p'\| \leq \frac{\sqrt{d}}{2} \cdot \frac{2X}{\sqrt{d}} = X$. Let $c^* \in \mathcal{C}^*$. There exists a curve $c \in \mathcal{C}$ with $d_{dF}(c, c^*) \leq \delta$, which means that each vertex u of c^* has distance at most δ to some vertex v of c . So, there exists a vertex $v' \in L_v(2\delta, 2\frac{X}{\sqrt{d}})$ such that $\|u - v'\| \leq X$. Construct the curve \tilde{c} by connecting all such vertices v' by line segments. By construction, $d_{dF}(\tilde{c}, c^*) \leq \delta$, $|\tilde{c}| = \ell$, and all vertices of \tilde{c} are in V . So, we can take $\tilde{\mathcal{C}} = \{\tilde{c} \mid c^* \in \mathcal{C}^*\}$. ◀

By the triangle inequality, curves of distance at most $\varepsilon\Delta/3$ to an optimal solution are an $(1 + \varepsilon)$ -approximation. We use Lemma 8 that show there is such a set of curves in the hypercube grids our algorithm searches, leading to the following theorem:

► **Theorem 9.** *Given m input curves in \mathbb{R}^d , each of complexity at most n , and positive integers k, ℓ and some $0 < \varepsilon \leq 1$, we can compute an $(1 + \varepsilon)$ -approximation to the (k, ℓ) -center problem for the discrete Fréchet distance in $O(((Ck\ell)^{k\ell} + \log(\ell + n)) \cdot k\ell \cdot mn)$ time, with $C = \left(\frac{6\sqrt{d}}{\varepsilon} + 1\right)^d$.*

Proof. We first show that the algorithm above achieves this approximation ratio. Let \mathcal{C}^* be an optimal optimal solution for the (k, ℓ) -center problem, and O its cost. Let $c^* \in \mathcal{C}^*$, then there is a curve $g \in G$ such that $d_{dF}(c^*, g) \leq O$ (assuming without loss of generality that its cluster is non-empty). Since the solution \mathcal{C} has cost Δ , there is a $c \in \mathcal{C}$ such that $d_{dF}(c, g) \leq \Delta$. So, $d_{dF}(c, c^*) \leq d_{dF}(c, g) + d_{dF}(g, c^*) \leq 2\Delta$, and by Lemma 8 with $\delta = 2\Delta$ and $X = \varepsilon \cdot \Delta/3 \leq \varepsilon O$, there is a solution $\tilde{\mathcal{C}}$ with the properties in the Lemma. Since for any $g \in G$, there is a curve $c^* \in \mathcal{C}^*$ such that $d_{dF}(g, c^*) \leq O$, there is a $\tilde{c} \in \tilde{\mathcal{C}}$ such that $d_{dF}(g, \tilde{c}) \leq d_{dF}(g, c^*) + d_{dF}(\tilde{c}, c^*) \leq (1 + \varepsilon)O$. Since the algorithm returns the best solution using only from V , it returns a solution of cost at most that of $\tilde{\mathcal{C}}$, and is therefore an $1 + \varepsilon$ -approximation.

For the running time, computing the 3-approximation \mathcal{C} takes $O(k\ell mn \log(\ell + n))$ time [6]. A grid $L_v(a, b)$ has at most $(\lceil \frac{a}{b} \rceil + 1)^d$ vertices and the curves in \mathcal{C} have at most $k\ell$ vertices, so $|V| \leq k\ell(\lceil \frac{6\sqrt{d}}{\varepsilon} \rceil + 1)^d$. There are $O(|V|^{k\ell})$ solutions using only vertices from V , and we can test each solution in $O(k\ell mn)$ time: computing the discrete Fréchet distance between an input curve and a center curve takes $O(\ell n)$ time using dynamic programming, which we do for all km pairs of input and center curves. In total, we get a running time of $O((|V|^{k\ell} + \log(\ell + n)) \cdot k\ell mn)$. ◀

Note that we can use any α -approximation algorithm instead of the 3-approximation algorithm by Buchin et al. [6], if we scale the grids accordingly. This changes the value of C to $\left(\frac{2\alpha\sqrt{d}}{\varepsilon} + 1\right)^d$. If ε is very small, we can use this to get a smaller C constant by running our algorithm twice, first computing a 1.01-approximation, and using that approximation to compute the $(1 + \varepsilon)$ -approximation.

When ε and d are fixed constants, the algorithm from Theorem 9 yields fixed parameter tractability for the parameter $k + \ell$. There is no $(1 + \varepsilon)$ -approximation algorithm that is fixed parameter tractable in either k or ℓ separately (the problem is not even in XP, in fact),

unless $P = NP$. If we do not fix ℓ , then achieving an approximation factor strictly better than 2 is already NP-hard when $k = 1$ and $d = 1$ [6]. If we do not fix k and if $\ell = 1$, the (k, ℓ) -center problem for discrete Fréchet is equivalent to the Euclidean k -center problem, which is NP-hard to approximate within a factor of 1.82 for $d \geq 2$ [14].

4.2 Approximation algorithms for (k, ℓ) -median clustering for the discrete Fréchet distance in \mathbb{R}^d

We construct an $(1 + \varepsilon)$ -approximation for the (k, ℓ) -median problem for the discrete Fréchet distance with a similar approach as above: first compute a constant factor approximation, and then search in hypercube grids around the vertices of that approximation. The algorithm for the constant factor approximation is essentially the same as the approximation algorithm from [12] for 1D curves, except we use different subroutines and derive a tighter approximation bound. We first introduce some techniques we will use to get a 12-approximation. Given a polygonal curve γ , a *simplification* is a polygonal curve that is similar to γ , but has only a few vertices. Specifically, a *minimum error ℓ -simplification* $\bar{\gamma}$ of a curve γ is a curve of complexity at most ℓ that has a minimum distance to γ among all curves with complexity at most ℓ . We can compute a minimum error ℓ -simplification under the discrete Fréchet distance for a curve γ of complexity n in $O(n\ell \log n \log(n/\ell))$ time [3].

The 12-approximation algorithm goes as follows: First, compute a minimum error ℓ -simplification \bar{g} for each input curve g and let \bar{G} be the set of all simplified curves. Then, compute a 4-approximation for the k -median problem with $F = \bar{G}$ and $C = G$, using the algorithm by Jain et al. [19]. This yields a 12-approximation:

► **Theorem 10.** *Given m input curves in \mathbb{R}^d , each of complexity at most n , and positive integers k, ℓ , we can compute a 12-approximation to the (k, ℓ) -median problem for the discrete Fréchet distance in $O(m^3 + mn\ell(m + \log n \log(n/\ell)))$ time.*

Proof. We first show the approximation ratio. Let \mathcal{C}^* be the optimal solution to the (k, ℓ) -median problem with cost O , and let \mathcal{C} be the solution computed by our algorithm above. Each center curve c_i^* has a set $G_i^* \subseteq G$ as its cluster. Let c'_i be the minimum error ℓ -simplification of a curve c_i from G_i^* that has minimum distance to c_i^* . The curves $\mathcal{C}' = \{c'_1, \dots, c'_k\}$ are a 3-approximation to the (k, ℓ) -median problem: we have $\sum_{g \in G} \min_{i=1}^k d_{dF}(g, c'_i) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c'_i) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c_i^*) + d_{dF}(c_i^*, c_i) + d_{dF}(c_i, c'_i) \leq 3 \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c_i^*) = 3O$, where $d_{dF}(c'_i, c_i) \leq d_{dF}(c_i^*, c_i)$ because $|c'_i| = \ell$ and c'_i is a minimum error ℓ -simplification of c_i , and $d_{dF}(c_i, c_i^*) \leq d_{dF}(g, c_i^*)$ for all $g \in G_i^*$ by definition of c_i . \mathcal{C}' is some solution to the k -median problem with $F = \bar{G}$ and $C = G$ of cost at most $3O$, so the optimal solution to this problem has cost at most $3O$. Since we compute a 4-approximation for that problem, the result has cost at most $12O$.

For the running time, note that computing the simplification of all curves in G takes $O(mn\ell \log n \log(n/\ell))$ time. Then, we can compute the discrete Fréchet distances between pairs from $\bar{G} \times G$ in $O(m^2 \cdot \ell n)$ time, and run the algorithm by Jain et al. [19] in $O(m^3)$ time. ◀

We can modify the algorithm above to run in $\tilde{O}(mn)$ time when k, ℓ are constant: Compute \bar{G} as before, but now use the algorithm by Chen [10] to compute a 10.5-approximation to the k -median problem with $F = C = \bar{G}$. This gives a 42-approximation.

► **Lemma 11.** *Given m input curves in \mathbb{R}^d , each of complexity at most n , and positive integers k, ℓ , we can compute a 42-approximation to the (k, ℓ) -median problem for the discrete Fréchet distance in $O(mn\ell \log n \log(n/\ell) + \ell^2(mk + k^7 \log^5 m))$ time.*

Proof. The proof is similar to Theorem 10, but now simplifications are clustered instead of the original curves. We first show the approximation ratio. Given a cluster $G_i^* \subset \bar{G}$ from the optimal clustering with center c_i^* , let \bar{c}_i be the simplification of a curve g in this cluster such that $d_{dF}(\bar{c}_i, c_i^*)$ is minimal. The curves $\bar{\mathcal{C}} = \{\bar{c}_1, \dots, \bar{c}_k\}$ are a 4-approximation to the (k, ℓ) -median problem: we have $\sum_{g \in G} \min_{i=1}^k d_{dF}(g, \bar{c}_i) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, \bar{c}_i) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c_i^*) + d_{dF}(c_i^*, \bar{c}_i) \leq \sum_{i=1}^k \sum_{g \in G_i^*} 2 d_{dF}(g, c_i^*) \leq 2 \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c_i^*) + d_{dF}(g, c_i^*) \leq 2 \sum_{i=1}^k \sum_{g \in G_i^*} 2 d_{dF}(g, c_i^*) = 4O$, where $d_{dF}(\bar{c}_i, c_i^*) \leq d_{dF}(g, c_i^*)$ by definition of \bar{c}_i and $d_{dF}(\bar{g}, g) \leq d_{dF}(g, c_i^*)$ because $|c_i^*| = \ell$ and \bar{g} is a minimum error ℓ -simplification of g . Since we compute a 10.5-approximation to the problem for which $\bar{\mathcal{C}}$ is a solution, the approximation ratio $10.5 \cdot 4 = 42$.

Computing the simplification of all curves in G takes $O(mn\ell \log n \log(n/\ell))$ time. The algorithm by Chen [10] takes $O(mk + k^7 \log^5 m)$ time, so it uses at most that number of distance computations between curves in \bar{G} , which take $O(\ell^2)$ time each. ◀

We use the 42-approximation algorithm to compute an $(1 + \varepsilon)$ -approximation \mathcal{C} for the (k, ℓ) -median problem similar to section 4.1. Let $\mathcal{C} = \{c_1, \dots, c_k\}$ be the solution given by the 42-approximation algorithm above, and Δ its cost. If $k = 1$, let V be the union of the hypercube grids $L_v(4\Delta/m, \frac{\varepsilon\Delta}{21m\sqrt{d}})$ over all vertices v of curves in \mathcal{C} . If $k > 1$, let V be the union of the grids $L_v(4\Delta, \frac{\varepsilon\Delta}{21m\sqrt{d}})$ over the same vertices, instead. For every set of k center curves with complexity ℓ using only vertices from V , compute the clustering and cost (using the median objective) as centers for G , and return the set with minimal cost.

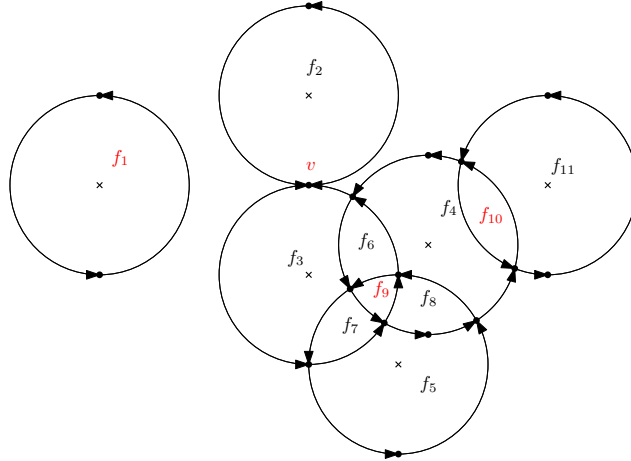
► **Theorem 12.** *Given m input curves in \mathbb{R}^d , each of complexity at most n , and positive integers k, ℓ and some $0 < \varepsilon \leq 1$, we can compute an $(1 + \varepsilon)$ -approximation to the (k, ℓ) -center problem for the discrete Fréchet distance in $O(mn\ell((C\ell)^\ell + \log n \log(n/\ell)))$ time when $k = 1$ with $C = \left(\frac{84\sqrt{d}}{\varepsilon}\right)^d$. When $k > 1$, we require $O((Ck\ell)^{k\ell} \cdot k\ell \cdot m^{dk\ell+1}n + mn\ell \log n \log(n/\ell) + \ell^2(mk + k^7 \log^5 m))$ time.*

Proof. We first show the approximation ratio. Let $\mathcal{C}^* = \{c_1^*, \dots, c_k^*\}$ be an optimal solution for the (k, ℓ) -median problem, $G_i^* \subset G$ the cluster induced by the center c_i^* , and O the total cost of this solution. Let $\tilde{\mathcal{C}} = \{\tilde{c}_1, \dots, \tilde{c}_k\}$ be a set of curves with complexity at most ℓ such that for all $1 \leq i \leq k$, there is a curve $\tilde{c}_j \in \tilde{\mathcal{C}}$ with $d_{dF}(c_i^*, \tilde{c}_j) \leq \varepsilon O/m$. Since $\sum_{g \in G} \min_{j=1}^k d_{dF}(g, \tilde{c}_j) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, \tilde{c}_j) \leq \sum_{i=1}^k \sum_{g \in G_i^*} d_{dF}(g, c_i^*) + d_{dF}(c_i^*, \tilde{c}_j) \leq \sum_{g \in G} \min_{i=1}^k d_{dF}(g, c_i^*) + \varepsilon O/m = (1 + \varepsilon)O$, the set $\tilde{\mathcal{C}}$ is an $(1 + \varepsilon)$ -approximation. We will show that there is such a set that uses only vertices of V .

If $k = 1$, then $d_{dF}(c_1, c_1^*) = \frac{1}{m} \sum_{g \in G} d_{dF}(c_1, c_1^*) \leq \frac{1}{m} \sum_{g \in G} d_{dF}(c_1, g) + d_{dF}(g, c_1^*) \leq (\Delta + O)/m \leq 2\Delta/m$. Applying Lemma 8 with $\delta = 2\Delta/m$ and $X = \varepsilon\Delta/(42m) \leq \varepsilon O/m$, there is a $(1 + \varepsilon)$ -approximation using only vertices of V .

Otherwise, if $k > 1$, then for each c_i^* there is a c_j such that the clusters of these centers share some curve $g \in G$. So, $d_{dF}(c_i^*, c_j) \leq d_{dF}(c_i^*, g) + d_{dF}(g, c_j) \leq O + \Delta \leq 2\Delta$. Applying Lemma 8 with $\delta = 2\Delta$ and $X = \varepsilon\Delta/(42m) \leq \varepsilon O/m$, there is a $(1 + \varepsilon)$ -approximation using only vertices of V .

For the running time, we have $|V| \leq k\ell(\lceil \frac{a}{b} \rceil + 1)^d$ when we use grids with width a and resolution b . If $k = 1$, $\frac{a}{b} = \frac{4\Delta/m}{\varepsilon\Delta/(21m\sqrt{d})} = \frac{84\sqrt{d}}{\varepsilon}$. If $k > 1$, $\frac{a}{b} = \frac{84m\sqrt{d}}{\varepsilon}$. The rest of the analysis is similar to that in Theorem 9. ◀



■ **Figure 5** An example configuration of $\mathcal{G} = (V, E)$. Crosses indicate the vertices from the curves in G , dots indicate vertices from V and all bounded faces are numbered. The maximal intersection regions are the faces f_1 and f_9 and the vertex v (in red). Note that while all arcs on the boundary of f_2 are convex for that face, f_2 is not maximal, since its boundary intersects the boundary of f_3 only at vertex v .

4.3 Exact algorithm for (k, ℓ) -center under discrete Fréchet in \mathbb{R}^2

For the (k, ℓ) -center problem under the discrete Fréchet distance in \mathbb{R}^2 , we can give a polynomial time algorithm if k and ℓ are fixed.

► **Theorem 13.** *Given a set of m curves G in the plane with at most n vertices each, we can find a solution to the (k, ℓ) -center problem for the discrete Fréchet distance in $O((mn)^{2k\ell+1}k\ell \log(mn))$ time.*

Proof. We first give an algorithm for the decision version of the problem: Given a set of m curves G in the plane with at most n vertices each and a positive real number r , does there exist a set of k center curves \mathcal{C} with at most ℓ vertices each such that $\min_{c \in \mathcal{C}} d_{dF}(c, g) \leq r$ for all $g \in G$?

For a solution \mathcal{C} of cost r , consider the planar subdivision formed by the circles of radius r centred at the vertices of the input curves. Observe that we can move the vertices of curves in \mathcal{C} to different positions within the same region of the subdivision without changing the cost. So, we select a single vertex per region and exhaustively test all sets with k curves of ℓ vertices that can be constructed by using only the selected vertices to determine if there exists a set of curves \mathcal{C} such that $\min_{c \in \mathcal{C}} d_{dF}(c, g) \leq r$ for all $g \in G$.

To find all maximal intersection regions, we first compute the planar graph $\mathcal{G} = (V, E)$, where V is the set of all intersection points between boundaries of disks centred around a vertex from our input curves with radius r and E is the set of arcs on the boundary of those disks ending at two intersection points. This graph has $O((nm)^2)$ vertices and arcs and can be computed in $O((nm)^2)$ time [9], see Figure 5 for an example.

By traversing the intersection points and arcs on the boundary, we can find the at most $O((nm)^2)$ maximal intersection regions. So, we test $O((mn)^{2k\ell})$ sets of center curves, for which we can test whether a single input curve has discrete Fréchet distance less than r to a single curve among the k center curves in $O(n\ell)$. This means the algorithm for the decision version takes $O((mn)^{2k\ell+1}k\ell)$ time.

To find a minimum r such that a (k, ℓ) -center exists, note that we only have to consider the decision problem for those r where the topology of the intersection regions in \mathcal{G} is different. If we start with $r = 0$ and gradually increase it, the topology of \mathcal{G} changes only when a new maximal intersection is created, which then consists of exactly one point p . This means that there is a subset of our disks such that point p is the earliest point where all disks have a non-empty intersection. So, p must be the center of the minimum enclosing disk for this subset of disks. Since a minimum enclosing disk is determined by at most 3 points, there can be at most one unique point for every triple in set of vertices of the input curves which give at most $O((mn)^3)$ distinct values of r where the topology of \mathcal{G} changes. By performing a binary search on these values, we can find the optimal value in $O(\log(mn))$ calls to the algorithm for the decision. ◀

5 Conclusion

In this paper, we have shown that the 1-median problem is computationally hard under the discrete Fréchet, continuous Fréchet, and DTW distance. A natural question is whether this problem is hard to approximate. Efficient constant factor approximation algorithms are known for the Fréchet distance (see Section 4.2), but not for DTW. If we extend our analysis in Lemma 3 to a solution c^* with cost $(1 + \varepsilon)r$ for some $\varepsilon > 0$, we can show $d_{dF}(c^*, g) \leq 1 + O(\varepsilon m)$ for all input curves g (where the constant is independent of other input parameters). Together with the approximation lower bound of 2 for 1-center under continuous Fréchet distance [29], this implies a lower bound of $1 + \Omega(\frac{1}{m})$ on the approximation factor for 1-median. If we do the same for Lemma 6, we get that it is hard to approximate 1-median under (p, q) -DTW for any factor $< 1 + 2((1 + \frac{1}{\min(i, j)})^{q/p} - 1)$. So, it remains an open problem to find a constant lower bound for approximating 1-median for these distance measures.

We have shown that computing a center curve for (p, q) -DTW is NP-hard even when both the center and input curves are ternary. Bultheau et al. [8] have shown that this problem is hard for $(2, 2)$ -DTW when the input is binary, but the center curve is unrestricted. Can this hardness result for binary inputs be extended to (p, q) -DTW? If both the center and input are binary, a center curve for $(2, 2)$ -DTW can be computed in polynomial time [28]. Can this be done for (p, q) -DTW? Can a mean be found in polynomial time if the input is binary, but the center restricted to be ternary?

On the positive side, we have given $(1 + \varepsilon)$ -approximation algorithms for (k, ℓ) -center and (k, ℓ) -median problems under discrete Fréchet in Euclidean space and an exact algorithm for the (k, ℓ) -center problem under discrete Fréchet in 2D that all run in polynomial time for fixed k, ℓ, ε . It would be interesting to see if these algorithms can be adapted to the DTW or continuous Fréchet settings. Our approximation algorithms rely on the fact that good approximations have small distance to some optimal solution and that we can search a bounded space (the set of balls surrounding the vertices) for better approximations. The first property does not hold for DTW, since it is non-metric and the second property does not hold for continuous Fréchet, since the vertices of a curve with small continuous Fréchet distance do not have to be near the vertices of the other curve. The latter property is also crucial for the exact algorithm.

References

- 1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.
- 2 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal of Computing*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.
- 3 Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proc. 8th Latin American Conference on Theoretical Informatics*, pages 630–641, 2008.
- 4 Markus Brill, Till Fluschnik, Vincent Froese, Brijnesh Jain, Rolf Niedermeier, and David Schultz. Exact mean computation in dynamic time warping spaces. *Data Mining and Knowledge Discovery*, 33(1):252–291, 2019. doi:10.1007/s10618-018-0604-8.
- 5 Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, and Maarten Löffler. Approximating (k, ℓ) -center clustering for curves. *CoRR*, abs/1805.01547, 2018. arXiv:1805.01547.
- 6 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 ACM-SIAM Symposium on Discrete Algorithms*, pages 2922–2938, 2019.
- 7 Kevin Buchin, Anne Driemel, Natasja van de L’Isle, and André Nusser. kcluster: Center-based clustering of trajectories. In *Proc of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 496–499, 2019.
- 8 Laurent Bulteau, Vincent Froese, and Rolf Niedermeier. Tight hardness results for consensus problems on circular strings and time series. *arXiv preprint arXiv:1804.02854*, 2018. URL: <http://arxiv.org/abs/1804.02854>.
- 9 Bernard Marie Chazelle and Der-Tsai Lee. On a circle placement problem. *Computing*, 36(1-2):1–16, 1986.
- 10 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.
- 11 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.
- 12 Anne Driemel, Amer Krivošija, and Christian Sohler. Clustering time series under the Fréchet distance. In *Proc. 27th ACM-SIAM Symposium on Discrete Algorithms*, pages 766–785, 2016.
- 13 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.
- 14 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.
- 15 Kaspar Fischer, Bernd Gärtner, and Martin Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *Proc. 11th Annual European Symposium on Algorithms*, pages 630–641, 2003. doi:10.1007/978-3-540-39658-1_57.
- 16 Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: The dtw package. *Journal of Statistical Software, Articles*, 31(7):1–24, 2009. doi:10.18637/jss.v031.i07.
- 17 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.
- 18 Lalit Gupta, Dennis L Molfese, Ravi Tammana, and Panagiotis G Simos. Nonlinear alignment and averaging for estimating the evoked potential. *IEEE Transactions on Biomedical Engineering*, 43(4):348–356, 1996.
- 19 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proc. 34th ACM Symposium on Theory of Computing*, pages 731–740, 2002. doi:10.1145/509907.510012.

- 20 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM Journal of Computing*, 45(2):530–547, 2016. doi:10.1137/130938645.
- 21 Nimrod Megiddo. Linear-time algorithms for linear programming in \mathbb{R}^3 and related problems. *SIAM Journal of Computing*, 12(4):759–776, 1983. doi:10.1137/0212052.
- 22 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal of Computing*, 13(1):182–196, 1984. doi:10.1137/0213014.
- 23 François Petitjean and Pierre Gançarski. Summarizing a set of time series by averaging: From Steiner sequence to compact multiple alignment. *Theoretical Computer Science*, 414(1):76–91, 2012. doi:10.1016/j.tcs.2011.09.029.
- 24 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.
- 25 Kari-Jouko Rähö and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187–198, 1981. doi:10.1016/0304-3975(81)90075-X.
- 26 Alexis Sardá-Espinosa. Comparing time-series clustering algorithms in R using the dtwclust package. *R package vignette*, 12:41, 2017.
- 27 Alexis Sardá-Espinosa. Time-Series Clustering in R Using the dtwclust Package. *The R Journal*, 11(1):22–43, 2019. doi:10.32614/RJ-2019-023.
- 28 Nathan Schaar, Vincent Froese, and Rolf Niedermeier. Faster binary mean computation under dynamic time warping, 2020. arXiv:2002.01178.
- 29 Martijn Struijs. Curve clustering: hardness and algorithms. Msc thesis, Eindhoven University of Technology, 2018. URL: https://research.tue.nl/files/125547043/thesis_Martijn_Struijs_IAM_311.pdf.

A

 Appendix

A.1 Proof of Lemma 3

► **Lemma.** *If $(G \cup R_{i,j}, r)$ is a true instance of the average curve problem for discrete and continuous Fréchet, then (S, i, j) is a true instance of FCCS.*

Proof. We will show the proof for the continuous Fréchet distance. Since the continuous Fréchet distance is a lower bound of the discrete version, this proves the discrete case as well.

Since $(G \cup R_{i,j}, r)$ is a true instance of the average curve problem for continuous Fréchet, there exists a curve c^* such that $\sum_{g \in G \cup R_{i,j}} d_F(c^*, g) \leq r = |S| + 2$. We start by deriving bounds for the distance between c^* and the individual curves in $G \cup R_{i,j}$.

▷ **Claim.** $d_F(\gamma(s), \gamma(s')) \geq 2$ for all $s, s' \in S$ such that $s \neq s'$.

Proof. If a letter vertex p on $\gamma(s)$ is matched with a point p' that does not lie on a peak of the same letter in $\gamma(s')$, then $|p - p'| \geq 2$ and so $d_F(\gamma(s), \gamma(s')) \geq 2$. By symmetry, the same holds if we exchange s and s' .

Otherwise, each letter vertex can be matched only with points on a peak of the same letter. Let k be the first index such that $s[k] \neq s'[k]$. Then, the k -th letter vertex of $\gamma(s)$ cannot be matched to any point on the k -th peak of $\gamma(s')$ and must be matched to a point on another peak; the same holds with s and s' exchanged. It is not possible that on both curves the k -th letter vertex is matched with a peak of index larger than k , since the matching is monotone. So, one of the curves has its k -th letter vertex matched with a point on a peak of index smaller than k , we assume w.l.o.g. that this curve is s .

By monotonicity, the first k letter vertices of s are matched to the first $k - 1$ peaks of s' , so there are two letter vertices on s that are both matched with a point on the same peak

19:18 On the Hardness of Computing an Average Curve

on s' . The interval between those two points on this peak on s' must be matched with the interval between the letter vertices on s , so all points in the buffer gadget between the letter vertices on s are matched to some point on the peak on s' . But then there is either a point on an A-peak matched to g_b or a point on a B-peak matched to g_a , which in both cases has distance at least 2, so $d_F(\gamma(s), \gamma(s')) \geq 2$. \triangleleft

\triangleright **Claim.** $d_F(c^*, A_i) + d_F(c^*, B_j) \leq 2$

Proof. Using the previous claim and the triangle inequality, we have

$$d_F(c^*, \gamma(s_k)) + d_F(c^*, \gamma(s_{k+1})) \geq d_F(\gamma(s_k), \gamma(s_{k+1})) \geq 2$$

for all $k \in \{1, \dots, m-1\}$ and $d_F(c, \gamma(s_m)) + d_F(c, \gamma(s_1)) \geq 2$. The summation of these m inequalities has each s_k exactly twice on the lefthand side, so $\sum_{k=1}^m 2d_F(c^*, \gamma(s_k)) \geq 2m$, hence $\sum_{k=1}^m d_F(c^*, \gamma(s_k)) \geq m = |S|$. So, $d_F(c^*, A_i) + d_F(c^*, B_j) \leq r - \sum_{k=1}^m d_F(c^*, \gamma(s_k)) \leq 2$. \triangleleft

\triangleright **Claim.** $d_F(c^*, A_i) \geq 1$ and $d_F(c^*, B_j) \geq 1$.

Proof. Suppose $d_F(c^*, A_i) < 1$. Then, all points p on c^* are matched to some point in $[-3, 1]$ with distance < 1 , which means $|p - g_B| > 1$. We can assume that each string in S contains at least one B character (if there is a string s with only A characters, any supersequence with i A -characters is a supersequence of s when $|s| \leq i$ and none when $|s| > i$, so we can remove such trivial strings from the instance and check if the instance is trivially false). Therefore, $d_F(c^*, \gamma(s)) > 1$ for any $s \in S$.

Since $|g_a - g_b| = 2$, we have $d_F(A_i, B_j) \geq 2$, so $d_F(c^*, A_i) + d_F(c^*, B_j) \geq d_F(A_i, B_j) \geq 2$. But then $r \geq \sum_{g \in G \cup R_{i,j}} d_F(c^*, g) > |S| + 2 = r$, a contradiction, so $d_F(c^*, A_i) \geq 1$. The proof of $d_F(c^*, B_j) \geq 1$ is analogous. \triangleleft

\triangleright **Claim.** $d_F(c^*, g) = 1$ for all $g \in G \cup R_{i,j}$.

Proof. The last two claims together imply $d_F(c^*, A_i) = d_F(c^*, B_j) = 1$. This means that for each point p on c^* , $|p| \leq 2$ (otherwise, p has distance > 1 to all points on A_i or all points on B_j), so $d_F(c^*, \gamma(s)) \geq 1$ for all $s \in S$, since we can assume s contains at least one A and B character. Therefore, $d_F(\gamma(s), c^*) \leq r - d_F(A_i, c^*) - d_F(B_j, c^*) - \sum_{s' \in S \setminus \{s\}} d_F(\gamma(s'), c^*) \leq |S| - (|S| - 1) = 1$ for all $s \in S$. \triangleleft

Now we have shown that any center curve that achieves a cost of $|S| + 2$ for the constructed k -median instance needs to have Fréchet distance equal to 1 to all curves in this instance. It remains to show that such a center curve encodes a solution to the initial FCCS instance. Note that such a center curve is also a solution to the 1-center problem for this set of curves. We can now apply the proof of Lemma 33 from [6, 5], where the same gadgets were used in the reduction to the 1-center problem. \blacktriangleleft

A.2 Proof of Lemma 6

\blacktriangleright **Lemma.** *If $(G \cup R_{i,j}, r)$ is a true instance of (p, q) -DTW average curve, then (S, i, j) is a true instance of FCCS.*

Proof. If $(G \cup R_{i,j}, r)$ is a true instance of (p, q) -DTW average curve, then there exists a curve c^* such that $\sum_{g \in G \cup R_{i,j}} \text{DTW}_p^q(c^*, g) \leq r$. Take a curve $g \in G \cup R_{i,j}$. First note that there is at least one signal vertex in c^* matched to each letter gadget in g : otherwise, matching all β vertices in the gadget costs at least $\varepsilon^q \cdot \beta = \varepsilon^q \cdot (r/\varepsilon^q + 1) > r$, which contradicts the choice of

c^* . Similarly, each signal vertex is matched to at most one letter gadget in g , since otherwise it would have to match a g_0^β subcurve in between the letter gadgets, which would have a cost of at least $(1 - \varepsilon)^q \cdot \beta > \varepsilon^q \cdot \beta > r$. This means that the sequence of letter gadgets in $\gamma(s)$ is a subsequence of the sequence of signal vertices in c^* . So, if we construct s' from the sequence of signal vertices in c^* by mapping A-signal vertices to A characters and B-signal vertices to B characters, we have that s' is a supersequence of S . What remains to be proven is that $\#_A(s') = i$ and $\#_B(s') = j$, i.e. there are exactly i A-signal vertices and j B-signal vertices.

First, note that the sequence of A letter gadgets in A_i is a subsequence of the sequence of signal vertices in c^* (using the same argument as above), so there are at least i A-signal vertices. Analogously, there are at least j B-signal vertices. Now if we can show that there are at most $i + j$ signal vertices, then we are done.

Observe that there is at least one buffer vertex within a distance ε to g_0 in between signal vertices that are matched to letter gadget in A_i or B_j , as such a vertex must cover a g_0^β subcurve between the letter gadgets. We call signal vertices that are matched to the same letter gadget in either A_i or B_j a group. (Note that by definition, a signal vertex cannot be matched to letter gadgets in both A_i and B_j) This means that there are at least i groups of A-signal vertices and at least j groups of B-signal vertices.

When matching c^* and $\gamma(s)$ for some $s \in S$, we can only match at most $|s|$ groups of signal vertices to a g_a or g_b vertex in a letter gadget in $\gamma(s)$. So, for the at least $i + j - |s|$ remaining groups of signal vertices, we can either match them to a g_0 vertex in $\gamma(s)$, or to a corresponding g_a or g_b vertex. In the latter case, the signal vertex is matched to the same g_a^β or g_b^β subcurve in $\gamma(s)$ as another signal vertex in a different group. This means that the buffer vertex that separates the two signal vertices is matched to a g_a or g_b vertex in the letter gadget. So in all cases, we match two vertices at distance at least $1 - \varepsilon$. Since we do this for at least $i + j - |s|$ vertices, $\text{DTW}_p(c^*, \gamma(s)) \geq (1 - \varepsilon)(i + j - |s|)^{1/p}$.

Now, we have


$$\begin{aligned} \alpha(\text{DTW}_p^q(c^*, A_i) + \text{DTW}_p^q(c^*, B_j)) &\leq r - \sum_{s \in S} \text{DTW}_p^q(c^*, \gamma(s)) \\ &\leq r - \sum_{s \in S} (1 - \varepsilon)^q (i + j - |s|)^{q/p} \\ &= \alpha(i^{q/p} + j^{q/p}) \\ &\quad + \sum_{s \in S} (1 - (1 - \varepsilon)^q) (i + j - |s|)^{q/p} \\ &\leq \alpha(i^{q/p} + j^{q/p}) + (1 - (1 - \varepsilon)^q) |S| (i + j)^{q/p}, \end{aligned}$$

so that $\text{DTW}_p^q(c^*, A_i) + \text{DTW}_p^q(c^*, B_j) \leq i^{q/p} + j^{q/p} + (1 - (1 - \varepsilon)^q) (i + j)^{q/p} < i^{q/p} + j^{q/p} + \frac{1}{2} \min_{x \in \{i, j\}} (x + 1)^{q/p} - x^{q/p}$. This means that there are at most $i + j$ signal vertices: suppose there are at least $i + 1$ A-signal vertices, then $\text{DTW}_p^q(c^*, A_i) + \text{DTW}_p^q(c^*, B_j) \geq (1 - \varepsilon)^q ((i + 1)^{q/p} + j^{q/p}) \geq i^{q/p} + j^{q/p} + ((i + 1)^{q/p} - i^{q/p})/2$, a contradiction. Analogously, at least $j + 1$ B-signal vertices lead to a contradiction. \blacktriangleleft

Sparse Regression via Range Counting

Jean Cardinal 

Université libre de Bruxelles (ULB), Brussels, Belgium
jcardin@ulb.ac.be

Aurélien Ooms 

BARC, University of Copenhagen, Denmark
aurelien.ooms@di.ku.dk

Abstract

The sparse regression problem, also known as best subset selection problem, can be cast as follows: Given a set S of n points in \mathbb{R}^d , a point $y \in \mathbb{R}^d$, and an integer $2 \leq k \leq d$, find an affine combination of at most k points of S that is nearest to y . We describe a $O(n^{k-1} \log^{d-k+2} n)$ -time randomized $(1 + \varepsilon)$ -approximation algorithm for this problem with d and ε constant. This is the first algorithm for this problem running in time $o(n^k)$. Its running time is similar to the query time of a data structure recently proposed by Har-Peled, Indyk, and Mahabadi (ICALP'18), while not requiring any preprocessing. Up to polylogarithmic factors, it matches a conditional lower bound relying on a conjecture about affine degeneracy testing. In the special case where $k = d = O(1)$, we provide a simple $O_\delta(n^{d-1+\delta})$ -time deterministic exact algorithm, for any $\delta > 0$. Finally, we show how to adapt the approximation algorithm for the sparse linear regression and sparse convex regression problems with the same running time, up to polylogarithmic factors.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Computational geometry; Information systems \rightarrow Nearest-neighbor search

Keywords and phrases Sparse Linear Regression, Orthogonal Range Searching, Affine Degeneracy Testing, Nearest Neighbors, Hyperplane Arrangements

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.20

Related Version A full version of this paper is available at <https://arxiv.org/abs/1908.00351>.

Funding *Jean Cardinal*: Supported by the Fonds de la Recherche Scientifique-FNRS under CDR Grant J.0146.18.

Aurélien Ooms: Supported by the VILLUM Foundation grant 16582. Part of this research was accomplished while the author was a PhD student at ULB under FRIA Grant 5203818F (FNRS).

Acknowledgements The authors wish to thank the reviewers of earlier versions of this manuscript, who provided useful comments, as well as John Iacono and Stefan Langerman for insightful discussions.

1 Introduction

Searching for a point in a set that is the closest to a given query point is certainly among the most fundamental problems in computational geometry. It motivated the study of crucial concepts such as multidimensional search data structures, Voronoi diagrams, dimensionality reduction, and has immediate applications in the fields of databases and machine learning. A natural generalization of this problem is to search not only for a single nearest neighbor, but rather for the nearest *em combination* of a bounded number of points. More precisely, given an integer k and a query point y , we may wish to find an affine combination of k points of the set that is the nearest to y , among all possible such combinations. This problem has a natural interpretation in terms of sparse approximate solutions to linear systems, and is known as the *sparse regression*, or *sparse approximation* problem in the statistics and machine learning literature. Sparsity is defined here in terms of the ℓ_0 pseudonorm $\|\cdot\|_0$, the number of nonzero components. The *sparse affine regression* problem can be cast as follows:



© Jean Cardinal and Aurélien Ooms;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 20; pp. 20:1–20:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

20:2 Sparse Regression via Range Counting

► **Problem 1** (Sparse affine regression). *Given a matrix $A \in \mathbb{R}^{d \times n}$, a vector $y \in \mathbb{R}^d$, and an integer $2 \leq k \leq d$, find $x \in \mathbb{R}^n$ minimizing $\|Ax - y\|_2$, and such that $\|x\|_0 \leq k$, and $\sum_{i=1}^n x_i = 1$.*

By interpreting the columns of A as a set of n points in \mathbb{R}^d , the problem can be reformulated in geometric terms as the *nearest induced flat* problem.

► **Problem 2** (Nearest induced flat). *Given a set S of n points in \mathbb{R}^d , an additional point $y \in \mathbb{R}^d$, and an integer k such that $2 \leq k \leq d$, find k points of S such that the distance from y to their affine hull is the smallest.*

Here the distance from a point to a flat is the distance to the closest point on the flat. Note that if we allow $k = 1$ in the definition above, we have the nearest neighbor problem as a special case. We consider the setting in which the dimension d of the ambient space as well as the number k of points in the sought combination are constant, and study the asymptotic complexity of the problem with respect to n . As observed recently by Har-Peled, Indyk, and Mahabadi [25], the problem is closely related to the classical *affine degeneracy testing* problem, defined as follows.

► **Problem 3** (Affine degeneracy testing). *Given a set S of n points in \mathbb{R}^d , decide whether there exist $d + 1$ distinct points of S lying on an affine hyperplane.*

The latter can be cast as deciding whether a point set is in so-called *general position*, as is often assumed in computational geometry problems. In the special case $d = 2$, the problem is known to be 3SUM-hard [24, 9]. In general, it is not known whether it can be solved in time $O(n^{d-\delta})$ for some positive δ [21, 3], even for randomized algorithms. Supposing it cannot, we directly obtain a conditional lower bound on the complexity of the nearest induced flat problem. This holds even for approximation algorithms, which return an induced flat whose distance is within some bounded factor of the distance of the actual nearest flat.

► **Lemma 1** (Har-Peled, Indyk, and Mahabadi [25]). *If the nearest induced flat problem can be approximated within any multiplicative factor in time $O(n^{k-1-\delta})$ for some positive δ , then affine degeneracy testing can be solved in time $O(n^{d-\delta})$.*

Proof. Suppose we have an approximation algorithm for the nearest induced flat problem. Then given an instance of affine degeneracy testing, we can go through every point $y \in S$ and run this algorithm on an instance composed of the set $S \setminus \{y\}$, the point y , and $k = d$. The answer to the degeneracy testing instance is positive if and only if for at least one of these instances, the distance to the approximate nearest flat is zero. The running time is $O(n^{d-\delta})$. ◀

Motivations and previous works

Sparse regression is a cornerstone computational task in statistics and machine learning, and comes in a number of flavors. It is also referred to as *best subset selection* or, more generally, as *feature selection* problems [31, 11]. In practice, it is often useful to allow for the sparsity constraint by including a penalty term in the objective function, hence writing the problem in a Lagrangian form. If the ℓ_1 norm is used instead of the ℓ_0 norm, this method is known as the LASSO method [32], to which a tremendous amount of research has been dedicated in the past twenty years. In the celebrated k -SVD algorithm for sparse dictionaries design [2], the sparse coding stage consists of a number of sparse regression steps. In this context, they are typically carried out using greedy methods such as the matching pursuit algorithm [29]. Efficient sparse regression is also at the heart of *compressed sensing* techniques [13, 18].

Aiming at an exhaustive survey of the variants and applications of sparse regression is futile; instead, we refer to Hastie, Tibshirani, and Friedman [26] (Chapter 3), Miller [30], and references therein. We also point to Bertsimas, Pauphilet, and Van Parys [12] for a recent survey on practical aspects of sparse regression methods.

The computational complexity of sparse regression problems is also well-studied [31, 17, 23, 22]. In general, when a solution x is sought that minimizes the number of nonzero components while being at bounded distance from y , the problem is known to be NP-hard [31]. However, the complexity of the sparse regression problem when the sparsity constraint k is taken as a fixed parameter has not been thoroughly characterized. In particular, no algorithm with running time $o(n^k)$ is known.

Recently, Har-Peled, Indyk, and Mahabadi [25] showed how to use approximate nearest neighbor data structures for finding approximate solutions to the sparse affine regression problem. They mostly consider the *online* version of the problem, in which we allow some preprocessing time, given the input point set S , to construct a data structure, which is then used to answer queries with input y . They also restrict to *approximate* solutions, in the sense that the returned solution has distance at most $(1 + \varepsilon)$ times larger than the true nearest neighbor distance for any fixed constant ε . They show that if there exists a $(1 + \varepsilon)$ -approximate nearest neighbor data structure with preprocessing time $S(n, d, \varepsilon)$ and query time $Q(n, d, \varepsilon)$, then we can preprocess the set S in time $n^{k-1}S(n, d, \varepsilon)$ and answer regression queries in time $n^{k-1}Q(n, d, \varepsilon)$. Plugging in state of the art results on approximate nearest neighbor searching in fixed dimension [8], we obtain a preprocessing time of $O(n^k \log n)$ with query time $O(n^{k-1} \log n)$ for fixed constants d and ε .

They also consider the *sparse convex regression* problem, in which the coefficients of the combination are not only required to sum to one, but must also be nonnegative. In geometric terms, this is equivalent to searching for the *nearest induced simplex*. They describe a data structure for the sparse convex regression problem having the same performance as in the affine case, up to a $O(\log^k n)$ factor. For $k = 2$, they also give a $(2 + \varepsilon)$ -approximation subquadratic-time offline algorithm. When $d = O(1)$, the running time of this algorithm can be made close to linear.

A closely related problem is that of searching for the nearest flat in a set [27, 10, 28]. This was also studied recently by Agarwal, Rubin, and Sharir [1], who resort to polyhedral approximations of the Euclidean distance to design data structures for finding an approximate nearest flat in a set. They prove that given a collection of n $(k - 1)$ -dimensional flats in \mathbb{R}^d , they can construct a data structure in time $O(n^k \text{polylog}(n))$ time and space that can be used to answer $(1 + \varepsilon)$ -approximate nearest flat queries in time $O(\text{polylog}(n))$. They also consider the achievable space-time tradeoffs. Clearly, such a data structure can be used for online sparse affine regression: We build the structure with all possible $\binom{n}{k}$ flats induced by the points of S . This solution has a very large space requirement and does not help in the offline version stated as Problem 2.

In this paper, we give an efficient algorithm for Problem 2, and bridge the gap between the trivial upper bound of $O(n^k)$ and the lower bound given by the affine degeneracy testing problem, without requiring any preprocessing.

Our results

Nearest induced line, flat, or hyperplane

We prove that the nearest induced flat problem (Problem 2), can be solved within a $(1 + \varepsilon)$ approximation factor for constant d and ε in time $O(n^{k-1} \log^{d-k+2} n)$, which matches the conditional lower bound on affine degeneracy testing, up to polylogarithmic factors. Har-

20:4 Sparse Regression via Range Counting

Peled, Indyk, and Mahabadi [25] gave a data structure to preprocess a set of data points to allow solving the nearest induced flat problem on this set for any query point. Their data structure requires $\tilde{O}(n^k)$ preprocessing and $\tilde{O}(n^{k-1})$ query time. We propose an algorithm that gets rid of the preprocessing for single queries: the overall running time of our algorithm is equal to the query time of their data structure, up to polylogarithmic factors. To the best of our knowledge, this is a near-linear improvement on all previous methods for this special case.

The two main tools that are used in our algorithms are on the one hand the approximation of the Euclidean distance by a polyhedral distance, as is done in Agarwal, Rubin, and Sharir [1], and on the other hand a reduction of the decision version of the problem to orthogonal range queries. Note that orthogonal range searching data structures are also used in [25], albeit in a significantly distinct fashion.

In §2, as warm-up, we focus on the special case of Problem 2 in which $d = 3$ and $k = 2$.

► **Problem 4** (Nearest induced line in \mathbb{R}^3). *Given a set S of n points in \mathbb{R}^3 , and an additional point y , find two points $a, b \in S$ such that the distance from y to the line going through a and b is the smallest.*

Our algorithm for this special case already uses all the tools that are subsequently generalized for arbitrary values of k and d . The general algorithm for the nearest induced flat problem is described in §3.

In §4, we consider the special case of Problem 2 in which $k = d$, which can be cast as the *nearest induced hyperplane* problem.

► **Problem 5** (Nearest induced hyperplane). *Given a set S of n points in \mathbb{R}^d , and an additional point y , find d points of S such that the distance from y to the affine hyperplane spanned by the d points is the smallest.*

For this case, we design an *exact* algorithm with running time $O(n^{d-1+\delta})$, for any $\delta > 0$. The solution solely relies on classical computational geometry tools, namely point-hyperplane duality and cuttings [16, 15].

Our algorithms can be adapted to perform *sparse linear regression*, instead of sparse affine regression. In the former, we drop the condition that the sum of the coefficients must be equal to one. This is equivalent to the nearest *linear* induced k -flat problem. It can be solved in the same time as in the affine case. To see this, realize that the problem is similar to the nearest induced flat problem where the first vertex is always the origin. The obtained complexity is the same as the one for the nearest induced flat problem.

Nearest induced simplex

Adapting our algorithm to sparse *convex* regression, which differs from sparse affine regression by requiring x to be positive, is a bit more involved.

Har-Peled, Indyk, and Mahabadi [25] augment their data structure for the nearest induced flat with orthogonal range searching data structures in $(k + 1)$ -dimensional space to solve this problem with an extra $O(\log^k n)$ factor in both the preprocessing and query time. We show we can perform a similar modification.

The sparse convex regression problem can be cast as the problem of finding the nearest simplex induced by k points of S .

► **Problem 6** (Nearest induced simplex). *Given a set S of n points in \mathbb{R}^d , an additional point y , and an integer k such that $2 \leq k \leq d$, find k points of S such that the distance from y to their convex hull is the smallest.*

■ **Table 1** Results. For the approximation algorithms, the dependency on ε in the running time is of the order of $\varepsilon^{(1-d)/2}$.

Problem	Details	Approximation	Running Time
Problem 4: Nearest induced line in \mathbb{R}^3	§2	$1 + \varepsilon$	$O_\varepsilon(n \log^3 n)$
Problem 2: Nearest induced flat	§3	$1 + \varepsilon$	$O_{d,\varepsilon}(n^{k-1} \log^{d-k+2} n)$
Problem 5: Nearest induced hyperplane	§4	1	$O_{d,\delta}(n^{d-1+\delta}), \forall \delta > 0$
Problem 6: Nearest induced simplex	§5	$1 + \varepsilon$	$O_{d,\varepsilon}(n^{k-1} \log^d n)$

We prove that this problem can also be approximated within a $(1 + \varepsilon)$ approximation factor for constant d and ε in time $O(n^{k-1} \log^d n)$, hence with an extra $O(\log^{k-2} n)$ factor in the running time compared to the affine case. This is described in §5.

Our results and the corresponding sections are summarized in Table 1.

2 A $(1 + \varepsilon)$ -approximation algorithm for the nearest induced line problem in \mathbb{R}^3

We first consider the nearest induced line problem (Problem 4). We describe a near-linear time algorithm that returns a $(1 + \varepsilon)$ -approximation to the nearest induced line in \mathbb{R}^3 , that is, a line at distance at most $(1 + \varepsilon)$ times larger than the distance to the nearest line.

► **Theorem 2.** *For any constant $\varepsilon > 0$, there is a randomized $(1 + \varepsilon)$ -approximation algorithm for the nearest induced line problem in \mathbb{R}^3 running in time $O_\varepsilon(n \log^3 n)$ with high probability.*

The sketch of our algorithm is as follows: First, reduce the problem of minimizing the Euclidean distance to that of minimizing the polyhedral distance for some well-chosen polyhedron depending on ε . Second, reduce the problem of minimizing the polyhedral distance to that of edge-shooting. Third, reduce the problem of edge-shooting to that of deciding whether an edge shot at a certain distance would hit any induced line through some sort of binary search. Fourth, efficiently solve this decision problem using orthogonal range counting data structures.

$(1 + \varepsilon)$ -approximation via polyhedral distances

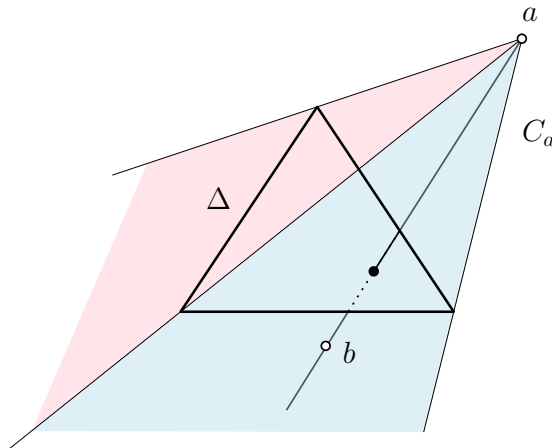
The *polyhedral distance* $d_Q(y, v)$ between two points y and v with respect to a polyhedron Q centered on the origin is the smallest λ such that the dilation λQ of Q centered on y contains v , hence such that $v \in y + \lambda Q$. Our proof uses the following result, of which a weaker variant due to Dudley [19] is a major ingredient in the design of the data structure described by Agarwal, Rubin, and Sharir [1].

► **Lemma 3** (Arya, Arya, da Fonseca, Mount [4]). *For any positive integer d and positive real ε , there exists a d -dimensional polyhedron Q with $O(1/\varepsilon^{(d-1)/2})$ faces such that for every $y, v \in \mathbb{R}^d$:*

$$\|y - v\|_2 \leq d_Q(y, v) \leq (1 + \varepsilon) \cdot \|y - v\|_2.$$

This bound is asymptotically optimal. See [5, 7, 6] for more details.

Next, we reduce Problem 4 to a counting problem in two steps.



■ **Figure 1** The cone C_a .

Edge-shooting

We use Lemma 3 for $d = 3$. We give an exact algorithm for computing the nearest induced line with respect to a polyhedral distance d_Q , where Q is defined from ε as in Lemma 3. Given a polyhedron Q , one can turn it into a simplicial polyhedron by triangulating it. Therefore, for constant values of ε , this reduces the problem to a constant number of instances of the *edge-shooting problem*, defined as follows: Given an edge e of Q , find the smallest value λ such that $y + \lambda e$ intersects a line through two points of S . We iterate this for all edges of Q , and pick the minimum value. This is exactly the polyhedral distance from y to its nearest induced line.

Binary search

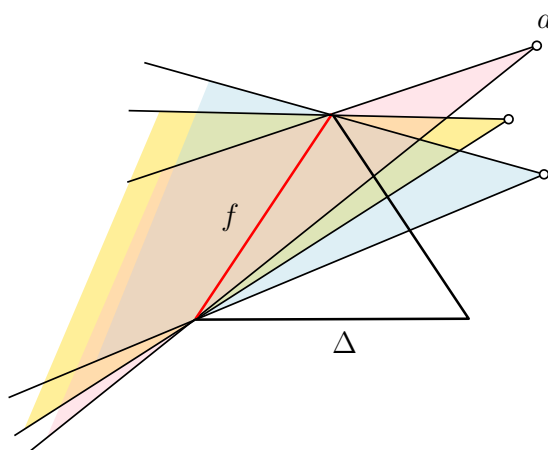
Using a randomized binary search procedure, we reduce the edge-shooting problem to a *counting problem*, defined as follows: given the triangle Δ defined as the convex hull of y and $y + \lambda e$, count how many pairs of points $a, b \in S$ are such that the line $\ell(a, b)$ through them intersects Δ . Suppose there exists a procedure for solving this problem. We can use this procedure to solve the edge-shooting problem efficiently as follows.

First initialize λ to some upper bound on the distance (for instance, initialize λ to the distance to the closest data point $p \in S$: $\lambda = \min_{p \in S} \|p - y\|_2$). Then count how many lines $\ell(a, b)$ intersect Δ , using the procedure. If there is only one, then return its (polyhedral) distance to y . Otherwise, pick one such line uniformly at random and compute the value λ' such that this line intersects $y + \lambda' e$. Then iterate the previous steps with $\lambda \leftarrow \lambda'$, unless $\lambda' = 0$ in which case we return 0. Since we picked the line at random, and since there are $O(n^2)$ such lines at the beginning of the search, the number of iterations of this binary search is $O(\log n)$ with high probability.

We therefore reduced the nearest induced line problem to $O(\varepsilon^{-1} \log n)$ instances of the counting problem.

Orthogonal range counting queries

Data structures for *orthogonal range counting queries* store a set of points in \mathbb{R}^g in such a way that the number of points in a given g -rectangle (cartesian product of g intervals) can be returned quickly. Known data structures for orthogonal range counting queries in \mathbb{R}^g



■ **Figure 2** The order of the points defined by the planes containing an edge f of Δ .

require $O(n \log^{g-1} n)$ preprocessing time and can answer queries in $O(\log^{g-1} n)$ time [34, 14]. Note that the actual coordinates of the points do not matter: We only need to know the order of their projections on each axis. We now show how to solve the counting problem using a data structure for orthogonal range queries in \mathbb{R}^3 .

Let us fix the triangle Δ and a point $a \in \mathbb{R}^3$, and consider the locus of points $b \in \mathbb{R}^3$ such that the line $\ell(a, b)$ intersects Δ . This is a double simplicial cone with apex a and whose boundary contains the boundary of Δ . This double cone is bounded by three planes, one for each edge of Δ . In fact, we will only consider one of the two cones, because $\ell(a, b)$ intersects Δ if and only if either b is contained in the cone of apex a , or a is contained in the cone of apex b . Let us call C_a the cone of apex a . This is illustrated on Figure 1.

Let us consider one edge f of Δ and all the planes containing f . These planes induce a circular order on the points of S , which is the order in which they are met by a plane rotating around the supporting line of f . This is illustrated on Figure 2. Now let us denote by H_f the plane containing a and f and by H_f^+ the halfspace bounded by H_f and containing Δ . The set of points of S contained in H_f^+ is an interval in the circular order mentioned above. Hence the set of points contained in C_a is the intersection of three intervals in the three circular orders defined by the three edges of Δ .

Proof of Theorem 2. Let Q be some polyhedron in \mathbb{R}^3 , $\lambda \in \mathbb{R}$, $S \subset \mathbb{R}^3$, $y \in \mathbb{R}^3$, and e an edge of Q . We use an orthogonal range counting data structure for storing the points of S with coordinates corresponding to their ranks in each of the three permutations induced by the three edges of $\Delta = \text{conv}(\{y, y + \lambda e\})$. We get those rank-coordinates by sorting S three times, once for each induced permutation, in time $O(n \log n)$, then construct the orthogonal range counting data structure with those coordinates in time $O(n \log^2 n)$. Then for each of the n points $a \in S$, we count the number of points b in the cone C_a by querying the data structure in $O(\log^2 n)$ time. Hence overall, the counting problem is solved in time $O(n \log^2 n)$. Note that the circularity of the order can be easily handled by doubling every point.

This can be combined with the previous reductions provided we can choose a line intersecting Δ uniformly at random within that time bound. This is achieved by first choosing a with probability proportional to the number of points b such that $\ell(a, b) \cap \Delta \neq \emptyset$. Then we can pick a point b uniformly at random in this set in linear time.

Combining with the previous reductions, we obtain an approximation algorithm running in time $O_\epsilon(n \log^3 n)$ for the nearest induced line problem in \mathbb{R}^3 . ◀

3 A $(1 + \varepsilon)$ -approximation algorithm for the nearest induced flat problem

This section is dedicated to proving our main result in full generality. We provide an efficient approximation algorithm for the nearest induced flat problem (Problem 2).

We use the following notations: $\text{aff}(X)$ denotes the affine hull of the set X and $\text{conv}(X)$ denotes its convex hull. The set $\{1, 2, \dots, n\}$ is denoted by $[n]$.

► **Theorem 4.** *For any constant positive real $\varepsilon > 0$ and constant positive integers d and k , there is a randomized $(1 + \varepsilon)$ -approximation algorithm for the nearest induced flat problem in \mathbb{R}^d running in time $O_\varepsilon(n^{k-1} \log^{d-k+2} n)$ with high probability.*

Proof. The algorithm is a generalization of the one in the previous section, in which the point a is replaced by a set composed of $k - 1$ points a_1, a_2, \dots, a_{k-1} , and the edge e is now a (simplicial) $(d - k)$ -face of Q . Given a $k - 1$ -tuple of points a_1, a_2, \dots, a_{k-1} , we characterize the locus of points a_k such that the affine hull of the points a_1, a_2, \dots, a_k intersects the convex hull of y and $y + \lambda e$. These hyperplanes are again such that counting all such points can be done using orthogonal range queries. More precisely, we perform the following steps.

$(1 + \varepsilon)$ -approximation and binary search

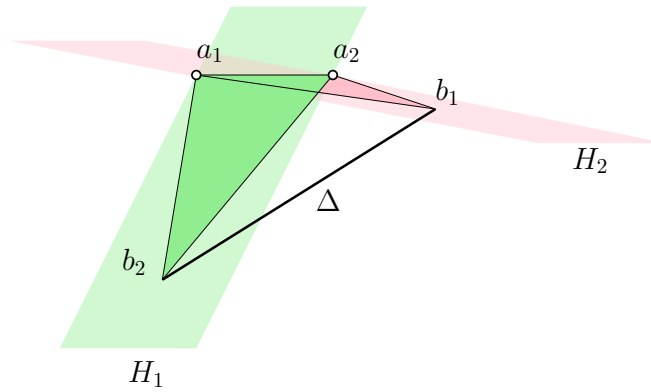
From Lemma 3, there exists a polyhedron with $O(1/\varepsilon^{(d-1)/2})$ faces such that the induced polyhedral distance $d_Q(\cdot, \cdot)$ is a $(1 + \varepsilon)$ -approximation of the Euclidean distance. We know that the distance d_Q from the point y to the nearest induced flat is attained at a point lying on a $(d - k)$ -face of $y + \lambda Q$. We can therefore perform the same procedure as in the previous case, except that we now shoot a $(d - k)$ -face e of Q , instead of an edge, in the same way as is done in Agarwal, Rubin, Sharir [1]. Δ still denotes the convex hull of y and $y + \lambda e$, which generalizes to a $(d - k + 1)$ -simplex. The binary search procedure generalizes easily: start with a large enough λ , if there is more than one flat $\text{aff}(\{a_1, a_2, \dots, a_k\})$ intersecting $\Delta = \text{conv}(\{y, y + \lambda e\})$, pick one such flat uniformly at random, and compute the value λ such that this flat intersects Δ . There are only $O(n^k)$ such flats at the beginning of the search, hence a search takes $O(\log n)$ steps with high probability. We can therefore reduce the problem to $O(\varepsilon^{(1-d)/2} \log n)$ instances of the following *counting problem*: given a $(d - k + 1)$ -simplex Δ , count the number of k -tuples of points $a_1, a_2, \dots, a_k \in S$ whose affine hull $\text{aff}(a_1, a_2, \dots, a_k)$ intersects Δ .

An intersection condition

We first make a simple observation that characterizes such k -tuples. Let A be a set of k points $\{a_1, a_2, \dots, a_k\}$, and let $B = \{b_1, b_2, \dots, b_{d-k+2}\}$ be the set of vertices of Δ . We assume without loss of generality that the points of A together with the vertices of Δ are in general position. We define $d - k + 2$ hyperplanes $H_i = \text{aff}(A \cup B \setminus \{b_i, a_k\})$, $i \in [d - k + 2]$. We then let H_i^+ be the halfspace supported by H_i that contains b_i , and H_i^- the halfspace that does not contain b_i .

► **Lemma 5.**

$$\text{aff}(A) \cap \Delta \neq \emptyset \iff a_k \in \left(\left(\bigcap_{i=1}^{d-k+2} H_i^+ \right) \cup \left(\bigcap_{i=1}^{d-k+2} H_i^- \right) \right).$$



■ **Figure 3** Illustration of Lemma 5 in the case $k = d = 3$. The plane through a_1, a_2, a_3 intersects the line segment Δ if and only if a_3 is located either above or below the two planes H_1, H_2 .

Proof. (\Rightarrow) Suppose that $a_k \notin (\bigcap_i H_i^+) \cup (\bigcap_i H_i^-)$. Hence there exists $i \in [d - k + 2]$ such that $a_k \in H_i^-$, and $j \in [d - k + 2]$ such that $a_k \in H_j^+$. We show that $\text{aff}(A) \cap \Delta = \emptyset$. Let us consider the intersection of the two halfspaces H_i^- and H_j^+ with the $(k - 1)$ -dimensional subspace $\text{aff}(A)$. In this subspace, both halfspaces have the points a_1, a_2, \dots, a_{k-1} on their boundary, and contain a_k . Hence it must be that $H_i^- \cap \text{aff}(A) = H_j^+ \cap \text{aff}(A)$. Therefore, every point $p \in \text{aff}(A)$ either lies in H_i^- , or in H_j^- . In both cases, it is separated from Δ by a hyperplane, and $p \notin \Delta$.

(\Leftarrow) Suppose that $\text{aff}(A) \cap \Delta = \emptyset$. We now show that there exists $i \in [d - k + 2]$ such that $a_k \in H_i^-$, and $j \in [d - k + 2]$ such that $a_k \in H_j^+$. Since both $\text{aff}(A)$ and Δ are convex sets, if $\text{aff}(A) \cap \Delta = \emptyset$ then there exists a hyperplane H containing $\text{aff}(A)$ and having Δ on one side. Since the points of A are affinely independent, H can be rotated to contain all points of A except a_k , and separate a_k from Δ . After this rotation, H has $d - (k - 1)$ degrees of freedom left, and can be further rotated to contain a whole $(d - k)$ -face of Δ , while still separating Δ from a_k . For some $i \in [d - k + 2]$, this is now the hyperplane H_i that separates some vertex b_i from a_k , and $a_k \in H_i^-$.

Similarly, the same hyperplane H can instead be rotated in order to contain all points of A except a_k , and have a_k and Δ this time on the same side. It can then be further rotated to contain a $(d - k)$ -face of Δ , while still having Δ and a_k on the same side. Now for some $j \in [d - k + 2]$, this is now the hyperplane H_j that has b_j and a_k on the same side, and $a_k \in H_j^+$. ◀

Note that for the case $k = 2$ and $d = 3$ the set $(\bigcap_i H_i^+) \cup (\bigcap_i H_i^-)$ is the double cone of apex a ; the lower part $(\bigcap_i H_i^+)$ is the cone C_a in Figure 1. The case where $k = 3$ and $d = 3$ is illustrated on Figure 3.

Reduction to orthogonal range queries

We now show that in perfect analogy with the previous section, we can solve the counting problem efficiently using an orthogonal range counting data structure.

Consider a vertex b_i of Δ and a $(k - 2)$ -subset T of points of S , denoted by $T = \{a_1, a_2, \dots, a_{k-2}\}$. Let us denote by f the facet of Δ that is induced by the vertices b_j such that $j \neq i$. Now consider the hyperplane containing f together with T , and one additional point p of S . These hyperplanes all contain $\text{aff}(f \cup T)$, which is a $(d - 2)$ -flat. Let us consider the unit normal vectors to these hyperplanes centered on some point contained in

this $(d - 2)$ -flat. These vectors lie in the orthogonal flat of dimension $d - (d - 2) = 2$, hence in a plane. Therefore, they induce a circular order on the points of S . Hence for a fixed set of $k - 2$ points of S and a fixed facet f of Δ , we can assign a rank to each other point of S . These will play the role of the coordinates of the points in the range counting data structure.

We now observe that counting the number of k -tuples whose affine hull intersects Δ amounts to orthogonal range counting with respect to these coordinates. Indeed, fix the first $(k - 2)$ -subset of points $T = \{a_1, a_2, \dots, a_{k-2}\}$, and compute the rank of each other point of S with respect to the circular order of the hyperplanes defined above, around each facet f of Δ . Now consider a $(k - 1)$ th point a_{k-1} . From Lemma 5, all points a_k contained in the range $(\bigcap_i H_i^+) \cup (\bigcap_i H_i^-)$ are such that $\text{aff}(a_1, a_2, \dots, a_k)$ intersects Δ . But this range is the union of two $(d - k + 2)$ -rectangles in the space of coordinates that we defined. The coordinates of these two $(d - k + 2)$ -rectangles are defined by the coordinates of a_{k-1} . We can therefore set up a new orthogonal range counting data structure for each $(k - 2)$ -subset T , and perform $2n$ queries in it, two for each additional point $a_{k-1} \in S$.

We can now outline our algorithm for solving the counting problem:

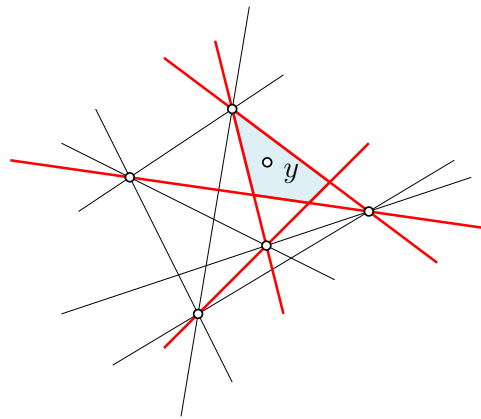
1. For each $(k - 2)$ -subset T of points a_1, a_2, \dots, a_{k-2} in $\binom{S}{k-2}$:
 - a. For each vertex b_i of Δ , compute the rank of each point of S with respect to the hyperplanes containing $f = \text{conv}(\{b_j : j \neq i\})$ and T .
 - b. Build a $(d - k + 2)$ -dimensional range counting data structure on S using these ranks as coordinates.
 - c. For each other point $a_{k-1} \in S$:
 - i. Perform two range counting queries using the rectangular ranges corresponding to $\bigcap_i H_i^+$ and $\bigcap_i H_i^-$, respectively.
 - d. Return the sum of the values returned by the range counting queries.

Note that there are a few additional technicalities which we have to take care of. First, the orders defined by the hyperplanes are circular, hence we are really performing range queries on a torus. This can be easily fixed, as mentioned previously, by doubling each point. Then we have to make sure to avoid double counting, since any permutation of the a_i in the enumeration of k -tuples yields the same set A , and hence, the same flat $\text{aff}(A)$. (Note that in §2 we avoided double counting by observing that only one of $a \in C_b$ and $b \in C_a$ can be true.) This only affects the counting problem and is not problematic if we consider *ordered* subsets T ; it causes each intersecting flat to be counted exactly $k!$ times.¹ The termination condition for the binary search can be changed to when the range count is $k!$ and the sampling method for finding a uniform random binary search pivot is unaffected since each candidate flat is represented an equal number of times.

As for the running time analysis, step 1b costs $O(n \log^{d-k+1} n)$, while step 1(c)i costs $O(\log^{d-k+1} n)$ and is repeated $n - k + 2$ times, hence costs $O(n \log^{d-k+1} n)$ overall as well [34, 14]. These are multiplied by the number of iterations of the main loop, yielding a complexity of $O(n^{k-1} \log^{d-k+1} n)$ for the counting procedure.

Finally, this counting procedure can be combined with the binary search procedure provided we can choose a flat intersecting Δ uniformly at random within that time bound. This is achieved by first choosing a set prefix $\{a_1, a_2, \dots, a_{k-1}\} \in \binom{S}{k-1}$ with probability proportional to the number of points $a_k \in S$ such that $\text{aff}(\{a_1, a_2, \dots, a_k\}) \cap \Delta \neq \emptyset$. Then we can pick a point a_k uniformly at random in this set in linear time. Multiplying by the number of edge-shooting problems we have to solve, the counting procedure is invoked $O(\varepsilon^{(1-d)/2} \log n)$ times, yielding the announced running time. ◀

¹ Enumerating each subset T exactly once as $(k - 2)$ -tuples in lexicographic order and only constructing the orthogonal range searching data structure on the points of S that come after a_{k-2} reduces this overcounting to 2 times per flat. In our case, this is unnecessary since k is constant.



■ **Figure 4** The candidate nearest hyperplanes.

4 An exact algorithm for the nearest induced hyperplane problem

In this section we consider the special case $k = d$, the nearest induced hyperplane problem (Problem 5). The previous result gives us a randomized $(1 + \varepsilon)$ -approximation algorithm running in time $O_\varepsilon(n^{d-1} \log^2 n)$ for this problem. We describe a simple deterministic $O(n^{d-1+\delta})$ -time exact algorithm using only standard tools from computational geometry.

► **Theorem 6.** *The nearest induced hyperplane problem can be solved in deterministic time $O(n^{d-1+\delta})$ for any $\delta > 0$.*

The first tool we need is point-hyperplane duality. Let \bar{H} be the hyperplane arrangement that is dual to S , in which each point of S is now a hyperplane. Note that every vertex of this arrangement is the dual of a hyperplane induced by d points of S .

Unfortunately, while some dualities preserve vertical distances, there does not exist a duality that preserves euclidean distances. To overcome this obstacle, we make a topological observation. Recall that the *zone* of a hyperplane h in an arrangement \bar{H} (not including h) is the union of the d -cells of \bar{H} intersected by h . Similarly, we define the *refined zone* of a hyperplane h in an arrangement \bar{H} (not including h) to be the union of the d -simplices of the bottom-vertex decomposition of \bar{H} intersected by h .

► **Lemma 7.** *Let \bar{H} be the hyperplane arrangement that is dual to S , and \bar{y} the hyperplane dual to the point y . The induced hyperplane that is nearest to y corresponds to a vertex of the refined zone of \bar{y} in the arrangement \bar{H} .*

Proof. Consider the arrangement of all $\binom{n}{k}$ hyperplanes induced by subsets of k points in S . Then clearly, the induced hyperplane nearest to y must be one of the hyperplanes bounding the cell of this arrangement that contains y (see Figure 4 for an illustration with $d = 2$). Consider a rectilinear motion of y towards this nearest hyperplane. In the dual arrangement \bar{H} , this corresponds to a continuous motion of the hyperplane \bar{y} that at some point hits a vertex of the arrangement. Because it is the first vertex that is hit, it must belong to a cell of the bottom vertex decomposition of \bar{H} that \bar{y} intersects, hence to the refined zone of \bar{y} . ◀

We refer to chapter 28 of the Handbook of Discrete and Computational Geometry [33] for background on hyperplane arrangements and their decompositions.

The second tool is an upper bound on the complexity of a zone in an arrangement [20]. The *complexity of a zone* is the sum of the complexities of its cells, and the complexity of a cell is the number of faces of the cell (vertices, edges, ...). The upper bound is as follows:

20:12 Sparse Regression via Range Counting

► **Theorem 8** (Zone Theorem [20]). *The complexity of a zone in an arrangement of n hyperplanes in \mathbb{R}^d is $O(n^{d-1})$.*

In particular, this result gives an upper bound of $O(n^{d-1})$ vertices for a given zone. Since the complexity of a refined zone is not more than the complexity of the corresponding zone, this bound also holds for the complexity of a given refined zone.

The third tool is Chazelle's efficient construction of cuttings [15]. A *cutting* of \mathbb{R}^d is a partition of \mathbb{R}^d into disjoint regions. Given a set of hyperplanes H in \mathbb{R}^d , a $\frac{1}{r}$ -*cutting* for H is a cutting of \mathbb{R}^d such that each region is intersected by no more than $\frac{|H|}{r}$ hyperplanes in H . In particular, we are interested in Chazelle's construction when r is constant. In that case, only a single step of his construction is necessary and yields regions that are the simplices of the bottom-vertex decomposition of some subset of H .

► **Theorem 9** (Chazelle [15, Theorem 3.3]). *Given a set H of n hyperplanes in \mathbb{R}^d , for any real constant parameter $r > 1$, we can construct a $\frac{1}{r}$ -cutting for those hyperplanes consisting of the $O(r^d)$ simplices of the bottom-vertex decomposition of some subset of H in $O(n)$ time.*

More details on cuttings can be found in chapters 40 and 44 of the Handbook [33].

► **Lemma 10.** *For any positive constant δ , given a hyperplane h and an arrangement of hyperplanes \bar{H} in \mathbb{R}^d , the vertices of the refined zone of h in \bar{H} can be computed in time $O(n^{d-1+\delta})$.*

Proof. Using Theorem 9 with some constant r , we construct, in linear time, a $\frac{1}{r}$ -cutting of the arrangement consisting of $O(r^d)$ simplicial cells whose vertices are vertices of \bar{H} . To find the vertices of the refined zone, we only need to look at those cells that are intersected by \bar{y} . If such a cell is not intersected by any hyperplane of \bar{H} then its vertices are part of the refined zone of \bar{y} . Otherwise, we recurse on the hyperplanes intersecting that cell. From Theorem 8, there are at most $O(r^{d-1})$ such cells. The overall running time for the construction is therefore:

$$T(n) \leq O(r^{d-1})T\left(\frac{n}{r}\right) + O(n).$$

For all constant $\delta > 0$, we can choose a sufficiently large constant r , such that $T(n) = O(n^{d-1+\delta})$, as claimed. ◀

Proof of Theorem 6. From Lemma 10, we find the vertices of the refined zone of \bar{y} in the arrangement \bar{H} in time $O(n^{d-1+\delta})$. Then we compute the distance from y to each of the induced hyperplanes corresponding to vertices of the refined zone in time $O(n^{d-1})$. From Lemma 7, one of them must be the nearest. ◀

5 A $(1 + \varepsilon)$ -approximation algorithm for the nearest induced simplex problem

We now consider the nearest induced simplex problem (Problem 6). The algorithm described in §2 for the case $k = 2$ and $d = 3$ can be adapted to work for this problem.

As in §2, consider the computation of the nearest induced segment under some polyhedral distance d_Q approximating the Euclidean distance. The reduction from this computation to edge-shooting still works with some minor tweak: if we shoot edges to find the nearest induced segment under d_Q , we may miss some of the segments. Fortunately, the points of these missed segments that are nearest to our query point under d_Q must be endpoints of

those segments. We can take those into account by comparing the nearest segment found by edge-shooting to the nearest neighbor, found in linear time. As before, edge-shooting is reduced to a counting problem.

Referring to the proof of Theorem 2 and Figure 1, the analogue of the counting problem in §2 for the nearest induced segment problem amounts to searching for the points b lying in the intersection of the cone C_a with the halfspace bounded by $\text{aff}(\Delta)$ that does not contain a . In dimension d , the affine hull of Δ is a hyperplane, and we restrict b to lie on one side of this hyperplane.

We therefore get a $(1 + \varepsilon)$ -approximation $O(n \log^d n)$ -time algorithm for the *nearest induced segment* problem in any fixed dimension d . This compares again favorably with the $(2 + \varepsilon)$ -approximation $O(n \log n)$ -time algorithm proposed in [25].

We generalize this to arbitrary values of k and prove the following result.

► **Theorem 11.** *For any constant positive real $\varepsilon > 0$ and constant positive integers d and k , there is a randomized $(1 + \varepsilon)$ -approximation algorithm for the nearest induced simplex problem in \mathbb{R}^d running in time $O(n^{k-1} \log^d n)$ with high probability.*

Again, we compute the nearest induced simplex under some polyhedral distance d_Q . As in the case $k = 2$, $(d - k)$ -face-shooting can be adapted to take care of missed simplices: for each $2 \leq k' \leq k$, shoot $(d - k')$ -faces of Q to find the nearest $(k' - 1)$ -simplex. For $k' = 1$, find the nearest neighbor in linear time. For any $(k - 1)$ -simplex, let $0 \leq k' \leq k$ be the smallest natural number such that no $(d - k')$ -face of Q hits the simplex when shot from the query point. It is obvious that, for all $t < k'$, some $(d - t)$ -face of Q hits the simplex, and that, for all $t \geq k'$, no $(d - t)$ -face of Q hits the simplex. For the sake of simplicity, we hereafter focus on solving the face-shooting problem when $k' = k$, thus ignoring the fact a simplex can be missed. Because the obtained running time will be of the order of $\tilde{O}(n^{k-1})$, the running time will be dominated by this case.

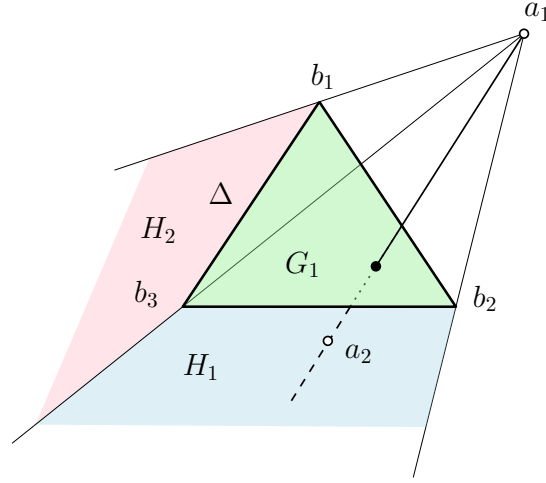
In order to reduce face-shooting to range counting queries, we need an analogue of Lemma 5 for convex combinations. Let A be a set of k points $\{a_1, a_2, \dots, a_k\}$, and let Δ be a $(d - k + 1)$ -simplex with vertices in $B = \{b_1, b_2, \dots, b_{d-k+2}\}$. We suppose that these points are in general position. We define the hyperplanes $H_i = \text{aff}(A \cup B \setminus \{b_i, a_k\})$, for $i \in [d - k + 2]$, and $G_j = \text{aff}(A \cup B \setminus \{a_j, a_k\})$, for $j \in [k - 1]$. We let H_i^+ be the halfspace supported by H_i that contains b_i , and G_j^- the halfspace supported by G_j that does not contain a_j .

► **Lemma 12.**

$$\text{conv}(A) \cap \Delta \neq \emptyset \iff a_k \in \left(\left(\bigcap_{i=1}^{d-k+2} H_i^+ \right) \cap \left(\bigcap_{j=1}^{k-1} G_j^- \right) \right).$$

Proof. (\Leftarrow) Suppose that $a_k \in (\bigcap_i H_i^+) \cap (\bigcap_j G_j^-)$. We have that $\text{conv}(A) \cap \Delta \neq \emptyset$ if and only if both $\text{aff}(A) \cap \Delta \neq \emptyset$ and $\text{conv}(A) \cap \text{aff}(\Delta) \neq \emptyset$ hold. From Lemma 5, we already have $\text{aff}(A) \cap \Delta \neq \emptyset$. It therefore remains to show that $\text{conv}(A) \cap \text{aff}(\Delta) \neq \emptyset$.

We first prove that $(\bigcap_j G_j) \cap \text{conv}(A) \neq \emptyset$. We proceed by induction on k . It can easily be shown to hold for $k = 2$. Let us suppose it holds for $k - 1$, and prove it for k . The hyperplane G_{k-1} separates a_{k-1} from a_k . Consider the point a'_{k-1} of the segment between a_{k-1} and a_k that lies on G_{k-1} . Let $A' = \{a_1, a_2, \dots, a_{k-2}, a'_{k-1}\}$. Consider the intersection G'_j of all hyperplanes G_j for $j \in [k - 2]$ with the subspace $\text{aff}(A')$. In the subspace $\text{aff}(A')$, The hyperplanes G'_j for $j \in [k - 2]$ all separate a_j from a'_{k-1} . Hence we can apply induction on A' and the hyperplanes G' in dimension $k - 2$, and we have that $(\bigcap_{j \in [k-2]} G'_j) \cap \text{conv}(A') \neq \emptyset$. Now because $a'_{k-1} \in \text{conv}(\{a_{k-1}, a_k\})$, we also have that $(\bigcap_{j \in [k-1]} G_j) \cap \text{conv}(A) \neq \emptyset$.



■ **Figure 5** Illustration of Lemma 12 in the case $d = 3$ and $k = 2$. The segment a_1a_2 intersects Δ if and only if a_2 is located in the colored region below Δ .

Now we also observe that $\bigcap_j G_j = \text{aff}(\Delta)$. The fact that $\text{aff}(\Delta) \subseteq \bigcap_j G_j$ is immediate since each G_j contains $\text{aff}(\Delta)$. To prove that $\bigcap_j G_j$ cannot contain more than $\text{aff}(\Delta)$ it suffices to show that those flats are of the same dimensions. Since the set $A \cup B$ is in general position, a_j (and a_k) cannot lie on G_j . Then we claim that the G_j are in general position. Indeed if they are not, then there must be some $1 \leq k' \leq k-1$ where $\bigcap_{j \leq k'-1} G_j = \bigcap_{j \leq k'} G_j$. However, this is not possible since $a_{k'} \in \bigcap_{j \leq k'-1} G_j$ but $a_{k'} \notin \bigcap_{j \leq k'} G_j$. The dimension of $\bigcap_j G_j$ is thus $d - k + 1$, the same as the dimension of $\text{aff}(\Delta)$.

Therefore, $\text{conv}(A) \cap \text{aff}(\Delta) \neq \emptyset$, as needed.

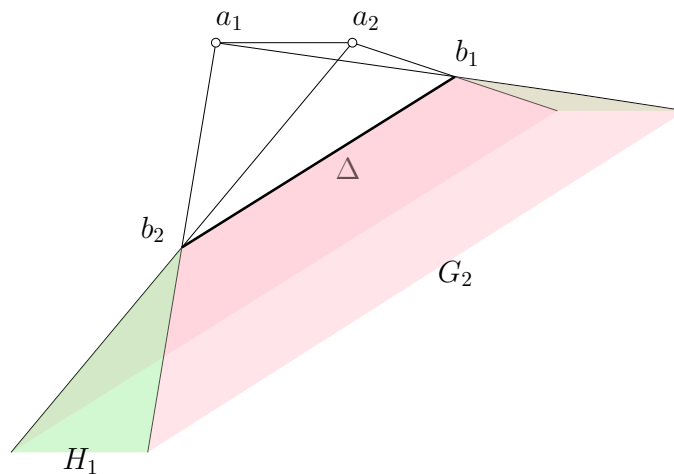
(\Rightarrow) Suppose that $a_k \notin (\bigcap_i H_i^+) \cap (\bigcap_j G_j^-)$. Then one of the halfspace does not contain a_k . It can be of the form H_i^+ or G_j^- . In both cases, all points of A are either contained in the hyperplane H_i or G_j , or lie in H_i^- or G_j^- . Hence the hyperplane H_i or G_j separates the interiors of the convex hulls. From the general position assumption, it also separates the convex hulls. ◀

The Lemma is illustrated on Figures 5 and 6 in the cases $d = 3, k = 2$, and $d = k = 3$.

Proof of Theorem 11. The algorithm follows the same steps as the algorithm described in the proof of Theorem 4, except that the ranges used in the orthogonal range counting data structure are different, and involve a higher-dimensional space.

We reduce the problem to that of counting the number of k -subsets A of S whose convex hull intersects a given $(d - k + 1)$ -simplex Δ . We already argued that when fixing the first $k - 2$ points a_1, a_2, \dots, a_{k-2} , the hyperplanes H_i induce a circular order on the points of S . Similarly, when the points a_1, a_2, \dots, a_{k-2} are fixed, the hyperplanes G_j all contain the $(d-2)$ -flat $\text{aff}(A \cup B \setminus \{a_j, a_{k-1}, a_k\})$, hence also induce a circular order on the points of S . Thus for each $(k - 2)$ -subset of S , we can assign $(d - k + 2) + (k - 1) = d + 1$ coordinates to each point of S , one for each family of hyperplanes. We then build an orthogonal range query data structure using these coordinates. For each point a_{k-1} , we query this data structure and count the number of points a_k such that $a_k \in (\bigcap_i H_i^+) \cap (\bigcap_j G_j^-)$. From Lemma 12, we can deduce the number of subsets A whose convex hull intersects Δ .

We can decrease by one the dimensionality of the ranges by realizing that the supporting hyperplane of G_{k-1}^- is unique as it does not depend on a_{k-1} , only the orientation of G_{k-1}^- does. To only output points a_k such that $a_k \in G_{k-1}^-$ we construct two data structures: one



■ **Figure 6** Illustration of Lemma 12 in the case $k = d = 3$. The triangle $a_1 a_2 a_3$ intersects Δ if and only if a_3 is located in the colored region.

with the points above G_{k-1} and one with the points below G_{k-1} . We query the relevant data structure depending whether a_{k-1} is above or below G_{k-1} . This spares a logarithmic factor and yields an overall running time of $O(n^{k-1} \log^{d-1} n)$ for the counting problem. Multiplying by the $O(\log n)$ rounds of binary search yields the claimed result. ◀

References

- 1 Pankaj K. Agarwal, Natan Rubin, and Micha Sharir. Approximate nearest neighbor search amid higher-dimensional flats. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 4:1–4:13, 2017. doi:10.4230/LIPIcs.ESA.2017.4.
- 2 Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, November 2006. doi:10.1109/TSP.2006.881199.
- 3 Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005. doi:10.1145/1059513.1059515.
- 4 Rahul Arya, Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. In *SODA*, pages 786–805. SIAM, 2020.
- 5 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. On the combinatorial complexity of approximating polytopes, April 2016. arXiv:1604.01175v4.
- 6 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Approximate convex intersection detection with applications to width and minkowski sums, July 2018. arXiv:1807.00484v1.
- 7 Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Near-optimal epsilon-kernel construction and related problems. In *Symposium on Computational Geometry*, volume 77 of *LIPIcs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 8 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998. doi:10.1145/293347.293348.
- 9 Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic algorithms for algebraic 3SUM. *Discrete & Computational Geometry*, 61(4):698–734, 2019. doi:10.1007/s00454-018-0040-y.

- 10 Ronen Basri, Tal Hassner, and Lih Zelnik-Manor. Approximate nearest subspace search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):266–278, 2011. doi:10.1109/TPAMI.2010.110.
- 11 Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *Ann. Statist.*, 44(2):813–852, April 2016. doi:10.1214/15-AOS1388.
- 12 Dimitris Bertsimas, Jean Pauphilet, and Bart Van Parys. Sparse Regression: Scalable algorithms and empirical performance. *arXiv e-prints*, February 2019. arXiv:1902.06547.
- 13 Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Information Theory*, 52(2):489–509, 2006. doi:10.1109/TIT.2005.862083.
- 14 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.
- 15 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.
- 16 Bernard Chazelle and Joel Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990. doi:10.1007/BF02122778.
- 17 G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, March 1997. doi:10.1007/BF02678430.
- 18 David L. Donoho. Compressed sensing. *IEEE Trans. Information Theory*, 52(4):1289–1306, 2006. doi:10.1109/TIT.2006.871582.
- 19 Richard M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974. doi:10.1016/0021-9045(74)90120-8.
- 20 Herbert Edelsbrunner, Raimund Seidel, and Micha Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993. doi:10.1137/0222031.
- 21 Jeff Erickson and Raimund Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete & Computational Geometry*, 13:41–57, 1995. doi:10.1007/BF02574027.
- 22 Dean P. Foster, Satyen Kale, and Howard J. Karloff. Online sparse linear regression. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*, pages 960–970, 2016. URL: <http://proceedings.mlr.press/v49/foster16.html>.
- 23 Dean P. Foster, Howard J. Karloff, and Justin Thaler. Variable selection is hard. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 696–709, 2015. URL: <http://proceedings.mlr.press/v40/Foster15.html>.
- 24 Anka Gajentaan and Mark H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 25 Sarel Har-Peled, Piotr Indyk, and Sepideh Mahabadi. Approximate sparse linear regression. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 77:1–77:14, 2018. doi:10.4230/LIPICs.ICALP.2018.77.
- 26 Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009. URL: <http://www.worldcat.org/oclc/300478243>.
- 27 Avner Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete & Computational Geometry*, 38(1):139–153, 2007. doi:10.1007/s00454-007-1329-4.
- 28 Sepideh Mahabadi. Approximate nearest line search in high dimensions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 337–354, 2015. doi:10.1137/1.9781611973730.25.
- 29 Stéphane Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Processing*, 41(12):3397–3415, 1993. doi:10.1109/78.258082.
- 30 Alan Miller. *Subset Selection in Regression*. Chapman and Hall/CRC, 2002.
- 31 Balas K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, 1995. doi:10.1137/S0097539792240406.

- 32 Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. URL: <http://www.jstor.org/stable/2346178>.
- 33 Csaba D. Tóth, Joseph O’Rourke, and Jacob E. Goodman, editors. *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 3rd edition, 2017. doi:10.1201/9781315119601.
- 34 Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985.

Drawing Graphs with Circular Arcs and Right-Angle Crossings

Steven Chaplick 

Maastricht University, the Netherlands
University of Würzburg, Germany
s.chaplick@maastrichtuniversity.nl

Henry Förster 

University of Tübingen, Germany
foersth@informatik.uni-tuebingen.de

Myroslav Kryven

University of Würzburg, Germany
myroslav.kryven@uni-wuerzburg.de

Alexander Wolff 

University of Würzburg, Germany

Abstract

In a RAC drawing of a graph, vertices are represented by points in the plane, adjacent vertices are connected by line segments, and crossings must form right angles. Graphs that admit such drawings are RAC graphs. RAC graphs are beyond-planar graphs and have been studied extensively. In particular, it is known that a RAC graph with n vertices has at most $4n - 10$ edges.

We introduce a superclass of RAC graphs, which we call *arc-RAC* graphs. A graph is arc-RAC if it admits a drawing where edges are represented by circular arcs and crossings form right angles. We provide a Turán-type result showing that an arc-RAC graph with n vertices has at most $14n - 12$ edges and that there are n -vertex arc-RAC graphs with $4.5n - O(\sqrt{n})$ edges.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces; Mathematics of computing → Combinatoric problems

Keywords and phrases circular arcs, right-angle crossings, edge density, charging argument

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.21

Funding *Myroslav Kryven*: M. Kryven acknowledges support from DFG grant WO 758/9-1.

Acknowledgements We thank the reviewers of our paper for their very detailed comments, which helped us to improve the writing a lot.

1 Introduction

A *drawing* of a graph in the plane is a mapping of its vertices to distinct points and each edge uv to a curve whose endpoints are u and v . Planar graphs, which admit crossing-free drawings, have been studied extensively. They have many nice properties and several algorithms for drawing them are known, see, e.g., [19, 20]. However, in practice we must also draw non-planar graphs and crossings make it difficult to understand a drawing. For this reason, graph classes with restrictions on crossings are studied, e.g., graphs that can be drawn with at most k crossings per edge (known as *k-planar graphs*) or where the angles formed by each crossing are “large”. These classes are categorized as *beyond-planar* graphs and have experienced increasing interest in recent years [13].

As introduced by Didimo et al. [12], a prominent beyond-planar graph class that concerns the crossing angles is the class of k -bend right-angle-crossing graphs, or RAC_k graphs for short, that admit a drawing where all crossings form 90° angles and each edge is a polygonal



© Steven Chaplick, Henry Förster, Myroslav Kryven, and Alexander Wolff;
licensed under Creative Commons License CC-BY

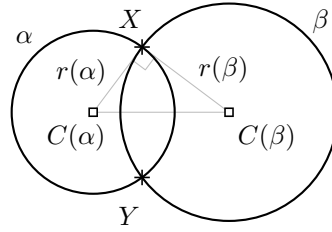
17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Circles α and β are orthogonal if and only if $\triangle XC(\alpha)C(\beta)$ is right-angled.

chain with at most k bends. Using right-angle crossings and few bends is motivated by several cognitive studies suggesting a positive correlation between large crossing angles or small curve complexity and the readability of a graph drawing [16–18]. Didimo et al. [12] studied the edge density of RAC_k graphs. They showed that RAC_0 graphs with n vertices have at most $4n - 10$ edges (which is tight), that RAC_1 graphs have at most $O(n^{\frac{4}{3}})$ edges, that RAC_2 graphs have at most $O(n^{\frac{7}{4}})$ edges and that all graphs are RAC_3 . Dujmović et al. [14] gave an alternative simple proof of the $4n - 10$ bound for RAC_0 graphs using charging arguments similar to those of Ackerman and Tardos [2] and Ackerman [1]. Arikushi et al. [5] improved the upper bounds to $6.5n - 13$ for RAC_1 graphs and to $74.2n$ for RAC_2 graphs. The bound of $6.5n - 13$ for RAC_1 graphs was also obtained by charging arguments. They also provided a RAC_1 graph with $4.5n - O(\sqrt{n})$ edges. The best known lower and upper bound for the maximum edge density of RAC_1 graphs of $5n - 10$ and $5.5n - 11$, respectively, are due to Angelini et al. [4].

We extend the class of RAC_0 graphs by allowing edges to be drawn as circular arcs but still requiring 90° crossings. An angle at which two circles intersect is the angle between the two tangents to each of the circles at an intersection point. Two circles intersecting at a right angle are called *orthogonal*. For any circle γ , let $C(\gamma)$ be its center and let $r(\gamma)$ be its radius. The following observation follows from the Pythagorean theorem.

► **Observation 1.1.** *Let α and β be two circles. Then α and β are orthogonal if and only if $r(\alpha)^2 + r(\beta)^2 = |C(\alpha)C(\beta)|^2$; see Figure 1.*

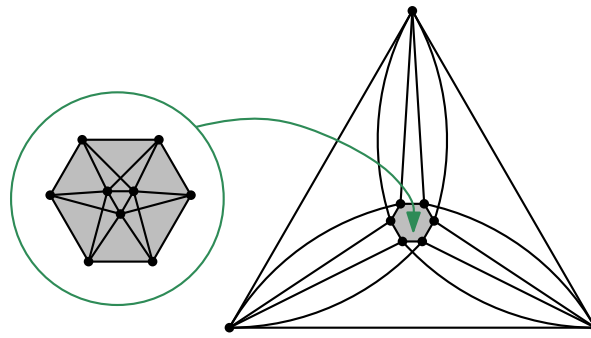
In addition we note the following.

► **Observation 1.2.** *Given a pair of orthogonal circles, the tangent to one circle at one of the intersection points goes through the center of the other circle; see Figure 1. In particular, a line is orthogonal to a circle if the line goes through the center of the circle.*

Similarly, two circular arcs α and β are orthogonal if they intersect properly (that is, ignoring intersections at endpoints) and the underlying circles (that contain the arcs) are orthogonal. For the remainder of this paper, all arcs will be circular arcs. We consider any straight-line segment to be an arc with infinite radius. Note, though, that the above observations do not hold for (pairs of) circles of infinite radius. As in the case of circles, for any arc γ of finite radius, let $C(\gamma)$ be its center.

We call a drawing of a graph an *arc-RAC drawing* if the edges are drawn as arcs and any pair of intersecting arcs is orthogonal; see Figure 2. A graph that admits an arc-RAC drawings is called an *arc-RAC graph*.

The idea of drawing graphs with arcs dates back to at least the work of the artist Mark Lombardi who drew social networks, featuring players from the political and financial sector [22]. Indeed, user studies [25, 27] state that users prefer edges drawn with curves of small curvature; not necessarily for performance (that is, tasks such as finding shortest paths,



■ **Figure 2** An arc-RAC drawing of a graph. This graph is not RAC_0 [6].

identifying common neighbors, or determining vertex degrees) but for aesthetics. Drawing graphs with arcs can help to improve certain quality measures of a drawing such as angular resolution [3, 11] or visual complexity [21, 26].

An immediate restriction on the edge density of arc-RAC graphs is imposed by the following known result.

► **Lemma 1.3** ([23]). *In an arc-RAC drawing, there cannot be four pairwise orthogonal arcs.*

It follows from Lemma 1.3 that arc-RAC graphs are *4-quasi-planar*, that is, an arc-RAC drawing cannot have four edges that pairwise cross. This implies that an arc-RAC graph with n vertices can have at most $72(n - 2)$ edges [1].

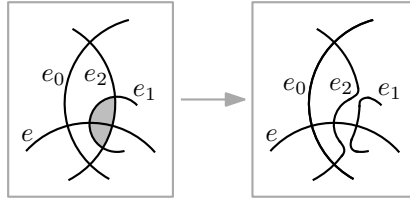
Our main contribution is that we reduce this bound to $14n - 12$ using charging arguments similar to those of Ackerman [1] and Dujmović et al. [14]; see Section 2. For us, the main challenge was to apply these charging arguments to a modification of an arc-RAC drawing and to exploit, at the same time, geometric properties of the original arc-RAC drawing to derive the bound. We also provide a lower bound of $4.5n - O(\sqrt{n})$ on the maximum edge density of arc-RAC graphs based on the construction of Arikushi et al. [5]; see Section 3. We conclude with some open problems in Section 4. Throughout the paper our notation won't distinguish between the entities (vertices and edges) of an abstract graph and the geometric objects (points and curves) representing them in a drawing.

As usual for topological drawings, we forbid vertices to lie in the relative interior of an edge and we do not allow edges to *touch*, that is, to have a common point in their relative interiors without crossing each other at this point. Hence an *intersection point* of two edges is always a *crossing*. When we say that two edges *share a point*, we mean that they either cross each other or have a common endpoint.

2 An Upper Bound for the Maximum Edge Density

Let G be a 4-quasi-planar graph, and let D be a 4-quasi-planar drawing of G . In his proof of the upper bound on the edge density of 4-quasi-planar graphs, Ackerman [1] first modified the given drawing so as to remove faces of small degree. We use a similar modification that we now describe.

Consider two edges e_1 and e_2 in D that intersect multiple times. A region in D bounded by pieces of e_1 and e_2 that connect two consecutive crossings or a crossing and a vertex of G is called a *lens*. If a lens is adjacent to a crossing and a vertex of G , then we call such a lens a *1-lens*, otherwise a *0-lens*. A lens that does not contain a vertex of G is *empty*. Every drawing with 0-lenses has a *smallest empty 0-lens*, that is, an empty 0-lens that does not



■ **Figure 3** A simplification step resolves a smallest empty 0-lens; if two edges e_1 and e_2 change the order in which they cross the edge e , they form an empty 0-lens intersecting e before the step, and thus, in the original 4-quasi-planar drawing.

contain any other empty 0-lenses in its interior. We can swap [1, 24] the two curves that bound a smallest empty 0-lens; see Figure 3. We call such a swap a *simplification step*. Since a simplification step resolves a smallest empty lens, we observe the following.

► **Observation 2.1.** *A simplification step does not introduce any new pairs of crossing edges or any new empty lenses.*

We exhaustively apply simplification steps to our drawing and refer to this as the *simplification process*. Observation 2.1 guarantees that applying the *simplification process* to a drawing D terminates, that is, it results in an empty-0-lens-free drawing D' of G . We call the resulting drawing D' *simplified*; it is a *simplification* of D . Observation 2.1 implies the following important property of any simplification step.

► **Observation 2.2.** *Applying a simplification step to a 4-quasi-planar drawing yields a 4-quasi-planar drawing.*

As mentioned above, Ackerman [1] used a similar modification to prepare a 4-quasi-planar drawing for his charging arguments; note, that unlike Ackerman, we do not resolve 1-lenses. We look at the simplification process in more detail, in particular, we consider how it changes the order in which edges cross.

► **Lemma 2.3.** *Let D be an arc-RAC drawing, and let D' be a simplification of D . If two edges e_1 and e_2 cross another edge e in D' in an order different from that in D , then e_1 and e_2 form an empty 0-lens intersecting e in D .*

Proof. Let e_1 and e_2 be two edges as in the statement of the lemma. Then there is a simplification step i where the order in which e_1 and e_2 cross e changes. Let D_i be the drawing immediately before simplification step i , and let D_{i+1} be the drawing right after step i . By construction, the order in which e_1 and e_2 cross e is different in D_i and in D_{i+1} . Since D_i is 4-quasi-planar (see Observation 2.2) and since we always resolve a smallest empty 0-lens, the edges e_1 and e_2 form a smallest empty 0-lens in D_i ; see Figure 3. Given that the simplification process does not introduce new empty lenses (see Observation 2.1), e_1 and e_2 form an empty 0-lens in the original 4-quasi-planar drawing. ◀

We now focus on the special type of 4-quasi-planar drawings we are interested in. Suppose that G is an arc-RAC graph, D is an arc-RAC drawing of G , and D' is a simplification of D . Note that, in general, D' is not an arc-RAC drawing. If two edges e_1 and e_2 cross in D' , then they do not form an empty 0-lens in D . This holds because for any two edges forming an empty 0-lens in D , the simplification process removes both of their crossings; therefore, in D' the two edges do not have any crossings. If e_1 and e_2 are incident to the same vertex, they also do not form an empty 0-lens in D , as otherwise they would share three points in D (the two crossing points of the lens and the common vertex of G). Thus, we have the following observation.

► **Observation 2.4.** *Let D be an arc-RAC drawing, and let D' be a simplification of D . If two edges e_1 and e_2 share a point in D' , then they do not form an empty 0-lens in D .*

In the following, we first state the main theorem of this section and provide the structure of its proof (deferring one small lemma and the main technical lemma until later). Then, we prove the remaining technical details in Lemmas 2.6 to 2.10 to establish the result.

► **Theorem 2.5.** *An arc-RAC graph with n vertices can have at most $14n - 12$ edges.*

Proof. Let $G = (V, E)$ be an arc-RAC graph, let D be an arc-RAC drawing of G , let D' be a simplification of D , and let $G' = (V', E')$ be the planarization of D' . Our charging argument consists of three steps.

First, each face f of G' is assigned an initial charge $ch(f) = |f| + v(f) - 4$, where $|f|$ is the degree of f in the planarization and $v(f)$ is the number of vertices of G on the boundary of f . Applying Euler's formula several times, Ackerman and Tardos [2] showed that $\sum_{f \in G'} ch(f) = 4n - 8$, where n is the number of vertices of G . In addition, we set the charge $ch(v)$ of a vertex v of G to $16/3$. Hence the total charge of the system is $4n - 8 + 16n/3 = 28n/3 - 8$.

In the next two steps (described below), similarly to Dujmović et al. [14], we redistribute the charges among faces of G' and vertices of G so that, for every face f , the final charge $ch_{\text{fin}}(f)$ is at least $v(f)/3$ and the final charge of each vertex is non-negative. Observing that

$$28n/3 - 8 \geq \sum_{f \in G'} ch_{\text{fin}}(f) \geq \sum_{f \in G'} v(f)/3 = \sum_{v \in G} \deg(v)/3 = 2|E|/3$$

yields that the number of edges of G is at most $14n - 12$ as claimed. (The second-last equality holds since both sides count the number of vertex-face incidences in G' .)

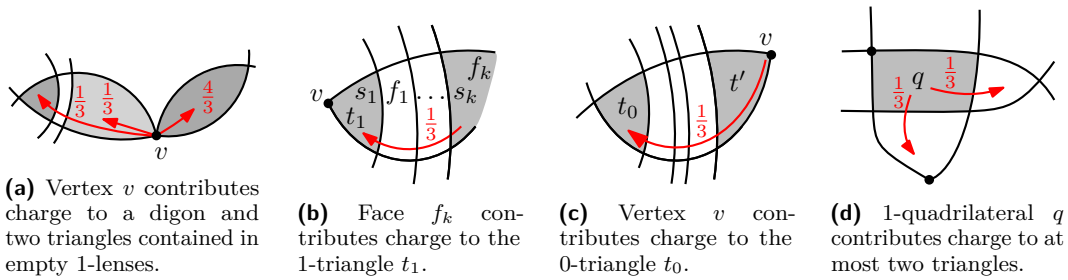
After the first charging step above, it is easy to see that $ch(f) \geq v(f)/3$ holds if $|f| \geq 4$. We call a face f of G' a k -triangle, k -quadrilateral, or k -pentagon if f has the corresponding shape and $v(f) = k$. Similarly, we call a face of degree two a *digon*. Note that any digon is a 1-digon since all empty 0-lenses have been simplified.

After the first charging step, each digon and each 0-triangle has a charge of -1 , and each 1-triangle has a charge of 0 . Thus, in the second charging step, we need to find $4/3$ units of charge for each digon, one unit of charge for each 0-triangle, and $1/3$ unit of charge for each 1-triangle. Note that all other faces including 2- and 3-triangles already have sufficient charge.

To charge a digon d incident to a vertex v of G , we decrease $ch(v)$ by $4/3$ and increase $ch(d)$ by $4/3$; see Figure 4a. We say that v *contributes* charge to d .

To charge triangles, we proceed similarly to Ackerman [1] and Dujmović et al. [14, Theorem 7].

Consider a 1-triangle t_1 . Let v be the unique vertex incident to t_1 , and let $s_1 \in E'$ be the edge of t_1 opposite of v ; see Figure 4b. Note that the endpoints of s_1 are intersection points in D' . Let f_1 be the face on the other side of s_1 . If f_1 is a 0-quadrilateral, then we consider its edge $s_2 \in E'$ opposite to s_1 and the face f_2 on the other side of s_2 . We continue iteratively until we meet a face f_k that is not a 0-quadrilateral. If f_k is a triangle, then all the faces $t_1, f_1, f_2, \dots, f_k$ belong to the same empty 1-lens l incident to the vertex v of t_1 . In this case, we decrease $ch(v)$ by $1/3$ and increase $ch(t_1)$ by $1/3$; see Figure 4a. Otherwise, f_k is not a triangle and $|f_k| + v(f_k) - 4 \geq 1$ (see Figure 4b). In this case, we decrease $ch(f_k)$ by $1/3$ and increase $ch(t_1)$ by $1/3$. We say that the face f_k *contributes* charge to the triangle t_1 over its side s_k .



■ **Figure 4** Transferring charge from vertices and high-degree faces to small-degree faces.

For a 0-triangle t_0 , we repeat the above charging over each side. If the last face on our path is a triangle t' , then t_0 and t' are contained in an empty 1-lens (recall that D' does not contain empty 0-lenses) and t' is a 1-triangle incident to a vertex v of G . In this case, we decrease $ch(v)$ by $1/3$ and increase $ch(t_0)$ by $1/3$; see Figure 4c.

Thus, at the end of the second step, the charge of each digon and triangle f is at least $v(f)/3$. Note that the charge of f comes either from a higher-degree face or from a vertex v incident to an empty 1-lens containing f .

- In the third step, we do not modify the charging any more, but we need to ensure that
- (i) $ch(f) \geq v(f)/3$ still holds for each face f of G' with $|f| \geq 4$ and
 - (ii) $ch(v) \geq 0$ for each v of G .

We first show statement (i). Ackerman [1] noted that a face f with $|f| \geq 4$ can contribute charges over each of its edges at most once. Moreover, f can contribute at most one third unit of charge over each of its edges. Therefore, if $|f| + v(f) \geq 6$, then in the worst case (that is, f contributes charge over each of its edges) f still has a charge of $|f| + v(f) - 4 - |f|/3 \geq v(f)/3$. Thus, it remains to verify that 1-quadrilaterals and 0-pentagons, which initially had only one unit of charge, have a charge of at least $1/3$ unit or zero, respectively, at the end of the second step.

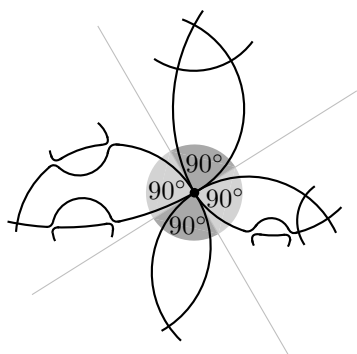
A 1-quadrilateral q can contribute charge to at most two triangles since the endpoints of any edge of G' over which a face contributes charge must be intersection points in D' ; see Figure 4d and recall that q now plays the role of f_k in Figure 4b.

A 0-pentagon cannot contribute charge to more than three triangles; see Lemma 2.10.

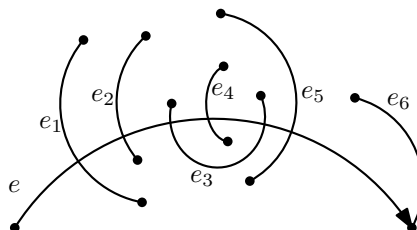
Now we show statement (ii). Recall that a vertex v can contribute charge to a digon incident to v or to at most two triangles contained in an empty 1-lens incident to v . Observe that two empty 1-lenses with either triangles or a digon taking charge from v cannot overlap; see Figure 4a. We show in Lemma 2.6 that v cannot be incident to more than four such empty 1-lenses. In the worst case, v contributes $4/3$ units of charge to each of the at most four incident digons representing these empty 1-lenses. Thus, v has non-negative charge at the end of the second step. ◀

► **Lemma 2.6.** *In any simplified arc-RAC drawing, each vertex is incident to at most four non-overlapping empty 1-lenses.*

Proof. Let v be a vertex incident to some non-overlapping empty 1-lenses. Consider a small neighborhood of the vertex v in the simplified drawing and notice that in this neighborhood the simplified drawing is the same as the original arc-RAC drawing. Let l be one of the



■ **Figure 5** The edges of an empty 1-lens form a $\pi/2$ angle at the vertex of the lens.



■ **Figure 6** The relation $\Pi(\cdot; \cdot)$ does not necessarily describe *all* intersection points along an edge. Here, e.g., $\Pi(e; e_1, e_2, e_3, e_4, e_5, e_6)$ and $\Pi(e; e_1, e_3, e_4, e_5)$ both hold.

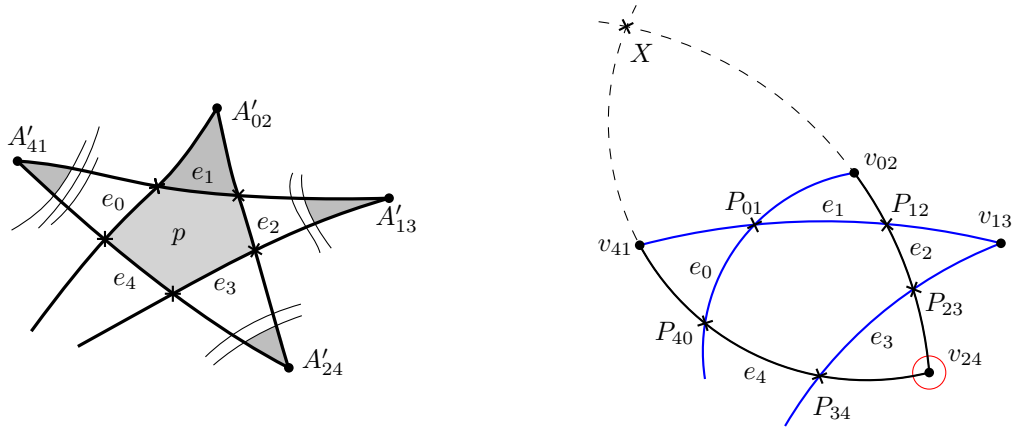
non-overlapping empty 1-lenses incident to v . Then l forms an angle of 90° between the two edges incident to v that form l ; see Figure 5. This is due to the fact that the other “endpoint” of l is an intersection point where the two edges must meet at 90° . Thus v is incident to at most four non-overlapping empty 1-lenses. ◀

We now set the stage for proving Lemma 2.10, which shows that a 0-pentagon in a simplified drawing does not contribute charge to more than three triangles. The proof goes by a contradiction. Consider a 0-pentagon that contributes charge to at least four triangles in the simplified drawing. First, we examine which edges of this 0-pentagon cross; see Lemma 2.7. We then describe the order in which these edges share points in the simplified drawing and show that the original arc-RAC drawing must adhere to the same order; see Lemma 2.8. Finally, we use geometric arguments to show that, under these order constraints, an arc-RAC drawing of the edges does not exist; see Lemma 2.9.

Let D be an arc-RAC drawing of some arc-RAC graph $G = (V, E)$, let D' be its simplification, and let p be a 0-pentagon that contributes charge to at least four triangles. Let s_0, s_1, \dots, s_4 be the sides of p in clockwise order and denote the edges of G that contain these sides as e_0, e_1, \dots, e_4 so that edge e_0 contains side s_0 etc. Since p contributes charge over at least four sides, these sides are consecutive around p . Without loss of generality, we assume that s_4 is the side over which p does not necessarily contribute charge.

For $i \in \{0, 1, 2, 3\}$, let t_i be the triangle that gets charge from p over the side s_i . The triangle t_i is bounded by the edges e_{i-1} and e_{i+1} . (Indices are taken modulo 5.) Note that all faces bounded by e_{i-1} and e_{i+1} that are between t_i and p must be 0-quadrilaterals. If t_i is a 1-triangle, then e_{i-1} and e_{i+1} are incident to the same vertex of the triangle. Otherwise, t_i is a 0-triangle and e_{i-1} and e_{i+1} cross at a vertex of the triangle. Let $A'_{i-1, i+1}$ denote this common point of e_{i-1} and e_{i+1} , and let $E_p = \{e_0, \dots, e_4\}$; see Figure 7a.

We now describe the order in which the edges in E_p share points in D' . To this end, we orient the edges in E_p so that this orientation conforms with the orientation of a clockwise walk around the boundary of p in D' . In addition, we write $\Pi(e_k; e_{i_1}, e_{i_2}, \dots, e_{i_l})$ if the edge e_k shares points (either crossing points or vertices of the graph) with the edges $e_{i_1}, e_{i_2}, \dots, e_{i_l}$ in this order with respect to the orientation of e_k ; see Figure 6. (Note that we can have $\Pi(e_k; e_i, e_j, e_i)$ as edges may intersect twice. We will not consider more than two edges sharing the same endpoint.) Due to the order in which we numbered the edges in E_p , it holds in D' that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$, and, for $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$; see Figure 7a. Now we show that in D the order is the same. Obviously every pair of edges



(a) Notation: A 0-pentagon p in D' and the edges in E_p . The points of type $A'_{i-1, i+1}$ are either intersection points or vertices of G .

(b) Also in D , it holds that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$, and, for $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$.

■ **Figure 7** A 0-pentagon cannot contribute charge to more than three triangles.

(e_{i-1}, e_{i+1}) that shares an endpoint in D' also shares an endpoint in D . Furthermore, every pair (e_i, e_{i+1}) or (e_{i-1}, e_{i+1}) of crossing edges crosses in D , too, because the simplification process does not introduce new pairs of crossing edges; see Observation 2.1.

► **Lemma 2.7.** *In the drawing D , the edges e_0 and e_3 do not cross.*

Proof. Assume that the edges e_0 and e_3 cross in D and notice that each of the pairs of edges (e_0, e_1) , (e_1, e_2) , and (e_2, e_3) forms a crossing in D' (see Figure 7a), and hence in D , too. For any arc e , let \bar{e} denote the circle containing e . Recall that a family of *Apollonian circles* [9, 23] consists of two sets of circles such that each circle in one set is orthogonal to each circle in the other set. Thus, the pairs of circles (\bar{e}_1, \bar{e}_3) and (\bar{e}_0, \bar{e}_2) belong to such a family; the pair (\bar{e}_1, \bar{e}_3) belongs to one set of the family and (\bar{e}_0, \bar{e}_2) belongs to the other set. If not all of the circles in the family share the same point, which is the case for the circles $\bar{e}_0, \bar{e}_1, \bar{e}_2$, and \bar{e}_3 , then one such set consists of disjoint circles. So either the pair (\bar{e}_0, \bar{e}_2) or the pair (\bar{e}_1, \bar{e}_3) must consist of disjoint circles. This is a contradiction because each of the two pairs shares a point in D' (see Figure 7a), and thus, in D . ◀

► **Lemma 2.8.** *In the drawing D , it holds that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$, and, for each $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$.*

Proof. Recall that in the drawing D' , it holds that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$, and, for each $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$; see Figure 7a. Consider distinct indices $i, j, k \in \{0, 1, 2, 3, 4\}$ so that the edges e_i and e_j share points with e_k in this order in D' , that is, $\Pi(e_k; e_i, e_j)$ in D' . We will show that the edges e_i and e_j share points with e_k in the same order in D , that is, $\Pi(e_k; e_i, e_j)$ in D . In other words, the order in which the edges in E_p share points in D is the same as in D' .

First, note that if the edge e_i or the edge e_j shares an endpoint with e_k , then e_i and e_j do not change the order in which they share points with e_k . This is due to the fact that the simplification process does not modify the graph. Therefore, e_i and e_j share points with e_k in the same order in D as in D' , that is, $\Pi(e_k; e_i, e_j)$ in D .

Assume now that both e_i and e_j cross e_k .

If $(i, j) \in \{(0, 3), (3, 0)\}$, then, according to Lemma 2.7, the edges e_i and e_j do not cross in D , so they do not form an empty 0-lens in D , and thus, by Lemma 2.3, e_i and e_j cross e_k in the same order in D as in D' , that is, $\Pi(e_k; e_i, e_j)$ in D .

Otherwise, the edges e_i and e_j share a point in D' ; see Figure 7a. Therefore, by Observation 2.4, e_i and e_j do not form an empty 0-lens in D , and thus, by Lemma 2.3, e_i and e_j cross e_k in the same order in D as in D' , that is, $\Pi(e_k; e_i, e_j)$ in D . ◀

Thus, we have shown that the order in which the edges in E_p share points in D is the same as in D' , see Figure 7b. We show now that an arc-RAC drawing with this order does not exist; see Lemma 2.9. This is the main ingredient to prove Lemma 2.10, which says that a 0-pentagon in a simplified arc-RAC drawing contributes charge to at most three triangles.

For simplicity of presentation and without loss of generality, we assume that the points $A'_{i-1, i+1}$ are vertices of G , which we denote by $v_{i-1, i+1}$.

► **Lemma 2.9.** *The edges in E_p do not admit an arc-RAC drawing where it holds that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$, and, for $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$.*

Proof. Assume that the edges in E_p admit an arc-RAC drawing where they share points in the order indicated above. For $i \in \{0, \dots, 4\}$, let $P_{i, i+1}$ be the intersection point of e_i and e_{i+1} ; see Figure 7b. Note that on e_i , the point $P_{i-1, i}$ is before the point $P_{i, i+1}$ (due to $\Pi(e_i; e_{i-1}, e_{i+1})$).

Recall that an *inversion* [23] with respect to a circle α , the *inversion circle*, is a mapping that takes any point $P \neq C(\alpha)$ to a point P' on the straight-line ray from $C(\alpha)$ through P so that $|C(\alpha)P'| \cdot |C(\alpha)P| = r(\alpha)^2$. Inversion maps each circle not passing through $C(\alpha)$ to another circle and each circle passing through $C(\alpha)$ to a line. The center of the inversion circle is mapped to the “point at infinity”. It is known that inversion preserves angles.

We invert the drawing of the edges in E_p with respect to a small inversion circle centered at v_{24} . Let e_i° be the image of e_i , $v_{i-1, i+1}^\circ$ be the image of $v_{i-1, i+1}$ (v_{24}° is the point at infinity), and $P_{i, i+1}^\circ$ be the image of $P_{i, i+1}$. Because in the pre-image the arcs e_2 and e_4 pass through v_{24} , in the image e_2° and e_4° are straight-line rays. We assume that in the image e_2° meets e_4° at the point at infinity, that is, at v_{24}° . Then, taking into account that inversion is a continuous and injective mapping, the order in which the edges in E_p share points is the same in the image.

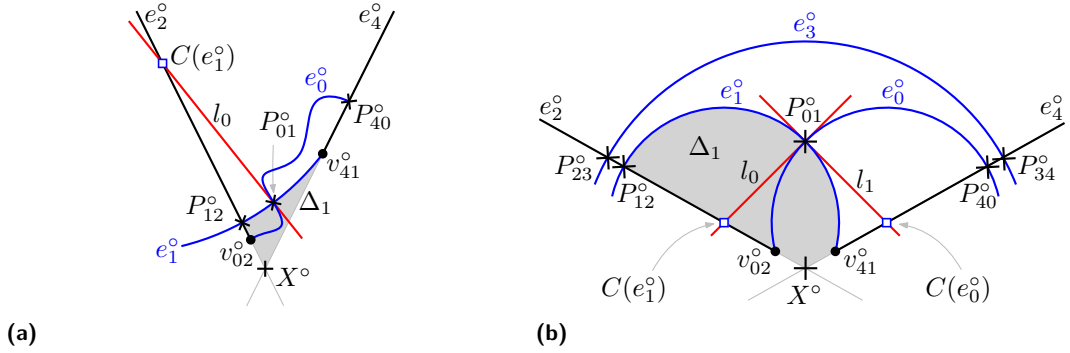
We consider two cases regarding whether the edges e_2 and e_4 belong to two different circles or not.

Case I: e_2 and e_4 belong to two different circles.

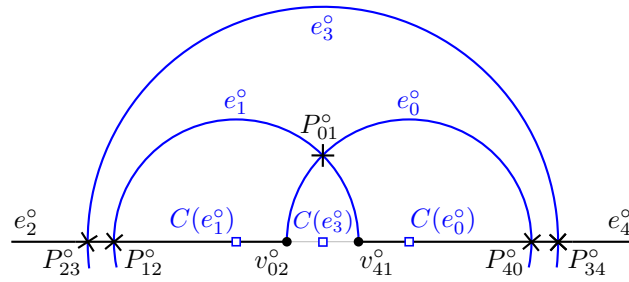
One of the intersection points of their circles is v_{24} , and we let X denote the other intersection point. Here we have that e_2° and e_4° are two straight-line rays meeting at infinity at v_{24}° . Their supporting lines are different and intersect at X° , which is the image of X ; see Figure 8.

We now assume for a contradiction that the arc e_1° forms a concave side of the triangle $\Delta_1 = P_{12}^\circ v_{41}^\circ X^\circ$; see Figure 8a where the triangle is filled gray. (Symmetrically, we can show that the arc e_0° cannot form a concave side of the triangle $\Delta_0 = P_{40}^\circ v_{02}^\circ X^\circ$.) By Observation 1.2, $C(e_1^\circ)$ must lie on the ray e_2° . Since we assume that the arc e_1° forms a concave side of the triangle Δ_1 , $C(e_1^\circ)$ and v_{02}° are separated by P_{12}° on e_2° . Consider the tangent l_0 to e_0° at P_{01}° . Again in light of Observation 1.2, l_0 has to go through $C(e_1^\circ)$ because e_0° and e_1° are orthogonal. On the one hand, v_{02}° is to the same side of l_0 as P_{12}° ; see Figure 8a. On the other hand, l_0 separates P_{12}° and v_{41}° due to $\Pi(e_1; e_4, e_0, e_2)$. Moreover, l_0 does not separate v_{41}° and P_{40}° since it intersects the line of e_4° when leaving the gray triangle Δ_1 . So the two points v_{02}° and P_{40}° of the same arc e_0° are separated by l_0 , which is a tangent of this arc; contradiction.

21:10 Drawing Graphs with Circular Arcs and Right-Angle Crossings



■ **Figure 8** Illustration for the proof of Lemma 2.9 when e_2 and e_4 belong to two different circles. Image of the inversion with respect to the red circle in Figure 7b.



■ **Figure 9** Illustration to the proof of Lemma 2.9 when e_2 and e_4 belong to the same circle. Image of the inversion with respect to the red circle in Figure 7b.

Thus, the arc e_1° forms a convex side of the triangle Δ_1 , and e_0° forms a convex side of Δ_0 ; see Figure 8b. Now, due to Observation 1.2, $C(e_0^\circ)$ is between v_{41}° and P_{40}° , and $C(e_1^\circ)$ is between v_{02}° and P_{12}° , because that is where the tangents l_1 of e_1° and l_0 of e_0° in P_{01}° intersect the lines of e_4° and e_2° , respectively. Taking into account that $C(e_3^\circ) = X^\circ$, because e_3° is orthogonal to both e_2° and e_4° , we obtain that the points $C(e_3^\circ), C(e_1^\circ), P_{12}^\circ, P_{23}^\circ$ appear on the line of e_2° in this order. Thus, the circle of e_1° is contained within the circle of e_3° . This is a contradiction because e_3° and e_1° must share a point; namely v_{13}° .

Case II: e_2 and e_4 belong to the same circle.

Here e_2° and e_4° are two disjoint straight-line rays on the same line l (meeting at infinity at v_{24}°); see Figure 9. We direct l as e_4° and e_2° (from right to left in Figure 9). Because e_0°, e_1° , and e_3° are orthogonal to l , their centers have to be on l . Due to our initial assumption, we have $\Pi(e_4; e_2, e_3, e_0, e_1)$ and $\Pi(e_2; e_0, e_1, e_3, e_4)$. Hence, along l , we have $P_{34}^\circ, P_{40}^\circ, v_{41}^\circ$, (on e_4°) and then $v_{02}^\circ, P_{12}^\circ, P_{23}^\circ$ (on e_2°). Therefore, the circle of e_1° is contained in that of e_3° . Hence, e_1° does not share a point with e_3° ; a contradiction. ◀

► **Lemma 2.10.** *A 0-pentagon in a simplified arc-RAC drawing contributes charge to at most three triangles.*

Proof. As discussed above, if a 0-pentagon formed by edges e_0, e_1, \dots, e_4 contributes charge to more than three triangles in a simplified drawing (see Figure 7a), then this implies the existence of an arc-RAC drawing where it holds that $\Pi(e_0; e_4, e_1, e_2)$, $\Pi(e_3; e_1, e_2, e_4)$ and, for $i \in \{1, 2, 4\}$, $\Pi(e_i; e_{i-2}, e_{i-1}, e_{i+1}, e_{i+2})$; see Figure 7b. This, however, contradicts Lemma 2.9. ◀

With the proofs of Lemmas 2.6 and 2.10 now in place, the proof of Theorem 2.5 is complete.

3 A Lower Bound for the Maximum Edge Density

In this section, we construct a family of arc-RAC graphs with high edge density. Our construction is based on a family of RAC₁ graphs of high edge density that Arikushi et al. [5] constructed. Let G be an embedded graph whose vertices are the vertices of the hexagonal lattice clipped inside a rectangle; see Figure 10a. The edges of G are the edges of the lattice and, inside each hexagon that is bounded by the cycle (P_0, \dots, P_5) , six additional edges $(P_i, P_{i+2 \bmod 6})$ for $i \in \{0, 1, \dots, 5\}$; see Figure 10b. We refer to a part of the drawing made up of a single hexagon and its diagonals as a *tile*. In Theorem 3.3 below, we show that each hexagon can be drawn as a regular hexagon and its diagonals can be drawn as two sets of arcs $A = \{\alpha_0, \alpha_1, \alpha_2\}$ and $B = \{\beta_0, \beta_1, \beta_2\}$, so that the arcs in A are pairwise orthogonal, the arcs in B are pairwise non-crossing, and for each arc in B intersecting another arc in A the two arcs are orthogonal; we use this construction to establish the theorem. In particular, the arcs in A form the 3-cycle (P_0, P_2, P_4) , and the arcs in B form the 3-cycle (P_1, P_3, P_5) .

We first define the radii and centers of the arcs in a tile and show that they form only orthogonal crossings. We use the geometric center of the tile as the origin of our coordinate system in the following analysis. We now discuss the arcs in A ; then we turn to the arcs in B . For each $j \in \{0, 1, 2\}$, the arc α_j has radius $r_A = 1$ and center $C(\alpha_j) = (d_A \cos(\pi/6 + j\frac{2\pi}{3}), d_A \sin(\pi/6 + j\frac{2\pi}{3}))$, where $d_A = \sqrt{2/3}$ is the distance of the centers from the origin; see Figure 11a.

► **Lemma 3.1.** *The arcs in A are pairwise orthogonal.*

Proof. Consider the equilateral triangle $\triangle C(\alpha_0)C(\alpha_1)C(\alpha_2)$ formed by the centers of the three arcs in A . Because the origin is in the center of the triangle, the edge length of the triangle is $2d_A \cos \pi/6 = \sqrt{2}$, and so the distance between the centers of any two arcs is $\sqrt{2}$. The radii of the arcs are 1, hence by Observation 1.1, every two arcs are orthogonal. ◀

As in Figure 11b, for each $j \in \{0, 1, 2\}$, the arc β_j has radius $r_B = \sqrt{\frac{70+40\sqrt{3}}{6}}$ and center $C(\beta_j) = (d_B \cos(\frac{\pi}{2} + \frac{(j+1)2\pi}{3}), d_B \sin(\frac{\pi}{2} + \frac{(j+1)2\pi}{3}))$, where $d_B = \sqrt{\frac{1}{6}} + \sqrt{\frac{73+40\sqrt{3}}{6}}$ is the distance of the centers from the origin.

► **Lemma 3.2.** *If an arc in B intersects an arc in A , then the two arcs are orthogonal.*

Proof. Let $i, j \in \{0, 1, 2\}$. If $j \neq i$, $\|C(\alpha_i) - C(\beta_j)\|^2 = \frac{76+40\sqrt{3}}{6} = 1 + \frac{70+40\sqrt{3}}{6} = r_A^2 + r_B^2$, so by Observation 1.1 α_i and β_j are orthogonal. Otherwise, for $i \in \{0, 1, 2\}$, $\|C(\alpha_i) - C(\beta_i)\| = \sqrt{\frac{112+64\sqrt{3}}{6}} > 1 + \sqrt{\frac{70+40\sqrt{3}}{6}} = r_A + r_B$, so α_i and β_i do not intersect. ◀

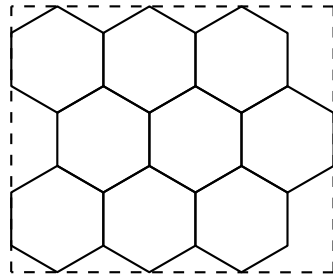
► **Theorem 3.3.** *For infinitely many values of n , there exists an n -vertex arc-RAC graph with $4.5n - O(\sqrt{n})$ edges.*

Proof. We first construct a tile and show that its drawing is indeed a valid arc-RAC drawing. Then it is easy to draw an embedded graph G with the claimed edge density.

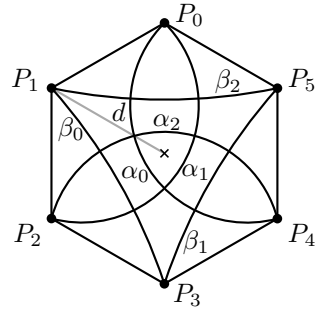
Consider two circles α and β that intersect in two points of different distance from the origin. Let $X_{\alpha\beta}^-$ be the intersection point that is closer to the origin, and let $X_{\alpha\beta}^+$ be the intersection point further from the origin.

Let the vertices of the hexagon in a tile be $P_0 = X_{\alpha_0\alpha_1}^+$, $P_1 = X_{\beta_2\beta_0}^-$, $P_2 = X_{\alpha_1\alpha_2}^+$, $P_3 = X_{\beta_0\beta_1}^-$, $P_4 = X_{\alpha_2\alpha_0}^+$, and $P_5 = X_{\beta_1\beta_2}^-$. Due to the symmetric definitions of the arcs, the angle between two consecutive vertices of the hexagon is $\pi/3$. Moreover, by a simple

21:12 Drawing Graphs with Circular Arcs and Right-Angle Crossings

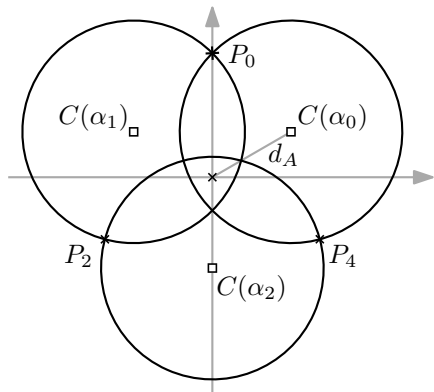


(a) The hexagonal lattice.

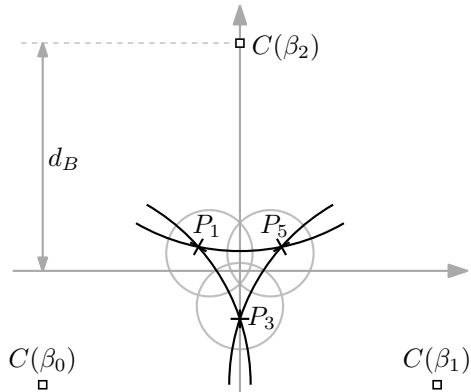


(b) A tile.

■ **Figure 10** Tiling used for the lower-bound construction.



(a) Circles covering the arcs in A .



(b) Circles covering the arcs in B .

■ **Figure 11** Construction for the lower bound on the maximum edge density of arc-RAC graphs.

computation, we see that for each $j \in \{0, 1, 2\}$ and with $d = \sqrt{1/2} + \sqrt{1/6}$ being the distance of the vertices of the hexagon from the origin, we have:

$$\begin{aligned} P_{2j} &= X_{\alpha_j \alpha_{j+1 \bmod 3}}^+ = (d \cos(\frac{\pi}{2} + j \frac{2\pi}{3}), d \sin(\frac{\pi}{2} + j \frac{2\pi}{3})) \\ P_{2j+3 \bmod 6} &= X_{\beta_j \beta_{j+1 \bmod 3}}^- = (d \cos(\frac{\pi}{6} + (j+2) \frac{2\pi}{3}), d \sin(\frac{\pi}{6} + (j+2) \frac{2\pi}{3})). \end{aligned}$$

Thus, all the vertices of the hexagon are equidistant from its center, so the hexagon is regular. According to Lemmas 3.1 and 3.2 all crossings of the arcs that belong to the same tile are orthogonal. Now we argue that the arcs in A and B are contained in the regular hexagon. To this end, we show that the arcs do not intersect the relative interior of the edges of the hexagon. To see this, take, for example, the arc α_2 , which connects P_2 and P_4 . The line segment P_2P_4 is orthogonal to the side P_1P_2 of the hexagon. As the center of α_2 is below P_2P_4 , the tangent of α_2 in P_2 enters the interior of the hexagon in P_2 . Thus, α_2 does not intersect the relative interior of the edge P_1P_2 (or of any other edge) of the hexagon. Similarly we can show that the arcs in B do not intersect the relative interior of an edge of the hexagon. Therefore, each tile is an arc-RAC drawing, and G is an arc-RAC graph.

Almost all vertices of the lattice with the exception of at most $O(\sqrt{n})$ vertices at the lattice's boundary have degree 9 [5]. Hence G has $4.5n - O(\sqrt{n})$ edges. ◀

As any n -vertex RAC graph has at most $4n - 10$ edges [12], we obtain the following.

► **Corollary 3.4.** *The arc-RAC graphs are a proper superclass of the RAC_0 graphs.*

4 Open Problems and Conjectures

An obvious open problem is to tighten the bounds on the edge density of arc-RAC graphs in Theorems 2.5 and 3.3.

Another immediate question is the relation to RAC_1 graphs, which also extend the class of RAC_0 graphs. This is especially intriguing as the best known lower bound for the maximum edge density of RAC_1 graphs is indeed larger than our lower bound for arc-RAC graphs whereas there may be arc-RAC graphs that are denser than the densest RAC_1 graphs.

The relation between RAC_k graphs and 1-planar graphs is well understood [5–8, 10, 15]. What about the relation between arc-RAC graphs and 1-planar graphs? In particular, is there a 1-planar graph which is not arc-RAC?

We are also interested in the area required by arc-RAC drawings. Are there arc-RAC graphs that need exponential area to admit an arc-RAC drawing? (A way to measure this off the grid is to consider the ratio between the longest and the shortest edge in a drawing.)

Finally, the complexity of recognizing arc-RAC graphs is open, but likely NP-hard.

References

- 1 Eyal Ackerman. On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discrete Comput. Geom.*, 41(3):365–375, 2009. doi:10.1007/s00454-009-9143-9.
- 2 Eyal Ackerman and Gábor Tardos. On the maximum number of edges in quasi-planar graphs. *J. Combin. Theory, Ser. A*, 114(3):563–571, 2007. doi:10.1016/j.jcta.2006.08.002.
- 3 Oswin Aichholzer, Wolfgang Aigner, Franz Aurenhammer, Kateřina Čech Dobiášová, Bert Jüttler, and Günter Rote. Triangulations with circular arcs. In Marc van Kreveld and Bettina Speckmann, editors, *Proc. Graph Drawing (GD'11)*, volume 7034 of *LNCS*, pages 296–307. Springer, 2012. doi:10.1007/978-3-642-25878-7_29.
- 4 Patrizio Angelini, Michael A. Bekos, Henry Förster, and Michael Kaufmann. On RAC drawings of graphs with one bend per edge. In Therese Biedl and Andreas Kerren, editors, *Proc. Graph Drawing & Network Vis. (GD'18)*, volume 11282 of *LNCS*, pages 123–136. Springer, 2018. doi:10.1007/978-3-030-04414-5_9.
- 5 Karin Arikushi, Radoslav Fulek, Balázs Keszegh, Filip Morić, and Csaba D. Tóth. Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012. doi:10.1016/j.comgeo.2011.11.008.
- 6 Christian Bachmaier, Franz J. Brandenburg, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. NIC-planar graphs. *Discrete Appl. Math.*, 232:23–40, 2017. doi:10.1016/j.dam.2017.08.015.
- 7 Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani. On RAC drawings of 1-planar graphs. *Theoretical Comput. Sci.*, 689:48–57, 2017. doi:10.1016/j.tcs.2017.05.039.
- 8 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theoret. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.
- 9 Steven Chaplick, Henry Förster, Myroslav Kryven, and Alexander Wolff. On arrangements of orthogonal circles. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *LNCS*, pages 216–229. Springer, 2019. doi:10.1007/978-3-030-35802-0_17.
- 10 Steven Chaplick, Fabian Lipp, Alexander Wolff, and Johannes Zink. Compact drawings of 1-planar graphs with right-angle crossings and few bends. *Comput. Geom.*, 84:50–68, 2019. Special issue on EuroCG 2018. doi:10.1016/j.comgeo.2019.07.006.

- 11 C. C. Cheng, Christian A. Duncan, Michael T. Goodrich, and Stephen G. Kobourov. Drawing planar graphs with circular arcs. *Discrete Comput. Geom.*, 25:405–418, 2001. doi:10.1007/s004540010080.
- 12 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theoret. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.
- 13 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.
- 14 Vida Dujmović, Joachim Gudmundsson, Pat Morin, and Thomas Wolle. Notes on large angle crossing graphs. In A. Potanin and A. Viglas, editors, *Proc. Comput. Australasian Theory Symp. (CATS'10)*, volume 109 of *CRPIT*, pages 19–24. Australian Computer Society, 2010. URL: <http://dl.acm.org/citation.cfm?id=1862317.1862320>.
- 15 Peter Eades and Giuseppe Liotta. Right angle crossing graphs and 1-planarity. *Discrete Appl. Math.*, 161(7):961–969, 2013. doi:10.1016/j.dam.2012.11.019.
- 16 Weidong Huang. Using eye tracking to investigate graph layout effects. In Seok-Hee Hong and Kwan-Liu Ma, editors, *Proc. Asia-Pacific Symp. Visual. (APVIS'07)*, pages 97–100. IEEE, 2007. doi:10.1109/APVIS.2007.329282.
- 17 Weidong Huang, Peter Eades, and Seok-Hee Hong. Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014. doi:10.1016/j.jvlc.2014.03.001.
- 18 Weidong Huang, Seok-Hee Hong, and Peter Eades. Effects of crossing angles. In *Proc. IEEE VGTC Pacific Visualization (Pacific Vis'08)*, pages 41–46, 2008. doi:10.1109/PACIFICVIS.2008.4475457.
- 19 Michael Jünger and Petra Mutzel, editors. *Graph Drawing Software*. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-642-18638-7.
- 20 Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs: Methods and Models*. Springer, Berlin, Heidelberg, 2001. doi:10.1007/3-540-44969-8.
- 21 Myroslav Kryven, Alexander Ravsky, and Alexander Wolff. Drawing graphs on few circles and few spheres. *J. Graph Algorithms Appl.*, 23(2):371–391, 2019. doi:10.7155/jgaa.00495.
- 22 Mark Lombardi and Robert Hobbs, editors. *Mark Lombardi: Global Networks*. Independent Curators, 2003.
- 23 C. Stanley Ogilvy. *Excursions in Geometry*. Oxford Univ. Press, New York, 1969.
- 24 János Pach, Radoš Radoičić, and Géza Tóth. Relaxing planarity for topological graphs. In Ervin Györi, Gyula O. H. Katona, László Lovász, and Tamás Fleiner, editors, *More Sets, Graphs and Numbers: A Salute to Vera Sós and András Hajnal*, pages 285–300. Springer Berlin Heidelberg, 2006. doi:10.1007/978-3-540-32439-3_12.
- 25 Helen C. Purchase, John Hamer, Martin Nöllenburg, and Stephen G. Kobourov. On the usability of Lombardi graph drawings. In Walter Didimo and Maurizio Patrignani, editors, *Proc. Graph Drawing (GD'12)*, volume 7704 of *LNCS*, pages 451–462. Springer, 2013. doi:10.1007/978-3-642-36763-2_40.
- 26 André Schulz. Drawing graphs with few arcs. *J. Graph Algorithms Appl.*, 19(1):393–412, 2015. doi:10.7155/jgaa.00366.
- 27 Kai Xu, Chris Rooney, Peter Passmore, and Dong-Han Ham. A user study on curved edges in graph visualisation. In Philip Cox, Beryl Plimmer, and Peter Rodgers, editors, *Proc. Theory Appl. Diagrams (DIAGRAMS'10)*, volume 7352 of *LNCS*, pages 306–308. Springer, 2012. doi:10.1007/978-3-642-31223-6_34.

Clustering Moving Entities in Euclidean Space

Stephane Durocher¹

University of Manitoba, Winnipeg, Canada
durocher@cs.umanitoba.ca

Md Yeakub Hassan

University of Manitoba, Winnipeg, Canada
hassanmy@cs.umanitoba.ca

Abstract

Clustering is a fundamental problem of spatio-temporal data analysis. Given a set \mathcal{X} of n moving entities, each of which corresponds to a sequence of τ time-stamped points in \mathbb{R}^d , a k -clustering of \mathcal{X} is a partition of \mathcal{X} into k disjoint subsets that optimizes a given objective function. In this paper, we consider two clustering problems, k -CENTER and k -MM, where the goal is to minimize the maximum value of the objective function over the duration of motion for the worst-case input \mathcal{X} . We show that both problems are NP-hard when k is an arbitrary input parameter, even when the motion is restricted to \mathbb{R} . We provide an exact algorithm for the 2-MM clustering problem in \mathbb{R}^d that runs in $O(\tau dn^2)$ time. The running time can be improved to $O(\tau n \log n)$ when the motion is restricted to \mathbb{R} . We show that the 2-CENTER clustering problem is NP-hard in \mathbb{R}^2 . Our 2-MM clustering algorithm provides a 1.15-approximate solution to the 2-CENTER clustering problem in \mathbb{R}^2 . Moreover, finding a $(1.15 - \epsilon)$ -approximate solution remains NP-hard for any $\epsilon > 0$. For both the k -MM and k -CENTER clustering problems in \mathbb{R}^d , we provide a 2-approximation algorithm that runs in $O(\tau dnk)$ time.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases trajectories, clustering, moving entities, k -CENTER, algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.22

Funding This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

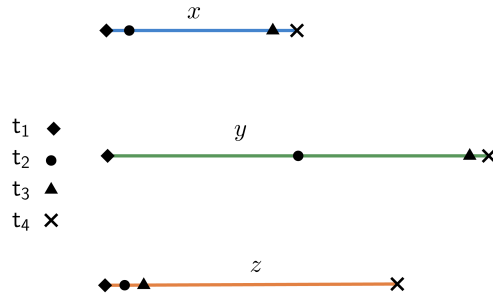
1 Introduction

The technology of tracking moving entities (e.g., humans, animals, vehicles, etc.) using GPS, motion capture, and other location-tracking devices has developed rapidly and has become widely adopted across a range of applications in the past decade. Tracking devices record the location of a moving entity over time, corresponding to a sequence of time-stamped spatial coordinates. In many cases, the positions of multiple moving entities are tracked simultaneously. In this paper, we consider a set of entities moving continuously in Euclidean space, such that at any given time, the position of each entity is represented by a point in \mathbb{R}^d . Several applications (e.g., animal migration analysis, weather forecasting, and component classification from motion capture data) require clustering moving entities [16]. The goal of clustering is to partition a given set of objects into groups of similar objects, where the degree of similarity, i.e., the quality of the clustering, is measured according to a given objective function. Some previous work on clustering moving entities has examined the problem of updating clusters over time to maintain a good clustering [7, 9, 14, 16, 18]. In these types of clustering, an entity is often required to switch from one cluster to another to maintain the

¹ This research was completed while Stephane Durocher was a visiting researcher with the Morpheo research group at INRIA Grenoble Rhône Alpes in Grenoble, France.



22:2 Clustering Moving Entities in Euclidean Space



■ **Figure 1** Trajectories of three entities x, y, z in \mathbb{R}^2 within time stamps t_1 to t_4 .

optimality of the objective function. These types of clustering are not suitable for applications that require an entity to remain in a cluster while optimizing a global objective over the entire motion. For example, consider an input consisting of trajectories of a set of sensors recorded during the motion capture of a person running, for which the goal is to identify different body parts during the motion, e.g., torso, arms, legs, etc. If we allow entities (in this case, sensors) to change clusters over time, then one part of the body (one trajectory) might be classified into one cluster (say an arm) at one time and another cluster (say a leg) later during the same motion. To prevent such inconsistencies, we examine the problem of clustering a set of moving entities where the moving entities cannot switch clusters over time. In particular, the basic unit of clustering is the whole trajectory of each moving entity.

The discrete sequence of time-stamped locations of an entity can be linearly interpolated to obtain a continuous piecewise-linear curve. Depending on the application, we may need to cluster just the curves (e.g., when clustering animal migration routes followed by multiple herds or flocks of animals, possibly at different times and speeds) or the curves along with their associated time stamps (e.g., when clustering animals into groups that moved together during migration). The k -CENTER clustering of trajectories of moving entities has been studied under the Fréchet distance metric [5, 6]. This type of clustering considers only the piecewise-linear curves of each entity, without factoring the time at which moving entities occupied points on their trajectory. Fréchet distance considers all possible walk sequences between two trajectories. Therefore, input consisting of position as a function of time leads to different clusters than when the input consists of trajectories only. For example, consider three entities x, y and z moving in \mathbb{R}^2 whose trajectories from time t_1 to t_4 are given in Figure 1. The Fréchet distance between the trajectories of x and y is greater than the Fréchet distance between the trajectories of y and z . Therefore, if we partition $\{x, y, z\}$ into two clusters to minimize the maximum intra-cluster Fréchet distance, then y and z would be in one cluster, and x would be in another cluster. However, the Euclidean distance between y and z at time t_3 is greater than the distance between x and y at any time. When time is considered, the resulting clustering assigns y and z to different clusters. As a result, when time is important, clustering should optimize the objective function based on the position of entities over the entire motion, i.e., at all times during the motion. In this paper, we examine clustering algorithms for moving entities in Euclidean space where each entity's trajectory has associated time stamps. In particular, we consider k -MM and k -CENTER clustering of moving entities, where k specifies the number of clusters.

Problem description

We are given a set \mathcal{X} of n entities moving in \mathbb{R}^d that are tracked over a given time interval. The spatial coordinates of entities are recorded at a set T of τ discrete time stamps. Therefore, the location of each entity over time traces a piecewise-linear trajectory in \mathbb{R}^d consisting of τ vertices. At any time $t \in T$, the position of each entity $x_i \in \mathcal{X}$ is a fixed point in \mathbb{R}^d , denoted $x_i(t)$. For any time $t \in T$, $\mathcal{X}(t) = \{x(t) \mid x \in \mathcal{X}\}$ denotes the multiset of points corresponding to the positions of the entities in \mathcal{X} at time t . Similarly, for a cluster $C_i \subseteq \mathcal{X}$, $C_i(t)$ denotes the multiset of points corresponding to the positions of entities in C_i at time t .

► **Definition 1.** *Given a set T of τ discrete time stamps and a finite set \mathcal{X} of n moving entities in \mathbb{R}^d , the goal of the **k -clustering** problem is to partition \mathcal{X} into k disjoint sets C_1, C_2, \dots, C_k to optimize a given objective function $f : \{C_1, C_2, \dots, C_k\} \rightarrow \mathbb{R}$. The goal is to minimize the maximum value of $f(t)$ over all $t \in T$ for the worst-case input \mathcal{X} .*

Kinetic clustering refers to clustering problems on moving or dynamic entities for which entities may switch clusters over time; *static clustering* refers to clustering problems on moving or dynamic entities for which each entity is assigned to a single cluster. Note that static facility location does not imply static entities, i.e., entity positions can change over time in static clustering, but their assignment to clusters cannot. This paper examines the static setting for the problem of clustering moving entities.

The particular objective function depends on the applications of the problem. Common clustering problems are k -CENTER, k -median, k -MM and k -means clustering. In this paper, we present the results for the k -MM and k -CENTER clustering of moving entities in \mathbb{R}^d . In the k -MM clustering problem, the objective function f measures the maximum distance during the motion between any two entities in any cluster C_i . The “MM” in the objective function refers to “max max”. The objective of the k -MM clustering is to find a partition C_1, \dots, C_k of \mathcal{X} that minimizes

$$\max_{i \in \{1, \dots, k\}} \max_{\{x, y\} \subseteq C_i} \max_{t \in T} \delta(x(t), y(t)), \quad (1)$$

where δ is the Euclidean (ℓ_2) distance metric. For any set of points in \mathbb{R}^2 , the k -MM clustering is often denoted k -2MM, where “2” refers to Euclidean (ℓ_2) distance. The k -2MM problem is NP-hard in \mathbb{R}^2 when k is an arbitrary input parameter [12]. In the k -CENTER clustering problem, the objective function f measures the maximum distance during the motion from any $x \in \mathcal{X}$ to its cluster center, where the center of each cluster C_i at time t is the center of the smallest enclosing d -ball of $C_i(t)$. Therefore, the objective is to find a partition C_1, \dots, C_k of \mathcal{X} that minimizes

$$\max_{i \in \{1, \dots, k\}} \max_{x \in C_i} \max_{t \in T} \delta(x(t), m_i(t)), \quad (2)$$

where $m_i(t)$ denotes the center of the smallest enclosing d -ball of $C_i(t)$. That minimum value of Equation 2 corresponds to the smallest radius r such that each cluster can be covered at all times by a d -ball of radius r . We consider the *continuous* k -CENTER clustering problem, where the center of a cluster can be placed anywhere in Euclidean space (i.e., the cluster center is not restricted to $\mathcal{X}(t)$, as is the case in discrete facility location). The k -CENTER clustering problem is also NP-hard in \mathbb{R}^2 when k is an arbitrary input parameter [19].

Results and organization.

In this paper, we study k -MM and k -CENTER clustering of moving entities in \mathbb{R}^d . In Section 2, we discuss related work on clustering moving entities. In Section 3, we present algorithms

that solve the 2-MM clustering problem optimally in $O(\tau dn^2)$ time in \mathbb{R}^d , and in $O(\tau n \log n)$ time in \mathbb{R} . In Section 4, we show that 2-CENTER clustering is NP-hard for entities moving in \mathbb{R}^2 ; this differs from the 2-CENTER problem on a set of points in \mathbb{R}^d , which can be solved in polynomial time [2]. The 2-MM clustering algorithm gives an 1.15-approximate solution for the 2-CENTER clustering problem in \mathbb{R}^2 . Moreover, computing a $(1.15 - \epsilon)$ -approximation remains NP-hard for any $\epsilon > 0$. In Section 5, we show that the k -MM and k -CENTER clustering problems are NP-hard when k is an arbitrary input parameter, even in \mathbb{R} ; again, this differs from the k -MM and k -CENTER problems on a set of points, which can be solved in polynomial time in \mathbb{R} [11]. We provide a 2-approximation algorithm for both problems by using the greedy technique of Gonzalez’s algorithm [12]. Finally, in Section 6, we conclude with a discussion of possible directions for future research.

2 Related work

The k -CENTER and k -MM clustering problems have been studied extensively for sets of points in \mathbb{R}^d [1, 12]. Both problems are NP-hard in \mathbb{R}^2 when k is an arbitrary input parameter [12, 19]. Gonzalez [12] provides a 2-approximation algorithm for both k -MM and k -CENTER clustering of a set of points in \mathbb{R}^2 which requires $O(nk)$ time. However, the algorithm works for any metric space. Frederickson [11] gives an algorithm for the k -CENTER problem on a tree that uses $O(n)$ time and space. The algorithm allows centers to be located at any points on the edges. Therefore, the k -CENTER problem for any set of points in \mathbb{R} can be solved in $O(n)$ time using Frederickson’s algorithm. The 2-CENTER clustering of a set of points in \mathbb{R}^2 can be solved exactly in $O(n \log^2 n)$ time [10, 21].

Har-Peled [13] examines the k -CENTER problem for sets of moving entities in \mathbb{R}^d , giving an algorithm that partitions the input into $k^{\mu+1}$ clusters for entities whose motion has algebraic degree μ . The 1-CENTER, 2-CENTER, and k -CENTER clustering problems have been studied for sets of moving entities in the kinetic setting, where entities may switch clusters, with the goal to maintain a good approximation of the objective function while bounding the relative velocity of the cluster centers [3, 7, 9, 8]. Variants of these problems have been studied, such as k -clustering to minimize the average value of the objective function throughout the motion (e.g., [14]). Li et al. [17] examine a movement pattern of moving entities, called swarm, where a group of entities moves together for a set of (possibly nonconsecutive) timestamps.

Previous work also examines clustering trajectories of moving entities under different similarity measures: Fréchet distance, dynamic time warping, and edit distance [22]. Lee et al. [15] propose a two-phase algorithm to cluster similar portions of a set of trajectories. Buchin et al. [4] provide a trajectory grouping framework where the structure of a group of similar trajectories is represented by a Reeb graph. Driemel et al. [6] introduce the (k, l) -CENTER clustering problem for a set of curves in \mathbb{R} under the Fréchet distance metric, where each of the k -CENTER curves has complexity at most l . A $(1 + \epsilon)$ -approximation algorithm is proposed for this problem, which runs in near-linear time. Buchin et al. [5] show that (k, l) -CENTER clustering under Fréchet distance is NP-hard even if $k = 1$.

3 Algorithms for 2-MM clustering

Given a set \mathcal{X} of n moving entities in \mathbb{R}^d , we can construct a weighted undirected complete graph G , where the vertices of G are the moving entities in $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ and the weight of the edge between two vertices x_i and x_j is

$$\max_{t \in T} \delta(x_i(t), x_j(t)). \quad (3)$$

► **Lemma 2.** *The weights of graph G satisfies the triangular inequality.*

Proof. Let x, y , and z be three moving entities in \mathbb{R}^d . At any time t , the positions of these entities satisfy triangle inequality (i.e., $\delta(x(t), y(t)) + \delta(y(t), z(t)) \geq \delta(z(t), x(t))$). For this instance, the graph G has three vertices $\{x, y, z\}$ and three edges $\{e_1(x, y), e_2(y, z), e_3(z, x)\}$. The weights of the edges are

$$e_1 = \delta(x(t_1), y(t_1)); \quad e_2 = \delta(y(t_2), z(t_2)); \quad e_3 = \delta(z(t_3), x(t_3));$$

where, $t_1 = \arg \max_t \delta(x(t), y(t))$; $t_2 = \arg \max_t \delta(y(t), z(t))$; $t_3 = \arg \max_t \delta(z(t), x(t))$. From the weights of the edges we get,

$$\begin{aligned} e_1 + e_2 &= \delta(x(t_1), y(t_1)) + \delta(y(t_2), z(t_2)) \\ &\geq \delta(x(t_3), y(t_3)) + \delta(y(t_3), z(t_3)); \quad (\text{from the definition of } t_1, t_2 \text{ and } t_3) \\ &\geq \delta(x(t_3), z(t_3)); \quad (\text{since } x, y, z \text{ satisfy the triangle inequality at time } t_3) \\ &\geq e_3 \end{aligned} \quad \blacktriangleleft$$

We can solve the 2-MM clustering of \mathcal{X} by partitioning G into two clusters C_1, C_2 such that the partitioning minimizes

$$\max_{i \in \{1, 2\}} \max_{\{x, y\} \subseteq C_i, x \neq y} w(x, y), \quad (4)$$

where $w(x, y)$ is the weight of the edge between x and y . In other words, we want to minimize the maximum intra-cluster edge weight, where an edge having both end vertices in the same cluster is called an intra-cluster edge. To partition the graph G , we first compute a tree T that is a Maximum Spanning Tree of G using Prim's algorithm. Proceed to 2-color T by alternatively coloring vertices red and blue in a breadth-first search. This coloring defines the two clusters, C_1 and C_2 .

► **Theorem 3.** *C_1 and C_2 define an optimal 2-MM clustering.*

Proof. The algorithm constructs a tree T , i.e., a bipartite graph whose vertices are divided into two disjoint and independent sets C_1 and C_2 . Suppose x and y are two vertices in C_1 (or C_2), such that the edge (x, y) in G is a maximum-weight intra-cluster edge (outcome of Equation 4). Since T is bipartite, there exists a path from x to y with an even number of edges. Adding the edge (x, y) to T would create a cycle in T . Since T is a maximum spanning tree of G , each edge in that cycle has a weight at least $w(x, y)$. As a result, there exists at least one vertex z_1 in C_2 (or C_1) having $w(x, z_1) \geq w(x, y)$, and there exists at least one vertex z_2 in C_2 (or C_1) having $w(y, z_2) \geq w(x, y)$. If there exists an edge between (x, z_2) in T , then $w(x, z_2) \geq w(x, y)$ (since x, z_2, y creates a cycle in T). If there is no edge between (x, z_2) in T , then adding the edge (x, z_2) to T would create a cycle where $w(x, z_2) \geq w(y, z_2)$. Since $w(y, z_2) \geq w(x, y)$, we can say that $w(x, z_2) \geq w(x, y)$. Similarly, we can say that $w(y, z_1) \geq w(x, y)$.

In every possible 2-clustering of G where x and y are in different cluster, there is an intra-cluster edge that has weight at least $w(x, y)$ (because z_1 or z_2 can be in any of the two clusters). On the other hand, when x and y are in the same cluster, every possible 2-clustering of G has an intra-cluster edge having weight $w(x, y)$. This proves that, in every possible 2-clustering of G , there exists an intra-cluster edge which has weight at least $w(x, y)$. Therefore, G cannot be partitioned into two clusters where the maximum intra-cluster edge has weight less than $w(x, y)$. Hence, the clustering C_1, C_2 is optimal. \blacktriangleleft

We have n moving entities in \mathbb{R}^d , and the pairwise distance of all pair of entities using Equation 3 can be computed in $O(\tau dn^2)$ time. Thus, constructing the graph G requires $O(\tau dn^2)$ time. The running time of Prim's algorithm is $O(n^2)$. Since, the tree T has $n - 1$ edges, the breadth-first search in the 2-coloring algorithm requires $\Theta(n)$ time. Therefore, the running time of the 2-MM clustering algorithm is $O(\tau dn^2)$.

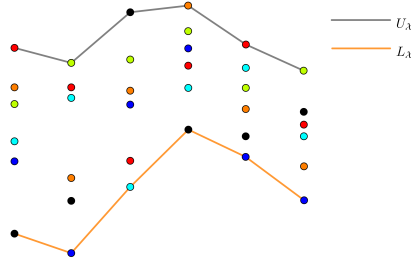
► **Theorem 4.** *The 2-MM clustering of moving entities in \mathbb{R}^d can be computed in $O(\tau dn^2)$ time.*

3.1 Improved algorithm for entities in \mathbb{R}

We consider a set \mathcal{X} of n moving entities in \mathbb{R} . We plot the entities' position functions over time, such that horizontal axis represents time and the vertical axis represents the positions of the entities.

► **Definition 5.** *The **upper trajectory** of \mathcal{X} , denoted $U_{\mathcal{X}}$, is a piecewise-linear trajectory corresponding to the upper envelope of the plots of trajectories of entities in \mathcal{X} . That is, at any time $t \in T$, the position of the upper trajectory $U_{\mathcal{X}}(t)$ is $\max\{x(t) | x \in \mathcal{X}\}$. The **lower trajectory** of \mathcal{X} , denoted $L_{\mathcal{X}}$, is defined analogously with respect to the lower envelope of the plots of the trajectories of entities in \mathcal{X} .*

To compute $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$, at each time t , we need to find the maximum and minimum value of $\mathcal{X}(t)$. Therefore, $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ can be computed in $\Theta(\tau n)$ time. Examples of the upper and lower trajectories of six entities are shown in Figure 2.



■ **Figure 2** Upper and Lower trajectories of six entities moving in \mathbb{R} .

► **Observation 6.** *The upper and lower trajectories of \mathcal{X} can be computed in $\Theta(\tau n)$ time.*

We can find the furthest pair of entities in \mathcal{X} by using $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$. A pair $(x, y) \in \mathcal{X}$ is called the furthest pair if

$$\max_{t \in T} \delta(x(t), y(t)) \geq \max_{\substack{\{x', y'\} \subseteq \mathcal{X}, \\ x' \neq x, y' \neq y}} \max_{t \in T} \delta(x'(t), y'(t)).$$

At each time stamp t , we calculate the distance between entities $x(t) \in U_{\mathcal{X}}(t)$ and $y(t) \in L_{\mathcal{X}}(t)$. The furthest pair is the one which has maximum distance over all time.

► **Observation 7.** *Given the upper and lower trajectories of \mathcal{X} , the furthest pair of entities in \mathcal{X} can be found in $\Theta(\tau)$ time.*

At every time stamp, we consider maintaining the sorted order of the entities where they are sorted according to their positions. This takes $O(\tau n \log n)$ time. If an entity is added to or removed from \mathcal{X} , then $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ need to be updated as well. Consider a new entity x that is added to \mathcal{X} . To maintain the sorted order, the insertion of x would take $O(\tau \log n)$ time by using a balanced search tree. After that, we can update the upper and lower trajectories of \mathcal{X} in $\Theta(\tau)$ time. For each time t , if $x(t) \geq U_{\mathcal{X}}(t)$, then we update the value of $U_{\mathcal{X}}(t)$ to $x(t)$. Similarly, if $x(t) \leq L_{\mathcal{X}}(t)$, then we update the value of $L_{\mathcal{X}}(t)$ to $x(t)$. If an entity x is deleted from \mathcal{X} , we can update the sorted order of \mathcal{X} in $O(\tau)$ time. $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ can also be updated in $\Theta(\tau)$ time. For each time t , if $x(t) = U_{\mathcal{X}}(t)$, we update the value of $U_{\mathcal{X}}(t)$ to $y(t)$, where $y \in \mathcal{X}$ and $y(t)$ is the closest neighbor of $x(t)$. Similarly, we can also update the lower trajectory of \mathcal{X} .

Consider two sets \mathcal{X}_1 and \mathcal{X}_2 of moving entities in \mathbb{R} . We want to find an entity $x \in \mathcal{X}_1$ that is furthest away from all entities in \mathcal{X}_2 . Therefore, $x \in \mathcal{X}_1$ is the entity that maximizes

$$\max_{y \in \mathcal{X}_2} \max_{t \in T} \delta(x(t), y(t)).$$

After computing the upper and lower trajectories of \mathcal{X}_1 and \mathcal{X}_2 , the furthest entity can be found in $\Theta(\tau)$ time. At each time t , we need to find the furthest entity $x(t) \in \mathcal{X}_1(t)$ by calculating the maximum of $\{\delta(U_{\mathcal{X}_1}(t), U_{\mathcal{X}_2}(t)), \delta(U_{\mathcal{X}_1}(t), L_{\mathcal{X}_2}(t)), \delta(L_{\mathcal{X}_1}(t), U_{\mathcal{X}_2}(t)), \delta(L_{\mathcal{X}_1}(t), L_{\mathcal{X}_2}(t))\}$.

► **Observation 8.** *Given two sets \mathcal{X}_1 and \mathcal{X}_2 of moving entities in \mathbb{R} and their upper and lower trajectories, the furthest entity $x \in \mathcal{X}_1$ from all entities in \mathcal{X}_2 can be computed in $\Theta(\tau)$ time.*

Unlike the 2-MM clustering algorithm in \mathbb{R}^d , we do not need to construct the graph G for the 2-MM clustering in \mathbb{R} . In this case, we construct a graph G' , which is equivalent to the maximum spanning tree of G . We use the idea of Prim's algorithm to construct G' . At every time stamp, we maintain the sorted order of the entities. We then use the following approach to find the 2-MM clustering of \mathcal{X} :

1. Calculate $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$.
2. Find the furthest pair (x_1, x_2) of entities in \mathcal{X} .
3. Let G' be a graph with two vertices (entities) x_1, x_2 .
4. Add an edge between x_1 and x_2 .
5. Let $\mathcal{Y} = \{x_1, x_2\}$. Remove x_1 and x_2 from \mathcal{X} .
6. Update $U_{\mathcal{X}}, L_{\mathcal{X}}, U_{\mathcal{Y}}$ and $L_{\mathcal{Y}}$.
7. For $i = 1$ to $n - 2$:
 - Find an entity $x \in \mathcal{X}$ that is furthest away from all entities in \mathcal{Y} . Let $y \in \mathcal{Y}$ be the entity for which x becomes the furthest from \mathcal{Y} .
 - Add x to \mathcal{Y} and remove x from \mathcal{X} . Then, update $U_{\mathcal{X}}, L_{\mathcal{X}}, U_{\mathcal{Y}}$ and $L_{\mathcal{Y}}$.
 - Add a vertex x to graph G' . Since, y is already added to G' , add an edge between y and x .
8. The graph G' is a bipartite graph; run the 2-coloring algorithm in graph G' .

Each vertex of G' represents a moving entity in \mathbb{R} . The algorithm initially adds the furthest pair of entities in G' and grows G' by one edge in each iteration. Similar to Prim's algorithm, in each iteration, the algorithm finds an entity that is furthest away from all entities (vertices) in G' , and then adds that entity to G' . Therefore the graph G' is equivalent to the maximum spanning tree of the graph G . After constructing G' , we apply the 2-coloring algorithm (similar to the 2-MM algorithm in \mathbb{R}^d) to find the clusters C_1 and C_2 .

► **Theorem 9.** *The 2-MM clustering of moving entities in \mathbb{R} can be computed in $O(\tau n \log n)$ time.*

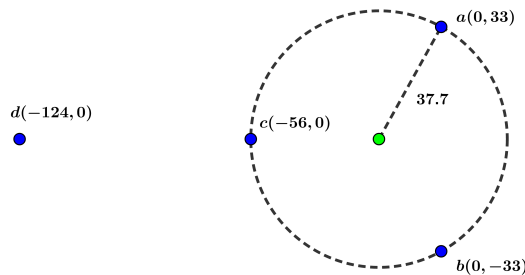
Proof. The sorting operations in all time stamps take total $O(\tau n \log n)$ time. Initially, we compute the upper and lower trajectories in $\Theta(\tau n)$ time. In each iteration, we find the furthest entity from \mathcal{X} , and then update the upper and lower trajectories of \mathcal{X} and \mathcal{Y} . Since we are maintaining the sorted order of entities in all time stamps, the insert operation in \mathcal{Y} would take $O(\tau \log n)$ time. The upper and lower trajectories of \mathcal{X} and \mathcal{Y} are updated in $\Theta(\tau)$ time. Therefore, constructing G' requires $O(\tau n \log n)$ time. The 2-coloring algorithm requires $\Theta(n)$ time. Thus, the overall running time of the algorithm is $O(\tau n \log n)$. ◀

4 Hardness results of 2-Center clustering

Given any set P of points in \mathbb{R} , the diameter of the smallest enclosing circle of P and the maximum possible distance between any two points in P are equal. Therefore, the k -CENTER and k -MM clustering of P are equivalent. This result also holds for the k -CENTER and k -MM clustering of a set of moving entities in \mathbb{R} . However, the two problems have different solutions in general when entities move in higher dimensions (see Example 11 below).

► **Observation 10.** *If \mathcal{X} is a set of moving entities in \mathbb{R} , then the k -CENTER and the k -MM clustering of \mathcal{X} are equivalent.*

► **Example 11.** Consider four points $a(0, 33)$, $b(0, -33)$, $c(-56, 0)$, $d(-124, 0)$ in \mathbb{R}^2 (see Figure 3). In exact 2-CENTER clustering, the radius of any cluster would be at most 34 by keeping $\{a, b\}$ in one cluster and $\{c, d\}$ in another cluster. Because the smallest enclosing circle containing any three points among $\{a, b, c, d\}$ would have radius at least 37.7 (the radius of circle passing through $\{a, b, c\}$). However, the exact 2-MM clustering would keep $\{a, b, c\}$ in one cluster, leaving $\{d\}$ for the other cluster, since the maximum distance between any two points in $\{a, b, c\}$ is at most 66.



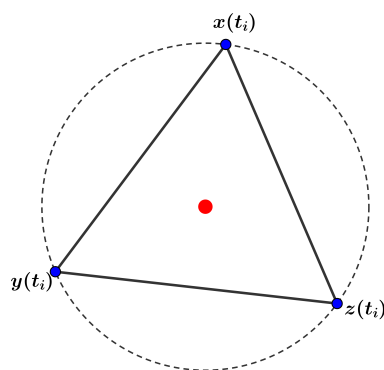
■ **Figure 3** Illustration in support of Example 11.

As we now show, computing the 2-CENTER clustering of a set \mathcal{X} of n moving entities in \mathbb{R}^2 is NP-hard. This differs from the problem of computing a 2-CENTER of a set of points in \mathbb{R}^2 which can be solved in $O(n \log^2 n)$ time [21]. Given a set \mathcal{X} of n moving entities and a real parameter r^* , the decision version of the 2-CENTER clustering problem asks to determine whether \mathcal{X} can be partitioned into two sets C_1 and C_2 , such that each cluster can be covered by a disc of radius r^* at all times. We prove that the decision version of 2-CENTER clustering in \mathbb{R}^2 is NP-complete. We obtain the result by reduction from the Monotone Not-All-Equal

3SAT (NAE-3SAT) problem. The Monotone NAE-3SAT is one of many variants of the 3SAT problem, which is NP-Complete [20]. The Monotone NAE-3SAT problem consists of a set of Boolean variables and a set of monotone clauses, i.e., variables in each clause are never negated. Unlike the 3SAT problem, each clause of the NAE3SAT problem requires to have at least one true and one false Boolean value.

► **Theorem 12.** *The decision version of the 2-CENTER clustering problem in \mathbb{R}^2 is NP-complete.*

Proof. Choose any instance of the Monotone NAE-3SAT problem, let \mathcal{S} denote its set of Boolean variables, and let $n = |\mathcal{S}|$. We describe how to construct a corresponding instance of the decision version of the 2-CENTER clustering problem with $r^* = 1$. Each element in \mathcal{S} is mapped to a corresponding moving entity in \mathbb{R}^2 , and the number of time stamps is equal to the number of clauses.



■ **Figure 4** Blue points represent the positions of entities x, y, z at time t_i . All entities other than x, y , and z are placed at the red point. The circumradius of the triangle is $1 + \epsilon$, for some small $\epsilon > 0$.

At each time t_i , we take the i^{th} clause from the instance of the Monotone NAE-3SAT problem and assign coordinates to the moving entities in the plane as follows. Let $x, y, z \in \mathcal{S}$ be the three entities from the clause and let $x(t_i), y(t_i), z(t_i)$ denote their respective positions at time t_i . We place $x(t_i), y(t_i), z(t_i)$ at the vertices of an equilateral triangle such that the circumradius of the triangle is slightly larger than one; thus, x, y , and z cannot be covered by a unit-radius disc at time t_i . The remaining $n - 3$ entities are placed at the circumcenter of the triangle. This transformation can be done in $O(n)$ time. Each cluster in the partitioning of \mathcal{S} can be covered by a unit-radius disc if and only if the three entities x, y, z are not in the same cluster. In the instance of the Monotone NAE-3SAT problem, the values of three variables in each clause cannot be equal to each other. Similarly, the corresponding three entities from each clause cannot be in the same cluster if the instance is a “yes” instance of the decision version of the 2-CENTER clustering problem. Therefore, the instance of the Monotone NAE-3SAT problem is satisfiable if and only if the corresponding instance of the decision version of 2-CENTER clustering problem is a “yes” instance. ◀

We can achieve a 1.15-approximate solution for the 2-CENTER clustering of moving entities in \mathbb{R}^2 by applying the 2-MM clustering algorithm from Section 3. We also prove that no polynomial-time algorithm can achieve a better approximation ratio. In what follows, we prove the approximation ratio and its lower bound.

► **Theorem 13.** *The exact 2-MM clustering of moving entities in \mathbb{R}^2 gives a 1.15-approximate solution for the 2-CENTER clustering problem.*

Proof. We partition \mathcal{X} into two clusters C_1 and C_2 by applying the algorithm described in Section 3. Let l be the maximum distance over all time between any pair of entities from the same cluster (C_1 or C_2). As argued in the proof of Theorem 3, in every possible 2-clustering of \mathcal{X} , there exists a pair of entities in some cluster whose maximum distance over all time is at least l . By Jung's theorem, if the largest possible distance between two points from a finite set of points in the plane is l , then there exists a circle enclosing all these points with a radius no greater than $l/\sqrt{3}$. Hence, in any 2-MM clustering of \mathcal{X} , the maximum radius of any cluster would be at most $l/\sqrt{3}$. Since every possible 2-clustering of \mathcal{X} consists a pair of entities in any cluster whose maximum distance over all time is at least l , the maximum radius of any cluster in exact 2-CENTER clustering of \mathcal{X} would be at least $l/2$. Thus, the exact 2-MM clustering gives a $2/\sqrt{3}$ -approximation (or 1.15-approximation) for the 2-CENTER clustering problem. ◀

► **Theorem 14.** *If $P \neq NP$, no polynomial-time algorithm can achieve a $(1.15 - \epsilon)$ -approximate solution for the 2-CENTER clustering of moving entities in \mathbb{R}^2 for any $\epsilon > 0$.*

Proof. Choose any instance of the Monotone NAE-3SAT problem, let \mathcal{S} denote its set of Boolean variables, and let $n = |\mathcal{S}|$. We show how to construct a corresponding instance of the 2-CENTER problem in \mathbb{R}^2 . Each Boolean variable is mapped to a moving entity in \mathbb{R}^2 . At each time t_i , the three entities in the i^{th} clause from the instance of the Monotone NAE-3SAT problem are placed at the vertices of an equilateral triangle where the length of each side of the triangle is 2. The remaining entities are placed at the circumcenter of the triangle. If we can partition \mathcal{S} into two clusters $\{C_1, C_2\}$ such that all three entities in each clause are split by this partition, then the maximum radius of any partition over all time would be 1. For any $\epsilon > 0$, a $(1.15 - \epsilon)$ -approximation algorithm for the 2-CENTER clustering of \mathcal{S} will give us two clusters whose maximum radius over all time would be less than 1.15, in this case, it has to be 1.

We consider the value of a Boolean variable in \mathcal{S} is true if the corresponding entity is in cluster C_1 in the 2-CENTER clustering of \mathcal{S} . Similarly, the value of a variable is false if the corresponding entity is in cluster C_2 . The instance of the Monotone NAE-3SAT problem is satisfiable if and only if all the entities in each clause are not in the same cluster. This can only be done by a $(1.15 - \epsilon)$ -approximation algorithm for 2-CENTER clustering of \mathcal{S} . Therefore, a polynomial-time $(1.15 - \epsilon)$ -approximation algorithm for 2-CENTER clustering of \mathcal{S} can be used to decide the satisfiability of the corresponding instance of the Monotone NAE-3SAT problem. This cannot happen assuming $P \neq NP$. ◀

5 k -MM and k -Center clustering

The k -MM and k -CENTER clustering problems are solvable in polynomial-time for any set of fixed points in \mathbb{R} , i.e., without motion [7]. As we now show, both problems are NP-hard for entities moving in \mathbb{R} . We reduce an instance of a restricted version of the k -gMM clustering problem to an instance of the k -MM clustering problem for moving entities in \mathbb{R} . k -gMM is the more general version of the geometric k -MM problem on arbitrarily weighted graphs; any instance of the k -MM problem directly corresponds to an instance of the k -gMM problem.

Given an arbitrary weighted graph $G = (V, E)$, the goal of the k -gMM clustering problem is to partition V into k disjoint sets (k clusters) such that the maximum weight of any intra-cluster edge is minimized. The decision version of the k -gMM problem has been proved to be NP-complete [12]. The result is obtained by a reduction from a restricted version of the exact cover by 3-sets problem. Given an instance of the restricted version of the exact

cover by 3-sets problem, Gonzalez constructed a weighted complete graph as an instance of the decision version of k -gMM problem, where the weight of each edge is either one or two. Therefore, the decision version of k -gMM problem remains NP-complete even when the input graph is a weighted complete graph with edge weights either one or two. We call this problem the restricted k -gMM problem which we use in our reduction.

► **Theorem 15.** *Given a set of moving entities in \mathbb{R}^d , for any $d \geq 1$, the k -MM and k -CENTER clustering problems are NP-hard when k is an arbitrary input parameter.*

Proof. We consider an instance of the restricted k -gMM clustering problem, where we are given a weighted complete graph G with edge weights in $\{1, 2\}$. We construct a corresponding instance of the k -MM clustering problem. Each vertex of G is mapped to a moving entity in \mathbb{R} . The number of time stamps and the number of edges in G are equal. For each time t_i , we take the i th edge e_i from G and assign coordinates to its two end vertices x, y (two entities) on the real line such that the Euclidean distance between them is equal to the weight of that edge. At that time, t_i , the remaining entities are positioned at the midpoint of the line segment between x and y . Since the minimum and maximum weight of an edge in G is one and two, respectively, the maximum distance between any two entities over all time is equal to the weight of the longest edge connecting two entities (vertices) in G . Therefore, we can achieve the optimal solution of the restricted k -gMM clustering problem for graph G if and only if we achieve the optimal solution of the k -MM clustering problem for the corresponding transformed instance. Since the k -MM and k -CENTER clustering problems are equivalent in \mathbb{R} (Observation 10), this hardness result also holds for the k -CENTER problem as well. ◀

Gonzalez [12] provides a 2-approximation algorithm for the k -MM and k -CENTER clustering of fixed points in \mathbb{R}^d . We provide a modified version of Gonzalez's algorithm to compute an approximate solution for the k -MM and k -CENTER clustering of a set \mathcal{X} of n moving entities in \mathbb{R}^d . The algorithm initially assigns all entities to cluster C_1 and arbitrarily labels one entity as the head of that cluster, denoted $head_1$. The remaining clusters are computed in $k - 1$ iterations. In the i^{th} iteration, we compute the head of cluster C_i . An entity $x \in \mathcal{X}$ is considered as the head of the cluster C_i if it maximizes

$$\max_{j \in \{1, \dots, i\}} \max_{x \in C_j} \max_{t \in T} \delta(head_j(t), x(t)).$$

We label x as $head_i$ and add it to cluster C_i . After computing the i^{th} head, the entities are assigned to a cluster such that, for each entity $y \in \{C_1 \cup \dots \cup C_{i-1}\}$, we move y to C_i if the maximum distance between y and its current cluster head over all time is larger than the maximum distance between y and $head_i$. The i^{th} iteration of choosing the i^{th} head takes $O(\tau n)$ time. Therefore, the total running time of the algorithm is $O(\tau nk)$. Since the algorithm makes its decision by comparing maximum distances between pairs of entities over all time, it can be generalized for moving entities in \mathbb{R}^d , where computing the Euclidean distance between two points takes $O(d)$ time. If d is an input parameter, then the algorithm runs in $O(\tau d nk)$ time. The algorithm guarantees a 2-approximation for both the k -MM and k -CENTER clustering problems. The objective function value of k -MM and k -CENTER clustering of \mathcal{X} by using the above algorithm is denoted by $\phi_{P_1}(\mathcal{X})$ and $\phi_{P_2}(\mathcal{X})$ respectively. Let $OPT_{P_1}(\mathcal{X})$ and $OPT_{P_2}(\mathcal{X})$ be the objective function values of optimal k -MM and k -CENTER clustering of \mathcal{X} respectively.

► **Lemma 16.** *Given a set \mathcal{X} of n moving entities in \mathbb{R}^d , where $\mathcal{L} = \{x_1, \dots, x_{k+1}\}$ is a subset of \mathcal{X} , for every pair $\{x, y\} \subseteq \mathcal{L}$, if $\max_{t \in T} \delta(x(t), y(t)) \geq h$, then $OPT_{P_1}(\mathcal{X}) \geq h$, for any $h > 0$.*

Proof. We have $k + 1$ entities to make k clusters. By the pigeonhole principle, one cluster must contain two entities, and the rest of the $k - 1$ clusters contain one entity in each. The maximum distance between any two entities in \mathcal{X} over all time is at least h , for any $h > 0$. We have only one cluster with two entities where the maximum distance between them over all time is at least h , therefore we can say that $OPT_{P_1}(\mathcal{X}) \geq h$. ◀

► **Theorem 17.** *The modified Gonzalez algorithm is a 2-approximation algorithm for the k -MM and k -CENTER clustering of \mathcal{X} .*

Proof. Let x denote an entity in cluster C_i that maximizes

$$\max_{i \in \{1, \dots, k\}} \max_{x \in C_i} \max_{t \in T} \delta(\text{head}_i(t), x(t)).$$

Let h be the maximum distance between x and head_i over all time. This distance satisfies triangle inequality (Lemma 2). Therefore, $\phi_{P_1}(\mathcal{X}) \leq 2 \cdot h$. In every iteration, the maximum distance between x and head_i over all time is at least h . Therefore, when a new cluster is added, the maximum distance between the new cluster's head and any previously added cluster's head over all time is at least h . That is to say, for any $i \neq j$, $\max_{t \in T} \delta(\text{head}_i(t), \text{head}_j(t)) \geq h$. Let $\mathcal{L} = \{\text{head}_1, \dots, \text{head}_k, x\}$ be a set of $k + 1$ entities. The maximum distance between any pair of entities in \mathcal{L} over all time is at least h . Since, \mathcal{L} is a subset of \mathcal{X} , $OPT_{P_1}(\mathcal{X}) \geq h$ (Lemma 16). Thus, we can conclude that $\phi_{P_1}(\mathcal{X}) \leq 2 \cdot OPT_{P_1}(\mathcal{X})$.

In the algorithm, the head of a cluster can be considered to be the center of that cluster. For k -CENTER clustering, we assume that $\phi_{P_2}(\mathcal{X}) > 2 \cdot OPT_{P_2}(\mathcal{X})$. This implies that in \mathcal{L} , we have $k + 1$ entities, and their pairwise maximum distance over all time is greater than $2 \cdot OPT_{P_2}(\mathcal{X})$. By the pigeonhole principle, at least two of these entities (say, head_i and head_j) must be in the same cluster in the optimal k -CENTER clustering. Let s be the center of that cluster. Therefore, we get

$$\max_{t \in T} \delta(\text{head}_i(t), s(t)) \leq OPT_{P_2}(\mathcal{X}), \quad \max_{t \in T} \delta(\text{head}_j(t), s(t)) \leq OPT_{P_2}(\mathcal{X}).$$

By triangle inequality, $\max_{t \in T} \delta(\text{head}_i(t), \text{head}_j(t)) \leq 2 \cdot OPT_{P_2}(\mathcal{X})$. This contradicts our initial assumption. Therefore, $\phi_{P_2}(\mathcal{X})$ cannot be greater than $2 \cdot OPT_{P_2}(\mathcal{X})$. ◀

From the above discussion, we can state the following theorem.

► **Theorem 18.** *A 2-approximate solution for the k -MM and k -CENTER clustering of moving entities in \mathbb{R}^d can be computed in $O(\tau d n k)$ time.*

6 Conclusion and possible directions for future research

In this paper, we examine the k -MM and k -CENTER clustering problems for sets of moving entities in \mathbb{R}^d . Both problems have been studied for sets of fixed points in \mathbb{R}^d ; this paper examines these problems in the mobile setting. We show that both problems are NP-hard, even if entities move in \mathbb{R} . The 2-MM clustering problem can be solved exactly in $O(\tau d n^2)$ time in \mathbb{R}^d , and $O(\tau n \log n)$ time in \mathbb{R} . Unlike the 2-MM clustering problem, the 2-CENTER clustering problem is NP-hard in \mathbb{R}^2 . We show that our 2-MM clustering algorithm gives a 1.15-approximate solution for the 2-CENTER clustering problem. Furthermore, we prove that no polynomial-time algorithm can achieve a better approximation ratio unless $P = NP$. We use the idea of Gonzalez's algorithm [12] and provide a 2-approximation algorithm for the k -MM and k -CENTER clustering problems. Possible directions for future research include

studying these problems when the time stamps of entities differ. It would be interesting to develop an algorithm for the 2-MM clustering problem that runs in less than $O(\tau dn^2)$ time. Finally, if the distance traveled by an entity between two time stamps is bounded by a constant, can we develop algorithms for these problems whose running time is logarithmic in τ ?

References

- 1 P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.
- 2 P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys (CSUR)*, 30(4):412–458, 1998.
- 3 S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Lower and upper bounds for tracking mobile users. In *Foundations of Information Technology in the Era of Network and Mobile Computing*, pages 47–58. Springer, 2002.
- 4 K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. In *Proc. Workshop on Algorithms and Data Structures*, volume 8037 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2013.
- 5 K. Buchin, A. Driemel, J. Gudmundsson, M. Horton, I. Kostitsyna, and M. Löffler. Approximating (k, l) -center clustering for curves. In *Proc. Symposium on Discrete Algorithms*, pages 2922–2938. SIAM, 2019.
- 6 A. Driemel, A. Krivošija, and C. Sohler. Clustering time series under the Fréchet distance. In *Proc. Symposium on Discrete Algorithms*, pages 766–785, 2016.
- 7 S. Durocher. *Geometric facility location under continuous motion*. PhD thesis, University of British Columbia, 2006.
- 8 S. Durocher and D. Kirkpatrick. The Steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *International Journal of Computational Geometry & Applications*, 16(04):345–371, 2006.
- 9 S. Durocher and D. Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *International Journal of Computational Geometry & Applications*, 18(03):161–183, 2008.
- 10 D. Eppstein. Faster construction of planar two-centers. In *Proc. Symposium on Discrete Algorithms*, pages 131–138. ACM/SIAM, 1997.
- 11 G. N. Frederickson. Parametric search and locating supply centers in trees. In *Proc. Workshop on Algorithms and Data Structures*, volume 519 of *Lecture Notes in Computer Science*, pages 299–319. Springer, 1991.
- 12 T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 13 S. Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004.
- 14 C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving objects. *IEEE Trans. Knowl. Data Eng.*, 19(9):1161–1174, 2007.
- 15 J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. international conference on Management of data*, pages 593–604. ACM, 2007.
- 16 Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proc. international conference on Knowledge discovery and data mining*, pages 617–622. ACM, 2004.
- 17 Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.
- 18 T. W. Liao. Clustering of time series data - a survey. *Pattern recognition*, 38(11):1857–1874, 2005.
- 19 N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

22:14 Clustering Moving Entities in Euclidean Space

- 20 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.
- 21 X. Tan and B. Jiang. Simple $O(n \log^2 n)$ algorithms for the planar 2-center problem. In *Proc. Computing and Combinatorics Conference*, volume 10392 of *Lecture Notes in Computer Science*, pages 481–491. Springer, 2017.
- 22 G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.

Trajectory Visibility

Patrick Eades

University of Sydney, Australia
patrick.eades@sydney.edu.au

Ivor van der Hoog

Utrecht University, the Netherlands
i.d.vanderhoog@uu.nl

Maarten Löffler

Utrecht University, the Netherlands
m.loffler@uu.nl

Frank Staals

Utrecht University, the Netherlands
f.staals@uu.nl

Abstract

We study the problem of testing whether there exists a time at which two entities moving along different piece-wise linear trajectories among polygonal obstacles are mutually visible. We study several variants, depending on whether or not the obstacles form a simple polygon, trajectories may intersect the polygon edges, and both or only one of the entities are moving.

For constant complexity trajectories contained in a simple polygon with n vertices, we provide an $\mathcal{O}(n)$ time algorithm to test if there is a time at which the entities can see each other. If the polygon contains holes, we present an $\mathcal{O}(n \log n)$ algorithm. We show that this is tight.

We then consider storing the obstacles in a data structure, such that queries consisting of two line segments can be efficiently answered. We show that for all variants it is possible to answer queries in sublinear time using polynomial space and preprocessing time.

As a critical intermediate step, we provide an efficient solution to a problem of independent interest: preprocess a convex polygon such that we can efficiently test intersection with a quadratic curve segment. If the obstacles form a simple polygon, this allows us to answer visibility queries in $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ time using $\mathcal{O}(n \log^5 n)$ space. For more general obstacles the query time is $\mathcal{O}(\log^k n)$, for a constant but large value k , using $\mathcal{O}(n^{3k})$ space. We provide more efficient solutions when one of the entities remains stationary.

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

Keywords and phrases trajectories, visibility, data structures, semi-algebraic range searching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.23

Funding All authors are partially supported through the University of Sydney – Utrecht University Partnership Collaboration Awards: Algorithms and data structures to support computational movement analysis.

Ivor van der Hoog: Supported by the Netherlands Organisation for Scientific Research (NWO); 614.001.504.

Maarten Löffler: Partially supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 614.001.504 and 628.011.005.

1 Introduction

We consider the following question: two entities q and r follow two different trajectories, with (possibly different) constant speed. Their trajectories lie in an environment with obstacles that block visibility. Can the two entities, at any time, see each other? This question combines two key concepts from computational geometry, namely *trajectories* and *visibility*.



© Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 23; pp. 23:1–23:22

Leibniz International Proceedings in Informatics



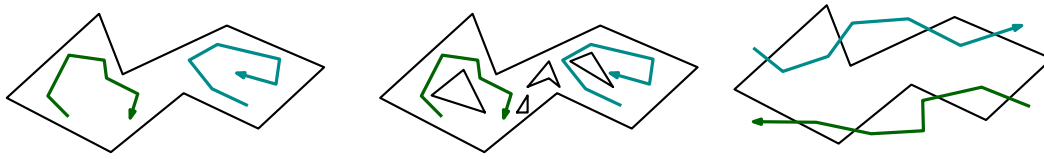
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Trajectories. A recent increase in the availability of low-cost, internet connected GPS tracking devices has driven considerable interest in spatio-temporal data (commonly called *trajectories*) across fields including GIScience, databases, and computational geometry. Problems studied recently in computational geometry include detecting and describing flocks [5, 31], detecting hotspots, clustering and categorising trajectories, map construction and others [9, 25, 32]. Formally, a trajectory is a sequence of time-stamped locations in the plane, or more generally in \mathbb{R}^d , which models the movement of an entity. Trajectory data is obtained by tracking the movements of animals [7, 30], hurricanes [41], traffic [33], or other moving entities [20] over time. For a more extensive overview of trajectory analysis we refer the reader to the excellent survey by Gudmundsson et al. [27].

Visibility. Two points, amidst a number of obstacles, are mutually *visible* if the line segment between them does not intersect any obstacle. Visibility is one of the most studied topics in computational geometry [36, 42] and in adjacent fields such as computer graphics [22], GIScience [24], and robotics [36]. Within computational geometry, Gosh and Giswani compiled a survey of *unsolved* problems in this area [26]. Core visibility problems in computational geometry include *ray shooting* [16, 13], guarding [15, 23] and visibility graph recognition [10]. For more information about visibility problems, refer to O'Rourke's book [39] ch. 8, and the surveys by Durant [22] and Gosh [26].

Trajectory visibility. In this paper, we study the following fundamental question, which we refer to as *trajectory visibility* testing. Given a simple polygon, or a polygonal domain, P , and the trajectories of two moving entities q and r , is there a time t at which q and r can see each other? We assume that q and r move linearly with constant (but possibly different) speeds between trajectory vertices, and cannot see through the edges of P . We distinguish several variants depending on whether P is a simple polygon or a polygonal domain, and whether the trajectories are allowed to intersect P (e.g. vehicles moving through fog, animals moving through foliage) or not (e.g. pedestrians moving among buildings, ships moving on water bodies). These variants are illustrated in Figure 1. We further consider the same variants in the simpler scenario in which one of the entities is a point (e.g. a stationary guard and a moving intruder). Note that we are interested only whether there *exists* a time at which the two entities see each other. This implies we can temporally decompose the problem: the answer is *no* if and only if the answer is no between all two consecutive time stamps. When considering this question, two fundamentally different approaches come to mind. On the one hand, when the number τ of trajectory vertices is small compared to the number n of polygon vertices, the best approach may be to simply solve the problem for each time interval separately. On the other hand, when τ is large compared to n , it may be more efficient to spend some time on preprocessing P first, if this allows us to spend less time per trajectory edge. We therefore distinguish between the *algorithmic* question and the *data structure* question. Our results are discussed below and summarized in Table 1.

Related work on visibility for moving entities. There is a vast amount of research on both trajectories and visibility, but surprisingly not much previous work exists on their combination. One reason is that the already developed tools for visibility and trajectory analysis cannot be combined in a straightforward manner. Consider two trajectories q and r within a simple polygon P : existing visibility tools allow us to easily check if there are subtrajectories of q and r which are mutually visible. However, the two moving entities see each other only if there is a time at which the two entities are simultaneously within two



■ **Figure 1** Different variations of the problem: two trajectories inside a simple polygon (left), two trajectories in a polygonal domain (middle), or two trajectories intersecting a simple polygon (right). Our approach for the middle variant is independent of whether the trajectories intersect the domain.

mutually visible subtrajectories. There could be quadratically many pairs of subtrajectories which are mutually visible; yet, it could be that the two entities are never simultaneously within such a pair. To determine visibility between moving entities, one needs to incorporate the concept of time into pre-existing tools for visibility. An early result in this direction is by Bern et al. [6] and Mulmuley [37], who study maintaining the visibility polygon of a point that moves over a straight path. Aronov et al. [3] demonstrate a kinetic data structure that tracks the visibility polygon of a moving query point q . The most recent result on visibility and motion is by Diez et al. [18] who show how to maintain the shortest path between two moving entities using a kinetic data structure. Here q and r are mutually visible if and only if their shortest path is a line segment.

From event based modelling to algebraic range searching. The above attempts for visibility testing model the passage of time using a sequence of *events*. However, as q and r traverse their trajectories there may be a linear number of such events and thus far, there does not exist a data structure that can answer if there is visibility between trajectories in sub-linear time. In this paper, we diverge from the classical event-based approach and model the problem in an algebraic way. Such an algebraic reformulation is not rare in computational geometry: for example, circular range queries get solved by reformulating them into higher-dimensional halfspace range queries. Yet we are unaware of any algebraic approaches used for visibility testing. The challenge with such an algebraic approach is that the more complicated the algebraic expression, the worse the complexity of both the paper and the runtimes involved. However, our transformation in Section 2 that transforms visibility testing into testing for an intersection between a convex polygon and an algebraic curve, uses only degree two planar polynomials. This transformation allows us to introduce tools and concepts from algebraic range searching to obtain the first sublinear query times for visibility testing for a variety of geometric settings.

Our Results. We focus on trajectories of at most two vertices; any set of trajectories of τ vertices can be handled by applying our algorithms or queries τ times. Our results are summarized in Table 1. In Section 2, we discuss our algorithmic results; we build on the structural geometric properties established in this section in the remainder of the paper. In Section 3 we consider the sub-problem of preprocessing a convex polygon P' for intersection queries with quadratic curve segments. We then extend the solution in Section 4 to a data structure for visibility testing in a simple polygon P using multi-level data structures. In Section 5 we discuss the case in which one of the entities is stationary. In Section 6 we briefly outline our results for polygonal domains, more detailed descriptions of this data structures can be found in Appendix C. Due to space constraints, several proofs and the full description of all results are deferred to the appendix.

■ **Table 1** The two leftmost columns specify if the query entity is a point (●) or line segment (/). The third column specifies if the domain P is a simple polygon where the query segments may not intersect P (S), a simple polygon where the query segments may intersect P (I), or a polygonal domain with n vertices where the query segments may intersect P (D). The value k is an unspecified constant.

q	r	P	algorithm	data structure			source
				space	preprocessing	query	
●	●	S or I	$\Theta(n)$	$\mathcal{O}(n)$	$\Theta(n)$	$\Theta(\log n)$	[28]
		D	$\Theta(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\Theta(\log n)$	[40]
●	/	S	$\Theta(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\Theta(\log n)$	Section 5
		I	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2 \log^3 n)$	$\mathcal{O}(n^2 \log^3 n)$	$\mathcal{O}(n^{\frac{3}{4}+\epsilon})$	Section 5
		D	$\Theta(n \log n)$	$\mathcal{O}(n^4 \log^3 n)$	$\mathcal{O}(n^4 \log^3 n)$	$\mathcal{O}(n^{\frac{3}{4}+\epsilon})$	Section 5
/	/	S	$\Theta(n)$	$\mathcal{O}(n \log^5 n)$	$\mathcal{O}(n^1 \log^5 n)$	$\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$	Section 4
		I	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(\log^k n)$	Appendix C
		D	$\Theta(n \log n)$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(\log^k n)$	Appendix C

2 Algorithms for testing visibility

Let P be a polygonal domain with n edges and let q and r each be a line segment or a point in the plane. We first present an $\mathcal{O}(n \log n)$ time algorithm to solve the visibility problem for q, r and P and we show this algorithm is tight in the worst case. Then we show how to solve the visibility problem in linear time in the case where P is a simple polygon and q and r are contained in P . All other sections depend on the notion of *hourglass* that is presented here.

An $\mathcal{O}(n \log n)$ time algorithm. The entities q and r each move along a line segment with constant speed (depending on the length of the line segment) during the time interval $[0, 1]$.¹ Consider the line $g(t)$ through both entities at time t . We dualize this line to a point using classical point-line dualization (i.e. we map the line $y = ax + b$ to the point $(a, -b)$); this point $\gamma(t)$ now traces a segment of a curve $\gamma : [0, 1] \rightarrow \mathbb{R}^2$ in the dual space.²

► **Lemma 1.** *The segment γ is a segment of a quadratic curve with 5 degrees of freedom.*

Let e be an edge of P and denote by L_e the set of lines intersecting e . The dual of L_e is a wedge Λ_e [17]. If the segment between q and r is blocked by e at time t then $g(t)$ must lie in L_e . In the dual, this means that the curve segment γ must intersect Λ_e . There are at most two connected time intervals where a quadratic curve segment γ can intersect a wedge Λ_e ; it follows that each edge e has at most two connected time intervals where it blocks the visibility between q and r . This leads to a straightforward general algorithm to test if there is a time at which q can see r : for each edge $e \in P$, we compute the at most two time intervals where it blocks visibility between q and r in constant time. We sort these time intervals in $\mathcal{O}(n \log n)$ time and check if their union covers the time interval $[0, 1]$.

► **Theorem 2.** *Given a polygonal domain P with n vertices and moving entities q and r , we can test trajectory visibility in $\mathcal{O}(n \log n)$ time.*

¹ With slight abuse of notation, we use q and r to refer to both the (moving) entities and their trajectory.

² Throughout this paper we follow the convention of using latin letters for objects in the primal space and greek letters for their duals.



■ **Figure 2** The reduction when P is a polygonal domain with $\Omega(n)$ holes.

An $\Omega(n \log n)$ lower bound. If P is a polygonal domain with $\Omega(n)$ holes, this result is tight: suppose we are given a set A of n numbers and we want to test if $A = B$ for a given arbitrary sorted set $B = \{x_1, x_2, \dots, x_n\}$. Ben-Or [4] shows that this problem has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model. This leads to the following reduction (illustrated in Figure 2): we construct a set of n horizontal edges whose y -coordinates are 0 and whose x -coordinates are $\{(x_i + \varepsilon, x_{i+1} - \varepsilon) \mid i \in [1, n - 1]\}$ where ε is smaller than half of the minimal difference between two consecutive numbers in B ; a value for ε can be found in linear time since B is sorted. For each of the n numbers $x \in A$ we construct an axis-aligned rectangle from the point $(x, 1)$ to $(x, 2)$ with a width of 2ε . The entity q walks from the point $(x_1, -1)$ to $(x_n, -1)$ and entity r walks from $(x_1, 3)$ to $(x_n, 3)$. Suppose the number x_j from B is not in A , then q can see r at the x -coordinate x_j . Note that this construction also extends to the case where one of the two entities is stationary: consider the cone between the stationary entity q and a horizontal line segment trajectory r : we can transform the set B into a set of n horizontal edges that cut in the cone between q and r . Each rectangle modelling a number $a \in A$ gets stretched such that it intersects a ray from q to r .

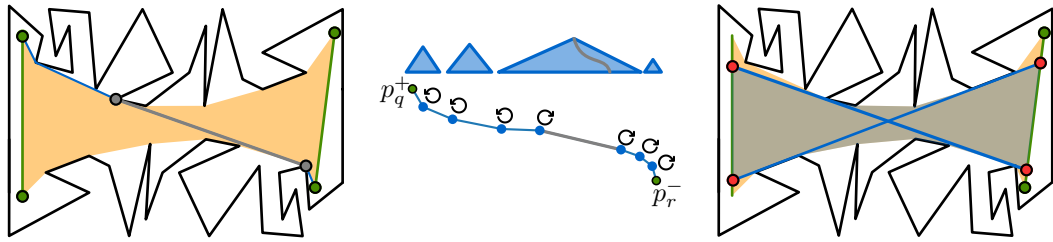
► **Theorem 3.** *There exists a polygonal domain P with n vertices, and entities q and r moving inside P for which testing trajectory visibility requires $\Omega(n \log n)$ time.*

A linear-time algorithm for when P is a simple polygon and $q, r \subset P$. Any segment between q and r that is contained in P is a geodesic shortest path in P between a point on q and a point on r . Guibas and Hershberger [28] define, for any two segments q and r in a simple polygon P , the *hourglass* $H(q, r)$ to be the union of all shortest paths between points on q and r . The hourglass $H(q, r)$ is a subset of P and bounded by the segments q and r and by two shortest paths. The “upper” chain³ is the shortest path between the upper end points p_q^+ and p_r^+ of q and r , and the “lower” chain is the shortest path between p_q^- and p_r^- (refer to Figure 3). We define the *visibility glass* $L(q, r)$ as the (possibly empty) union of *line segments* between q and r that are contained in P . Notice that $L(q, r)$ is a subset of $H(q, r)$.

► **Observation 1.** *For any two segments $q, r \subset P$ either $L(q, r)$ is empty or there exist segments $q' \subset q, r' \subset r$ such that $L(q, r) = H(q', r')$. Moreover, q' and r' are bounded by two bitangents on the shortest paths between the endpoints of q and r .*

Proof. Suppose that the interior of the upper and lower chains of $H(q, r)$ intersect. Then the visibility glass $L(q, r)$ is either a single segment or empty. Thus we can either find two points q' and r' on q and r whose line segment forms $H(q', r') = L(q, r)$ or $L(q, r)$ is empty. If the interior of the upper and lower chains are disjoint then they are semi-convex [28].

³ We use the names “upper” and “lower” since they intuitively correspond to our figures. If q and r are not vertical but their endpoints are in convex position, we rotate the plane until one of them is vertical. If the endpoints of q and r do not lie in convex position, the two chains share an endpoint, which is a simpler case.



■ **Figure 3** (left) $H(q, r)$ in orange. The path from p_q^+ to p_r^- shown as a dotted path. (middle) The path from p_q^+ to p_r^- can be represented as a collection of binary trees on the vertices. The bitangent (in grey) is incident to the only vertex at which the path switches from making left turns to making right turns (or vice versa) (right) Using the bitangents, we identify the endpoints of q' and r' and obtain $L(q, r)$ in grey.

Consider the shortest path from p_q^+ to p_r^- , it has one edge (u, v) connecting the upper and lower chain. This is the unique edge for which the path makes a clockwise turn at u and a counterclockwise turn at v or vice versa. There exists an edge with similar properties on the shortest path between p_q^- and p_r^+ . The extension of these two edges bounds q' and r' [14]. ◀

Chazelle and Guibas [14] note that (the supporting lines of) all line segments in $L(q, r)$ can be dualized into a convex polygon of linear complexity which we denote by $\Lambda(q, r)$. The shortest path between two points in P can be computed in linear time [29]. Finding the bitangents also takes linear time. It follows that we can compute $L(q, r)$ and its dual $\Lambda(q, r)$ in linear time. Suppose that we are given two entities q and r contained in a simple polygon P . Recall that the line $g(t)$ through q and r traces a quadratic segment γ in the dual.

► **Observation 2.** *Entities q and r are mutually visible at time t if $\gamma(t)$ lies in $\Lambda(q, r)$.*

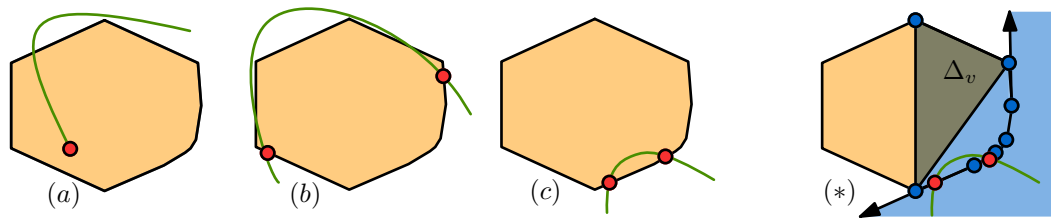
Proof. The entities can see one another at time t if and only if $g(t) \in L(q, r)$. ◀

We can derive γ in constant time, construct $\Lambda(q, r)$ in linear time, and we can check if a quadratic curve intersects a convex polygon in linear time. Thus we conclude:

► **Theorem 4.** *Given a simple polygon P with n vertices and two entities q and r moving linearly inside P , we can test trajectory visibility in $\Theta(n)$ time.*

3 Intersecting a convex polygon with an algebraic curve

We now turn our attention to the data structure question: can we preprocess P such that trajectory visibility may be tested efficiently (i.e. in sublinear time) for a pair of *query* segments q, r ? By Observation 2, we can phrase such a query as an intersection between a quadratic curve segment and a convex polygon. Note, however, that both the curve segment and the convex polygon depend on the query segments q and r . As an intermediate step, we study a simplified problem in which the convex polygon is independent of q and r . In particular, the question that we study is: let P' be a convex polygon with n edges. Is it possible to preprocess P' such that for any quadratic curve segment γ , we can quickly test if γ intersects P' ? We believe this problem to be of independent interest. We then use our solution to this question as a subroutine in Section 4.



■ **Figure 4** The cases (a), (b) and (c) of intersection between γ and P' . (*) the right-subspace bounded by three halfspaces. The red points are the points of intersection which we try to identify.

Semi-algebraic range searching. Let X be a set of n geometric objects, where each object is parametrized by a vector \vec{x} in \mathbb{R}^d (e.g. a point is parametrized by a vector of its coordinates). Let Γ be a family of geometric regions (called semi-algebraic ranges) in \mathbb{R}^d where each region $G \in \Gamma$ is bounded by an algebraic surface γ which is parametrized by a vector \vec{a} . Agarwal et al. [2] describe how to store X , such that for any query range $G \in \Gamma$, we can efficiently count the number of objects of X whose parameter vector lies in G . Let F be a function mapping (the parameterizations of) $x \in X$ and $G \in \Gamma$ to \mathbb{R} such that $F(\vec{x}, \vec{a}) \leq 0$ if and only if $\vec{x} \in G$. Agarwal et al. show that if F can be written in the form $F(\vec{x}, \vec{a}) = g_0(\vec{a}) + \sum_{i=1}^k g_i(\vec{a})f_i(\vec{x})$ then there is a function f that maps the objects in X to points in \mathbb{R}^k , and a function g that maps the ranges in Γ to halfspaces in \mathbb{R}^k , such that $f(x) \in g(G)$ if and only if $\vec{x} \in G$. This so-called *linearization* process transforms a d -dimensional semi-algebraic range searching problem into a halfspace range searching problem in \mathbb{R}^k . Refer to Appendix B for examples.

The resulting set of n points in \mathbb{R}^k can be stored in a data structure of linear size such that the points in a query halfspace can be counted in $\mathcal{O}(n^{1-\frac{1}{k}})$ time (with high probability) [11]. Testing if a query halfspace is empty can be done in expected $\mathcal{O}(n^{1-\frac{1}{k/2}})$ time. Data structures with a slightly slower deterministic query time are also known [11]. If we are willing to use (much) more space, faster query times are also possible [11, 35].

Semi-algebraic range searching and our intersection query. The n edges of a convex polygon P' are geometric objects. A natural parametrization for an edge $e \in P'$ is a 4-dimensional vector \vec{x}_e specifying its start and end points. A quadratic curve segment γ per definition is an semi-algebraic range parametrized by its own parameters \vec{a}_γ . If we want to apply semi-algebraic range searching, we need to design a predicate function $F(\vec{x}_e, \vec{a}_\gamma)$ that outputs a negative real number whenever the edge e is intersected by γ .

It is tempting to immediately construct such a predicate function using the parameters \vec{x}_e and \vec{a}_γ only (ignoring geometric facts such as the convexity or connectedness of P'). However, we have to linearize the resulting predicate function $F(\vec{x}_e, \vec{a}_\gamma)$ into k terms. The more complex the description of the objects, the query, and their intersection, the more complex the predicate will be and therefore the higher this number k will be.

Geometry of our intersection query. Let P' be a convex polygon with n edges. Let γ be a quadratic curve segment ending in the points s and z and let Γ denote the unique degree-2 curve $\Gamma \supset \gamma$ given by $a_1x^2 + a_2x + a_3xy + a_4y + a_5y^2 + a_6 = 0$. We say $\vec{a} = (a_1, \dots, a_6)$. Observe (Figure 4) that if γ intersects P' , then either (a) an endpoint s or z lies in P' , or (b) γ cuts off a vertex or (c) γ intersects only a single edge of P' twice and has no endpoint in P' (we call this *dipping*). Intersections of type (a) and (c) can be identified with a regular binary search on P' . An intersection of type (b) is detected using algebraic range searching.

Since P' is convex, we can test if an endpoint of γ lies inside P' in $\mathcal{O}(\log n)$ time. To detect an intersection of case (c), we store a Dobkin-Kirkpatrick hierarchy [19] of P' . This takes $\mathcal{O}(n)$ space and requires $\mathcal{O}(n)$ preprocessing time. Given γ , we detect an intersection of type (c) as follows (Figure 4 (*)): any node v in this decomposition represents a sub-polygon P'' of P' and a triangle Δ_v which splits P'' in a left and right part. Consider the border of the right part $R = (e_1, e_2, \dots, e_m)$. Note that if γ does not intersect Δ_v , then γ can only dip an edge in R if it is contained in the union of the halfspaces that lie to the right of the lines supporting: (1) one edge of Δ_v and (2) e_1 and e_m (refer to the blue area in the figure). Given the node v we do three constant time checks. First we check if γ intersects Δ_v . If not then we check for both the left and right sub-polygon if γ is contained in the specified area. If that is the case for both or neither sub-polygons then γ can never dip an edge of P' , else we recurse in the appropriate subtree. It follows that we can detect case (c) in $\mathcal{O}(\log n)$ time.

► **Lemma 5.** *We can preprocess a convex polygon P' consisting of n edges in $\mathcal{O}(n)$ time and using $\mathcal{O}(n)$ space, such that for any degree-2 curve segment γ we can detect an intersection of type (a) or (c) in $\mathcal{O}(\log n)$ time.*

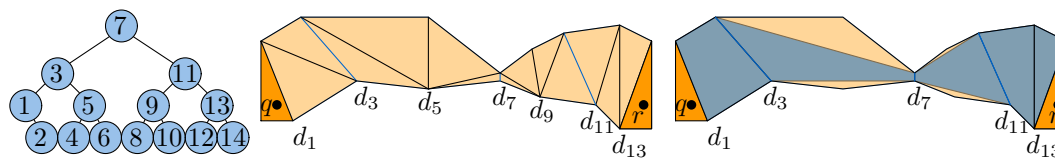
The curve Γ of which γ is a segment divides the plane into two areas, $\Gamma^- := \{a_1x^2 + a_2x + a_3xy + a_4y + a_5y^2 + a_6 \leq 0\}$ and its complement Γ^+ . An edge $((x_1, x_2), (x_3, x_4))$ of P' is intersected by γ with an intersection of type (b) only if one endpoint of the edge lies in Γ^- and the other in Γ^+ . If Γ is a curve with $k + 1 \leq 6$ degrees of freedom then the formulation of Γ^- and Γ^+ is a predicate that specifies whenever a point $\vec{x} = (x_1, x_2)$ lies in Γ^- or Γ^+ with k linearized terms. Thus we can detect if an edge has two endpoints on opposite sides of Γ with two consecutive halfspace range queries in \mathbb{R}^k . We build a three-level data structure where the first two levels are 5-dimensional partition trees [34, 11].⁴ On top of each node in the second level we build a binary tree on the clockwise ordering (with respect to P') of the edges in that node.

During query time, we transform the degree-2 curve Γ into two k -dimensional halfspaces $g(\Gamma^+)$ and $g(\Gamma^-)$. With two consecutive halfspace range queries we obtain the collection $E_\Gamma(P)$ of edges which have one endpoint on each side of Γ in $\mathcal{O}(n^{1-\frac{1}{k}})$ time. Note that the set $E_\Gamma(P)$ does not have to be a connected set of edges of P (refer to Figure 4 (b)). However, the subset of $E_\Gamma(P)$ that is intersected by the curve *segment* γ is consecutive in the clockwise ordering of $E_\Gamma(P)$. The set $E_\Gamma(P)$ is returned as $\mathcal{O}(n^{1-\frac{1}{k}})$ subtrees $\{T_1, T_2, \dots, T_m\}$ of the secondary trees. Consider a subtree T_i and the associated binary search tree on its edges. Because of the earlier discussed property, the subset of $E_\Gamma(P)$ that is intersected by the segment γ must be a consecutive subset of the leaves of T_i . Thus using T_i we can obtain these consecutive leaves in $\mathcal{O}(\log n)$ time by testing if the segment γ lies before or after the point of intersection between Γ and an edge in $E_\Gamma(P)$.

The time and space needed for detecting case (b) dominates the time and space needed for case (a) and (c) and we conclude:

► **Theorem 6.** *Let P' be a convex polygon with n vertices. In $\mathcal{O}(n \log^2 n)$ time we can build a data structure of size $\mathcal{O}(n \log^2 n)$ with which we can test if an arbitrary degree-2 query curve segment γ with $k + 1 \leq 6$ degrees of freedom intersects P' in $\mathcal{O}(n^{1-\frac{1}{k}} \log n)$ time.*

⁴ Alternatively, cutting trees [12] can be applied to obtain faster query time at a larger space cost.



■ **Figure 5** A triangulated polygon P with the diagonals labelled d_1 to d_{14} . There are 14 diagonals between q and r . However, we have pre-stored hourglasses $H(d_1, d_3)$, $H(d_3, d_7)$, $H(d_7, d_{11})$ and $H(d_{11}, d_{14})$. At query time, we only have to concatenate these $\mathcal{O}(\log n)$ hourglasses to get $H(d_1, d_{14})$.

4 A data structure for two entities moving inside a simple polygon

In this section we build a data structure to answer trajectory visibility queries when both the entities q and r move linearly, possibly at different speeds, inside a simple polygon P . Our main approach is the same as in our algorithm from Theorem 4: we obtain the convex polygon $\Lambda(q, r)$ that is the dual of the visibility glass $L(q, r)$, and test if the curve segment γ tracing the line through q and r in the dual space intersects $\Lambda(q, r)$. By Observation 2 this allows us to answer trajectory visibility queries. The main challenge is that we cannot afford to construct $\Lambda(q, r)$ explicitly. Instead, our data structure will allow us to obtain a compact representation of $\Lambda(q, r)$ that we can query for intersections with γ .

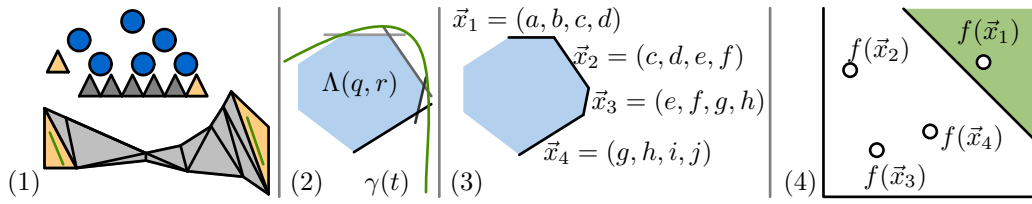
To obtain $\Lambda(q, r)$ we use a variation of two-point shortest-path query data structure of Guibas and Hershberger [28]. Their data structure (Figure 5) compactly stores a collection of hourglasses that can be concatenated to obtain a shortest path between two arbitrary points $p, p' \in P$. All shortest paths, in particular the boundary of the hourglasses, are represented using balanced binary search trees storing the vertices on the path. By reusing shared subtrees these $\mathcal{O}(n)$ hourglasses can be stored using only $\mathcal{O}(n)$ space. To report the shortest path between two query points their data structure concatenates $\mathcal{O}(\log n)$ of these hourglasses. The result is again represented by a balanced binary tree.

We now present a short overview of our data structure (refer to Figure 6). Unlike the Guibas and Hershberger structure, we store the hourglasses explicitly. In particular, the vertices on the boundary of an hourglass are stored in the leaves of a balanced binary search tree. The internal nodes of these trees correspond to semi-convex subchains. Let T denote the collection of all these nodes. Each node $v \in T$ stores its subchain C_v in an associated data structure. Specifically, we dualize the supporting-lines of the edges in C_v to points (refer to Figure 7). Two consecutive edges produce two points in the dual, which we again connect into semi-convex polygonal chains. So for every vertex in the sub-chain C_v the associated data structure Δ_v actually stores a line-segment; together these segments again form a polygonal chain Ψ_v . The associated data structure will support intersection queries with a quadratic query segment γ ; i.e. it will allow us to report the segments of Ψ_v intersected by γ . We implement Δ_v using the data structure from Theorem 6.

► **Lemma 7.** *The total size of all chains Ψ_v over all nodes v is $\mathcal{O}(n \log^3 n)$.*

Proof. The Guibas and Hershberger data structure is essentially a balanced hierarchical subdivision that recursively partitions the polygon into two roughly equal size subpolygons. Every subpolygon has $\mathcal{O}(\log n)$ diagonals [28], and thus stores at most $\mathcal{O}(\log^2 n)$ hourglasses.⁵

⁵ We use the version of Guibas and Hershberger's structure that achieves only $\mathcal{O}(\log^2 n)$ query time.



■ **Figure 6** (1) The base level of our data structure is a hierarchical triangulation. (2) Given q and r , we compute $\Lambda(q, r)$ and the degree-2 curve segment γ . (3) We store the parameters of each edge of $\Lambda(q, r)$ (4) Each parameter vector gets mapped to a point in \mathbb{R}^4 and the query curve segment gets mapped to a 4-dimensional halfspace which is empty only if γ intersects no edge from $\Lambda(q, r)$.

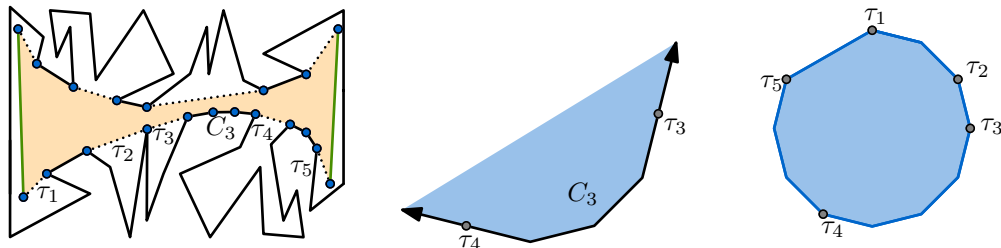
It follows that all hourglasses of a subpolygon of size m use at most $\mathcal{O}(m \log^2 n)$ space. The height of the balanced hierarchical subdivision is $\mathcal{O}(\log n)$, and at every level the total size of the subpolygons is $\mathcal{O}(n)$. Therefore, the total size of all subpolygons is $\mathcal{O}(n \log n)$. ◀

For a chain of size $m = |\Psi_v|$ the data structure Δ_v has size $\mathcal{O}(m \log^2 m)$ and can be built in $\mathcal{O}(m \log^2 m)$ time. It follows that our data structure uses $\mathcal{O}(n \log^5 n)$ space in total, and can be built in $\mathcal{O}(n \log^5 n)$ time.

Querying the data structure. When we get a trajectory visibility query with entities q and r we have to test if the curve γ traced by the point dual to the line through q and r intersects $\Lambda(q, r)$. By Observation 1 the primal representation $L(q, r)$ of $\Lambda(q, r)$ is an hourglass $H(q', r')$. We now argue that: (i) we can find the subsegments q' and r' in $\mathcal{O}(\log n)$ time, (ii) that our data structure can report $\mathcal{O}(\log^2 n)$ nodes from T that together represent an hourglass $H(q', r')$, and (iii) that we can then test if γ intersects $\Lambda(q, r)$ by using the associated data structures of these reported nodes. This will result in $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ query time.

► **Lemma 8.** *Given our data structure, we can detect if $L(q, r)$ is empty, or compute the subsegments $q' \subseteq q$ and $r' \subseteq r$ such that $L(q, r) = H(q', r')$ in $\mathcal{O}(\log n)$ time.*

Proof. By Observation 1 the visibility glass $L(q, r)$ is either empty or the hourglass $H(q', r')$ for two subsegments q' and r' and these two subsegments are bounded by the two bitangents of $H(q, r)$. These bitangents are the extension of two edges, from the shortest paths between the edges of q and r . We explained in the proof of Observation 1 that the hourglass $H(q, r)$ had an upper and lower semi-convex chain which may or may not share a point. The upper and lower chain are both a shortest path between endpoints of q and r . We can obtain them



■ **Figure 7** (left) An hourglass between q and r in orange. The lower chain consists of four chains that coincide with P , joined by outer tangents in dotted lines labelled $\tau_1 \dots \tau_5$. (middle) The area bounded by the dualized chain C_3 . Note that this chain has four edges since in the primal C_3 has four vertices. (right) A simplified version of $\Lambda(q, r)$. Outer tangents become vertices of $\Lambda(q, r)$.

using the data structure \mathcal{D} from [28] as a balanced binary search tree and we can verify if they share a point using this tree. If that is the case then $L(q, r)$ is either empty or a single segment and we can verify this using an additional $\mathcal{O}(\log n)$ time.

If the upper and lower chain do not share a point then we want to identify the subsegments q' and r' for which $L(q, r) = H(q', r')$ and recall that q' and r' are bounded by the bitangents of $H(q, r)$. Such a bitangent is the extension of an edge (u, v) on the shortest path between two endpoints of q and r . The edge (u, v) is the unique edge on this path for which the path makes a clockwise turn at u and a counterclockwise turn at v or vice versa. Using \mathcal{D} we can obtain any path as a balanced binary search tree. We perform a binary search on this tree to identify the edge (u, v) whose endpoints have this unique clockwise ordering. ◀

We use Lemma 8 to find the endpoints q_1, q_2 of q' and r_1, r_2 of r' , respectively. We can obtain the shortest paths $\pi(r_1, q_1)$ and $\pi(r_2, q_2)$ bounding $L(q, r) = H(q', r')$ by concatenating $\mathcal{O}(\log n)$ of the pre-stored hourglasses. To concatenate two hourglasses we actually select two contiguous subchains in both hourglasses, and compute two bridge edges connecting them. Such a contiguous subchain can be represented by $\mathcal{O}(\log n)$ nodes in the binary search trees representing the hourglass boundary. It follows that $\pi(r_1, q_1)$ can be represented by $\mathcal{O}(\log^2 n)$ nodes; each representing a pre-stored subchain in the data structure, together with $\mathcal{O}(\log^2 n)$ line-segments (the bridge segments). We now observe that the chains stored in the associated data structures of these nodes together with $\mathcal{O}(\log^2 n)$ line segments Ξ (the duals of the bridge segments) actually represent the dual $\Lambda(q, r)$ of $L(q, r)$.

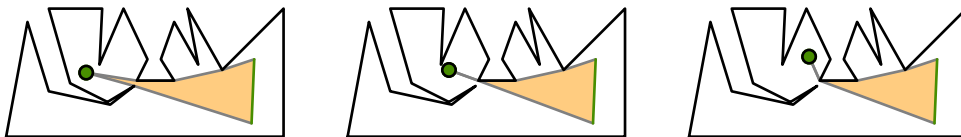
To check if the quadratic query segment γ intersects $\Lambda(q, r)$ we check if one of the endpoints of Q lies in $\Lambda(q, r)$; in this case one of the paths $\pi(r_1, q_1)$ or $\pi(r_2, q_2)$ is actually a single segment, or if γ intersects the boundary of $\Lambda(q, r)$. To this end, we query each of these associated data structures. Since γ has $k + 1 = 5$ degrees of freedom (Lemma 1) this takes $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ time. We test for intersection with the segments in Ξ separately. We thus obtain the following result.

► **Theorem 9.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n \log^5 n)$ that allows us to answer trajectory visibility queries in $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ time. Building the data structure takes $\mathcal{O}(n \log^5 n)$ time.*

5 A data structure for queries with one moving entity

In this section we develop data structures that can efficiently answer trajectory visibility queries in case one of the entities q is stationary, while r travels along a line segment. The approach described above also applies here; but we present a simpler solution using linear space which gives $\mathcal{O}(\log n)$ query time.

We consider three variants of this setting: (i) P is a simple polygon and r is contained in P , (ii) P is a simple polygon but the trajectory of r may intersect edges of P , and (iii) P is a polygonal domain and r may intersect edges of P .



■ **Figure 8** Three times a query pair (q, r) in a simple polygon. In the middle case, the paths π_1, π_2 share their first line segment but there still is a point on r which is visible from q .

Entity r is contained in a simple polygon. Consider the shortest paths π_1, π_2 from q to the end points of r . Observe that if edges of π_1 and π_2 coincide, they coincide in a connected chain from q [28]. Moreover (Figure 8) if more than one line segment of π_1 and π_2 coincide, then any shortest path from q to a point on r cannot be a single line segment. If no edges of π_1 and π_2 coincide then there is at least one point on r , whose shortest path to q is a line segment. If exactly one line segment of π_1 coincides with a segment of π_2 , then that segment must be connected to q and if there is a line-of-sight between q and r , it has to follow that line segment. This observation allows us to answer a visibility query by considering only the first three vertices of π_1 and π_2 . These vertices can be found in $\mathcal{O}(\log n)$ time using the two-point shortest path data structure of Guibas and Hershberger [28]. We conclude:

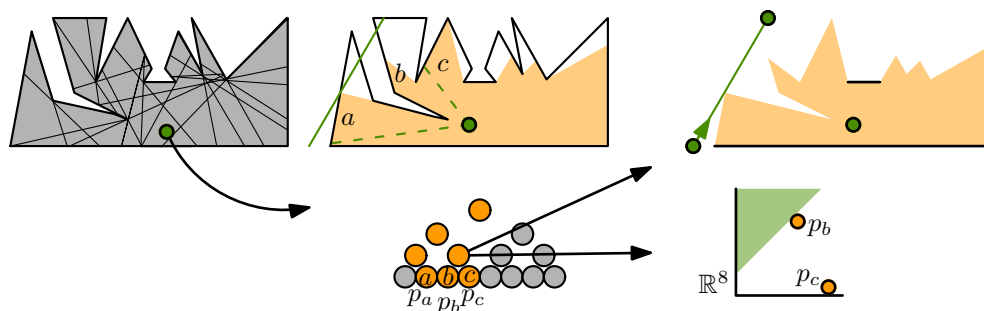
► **Theorem 10.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity in $\mathcal{O}(\log n)$ time. Building the data structure takes $\mathcal{O}(n)$ time.*

Entity r can cross a simple polygon. If r is able to move through edges of P then its trajectory may intersect the boundary of P linearly often. Inspecting each of the resulting subsegments explicitly would thus require at least $\Omega(n)$ time. Hence, we use a different approach. Let V_q denote the *visibility polygon* of point q : the set of all points visible from q . There is a time at which q can see r if and only if the trajectory of r intersects (the boundary of) V_q . We build a data structure to find such a point (if one exists).

Aronov et al. [3] actually developed an $\mathcal{O}(n^2)$ size data structure that can be built in $\mathcal{O}(n^2 \log n)$ time and can report the visibility polygon of an arbitrary query point $q \in P$ in $\mathcal{O}(\log^2 n)$ time. The visibility polygon V_q is returned in its combinatorial representation, that is, as a (pointer to a) balanced binary search tree, storing the vertices of V_q in order along the boundary. It is important to note that this combinatorial representation does not explicitly store the locations of all vertices of V_q . Instead, a vertex v of V_q may be represented by a pair (e, w) , indicating that v is the intersection point of polygon edge e and the line through vertex $w \in P$ and the query point q . Computing the explicit location of all vertices of V_q thus takes $\mathcal{O}(|V_q|)$ time, if so desired, by traversing the tree. We now extend the results of Aronov et al. in such a way that we can efficiently test if a line segment intersects V_q *without* spending the $\mathcal{O}(|V_q|)$ time to compute the explicit locations.

We briefly review the results of Aronov et al. first. They build a balanced hierarchical decomposition of P [14]. Each node v in the balanced hierarchical decomposition represents a subpolygon P_v of P (the root corresponds to P itself) and a diagonal of P_v that splits P_v into two roughly equal size subpolygons P_ℓ and P_r . For subpolygon P_ℓ the data structure stores a planar subdivision \mathcal{S}_ℓ (of the area outside P_ℓ) such that for all points in a cell of \mathcal{S}_ℓ the part of the visibility polygon inside P_ℓ has the same combinatorial representation. Moreover, for each cell it stores the corresponding combinatorial representation. These representations can be stored compactly by traversing \mathcal{S}_ℓ while maintaining the (representation of the) visibility polygon in P_ℓ in a partially persistent red black tree [3]. The data structure stores an analogous subdivision for P_r . The complete visibility polygon of q can be obtained by concatenating $\mathcal{O}(\log n)$ subchains of these pre-stored combinatorial representations (one from every level of the hierarchical decomposition).

We use the same approach as Aronov et al. [3], but we use a different representation of V_q (refer to Figure 9). Our representation will be a weight balanced binary search tree ($BB[\alpha]$ -tree [38]) whose leaves store the vertices of V_q in order along the boundary. An internal node of this tree corresponds to a subchain of vertices along V_q , which is stored in an associated data structure. We distinguish two types of vertices in such a chain: *fixed vertices*,



■ **Figure 9** (left) A simple polygon split in $\mathcal{O}(n^2)$ cells. For each cell, there exists a red-black tree that represents a visibility polygon. (middle) Given V_q and r , r could intersect the explicit V_q depending on the location of q . We shoot two rays from q to r and find their intersection with V_q in the red-black tree. That gives us three leaves highlighted in orange. (right top) All the fixed edges in this node are stored in a ray-shooting data structure, (right bottom) all the variate edges have 8-dimensional points that are stored in an 8-dimensional partition tree.

for which we know the exact location, and *variate* vertices, which are represented by an polygon-edge, polygon-vertex pair (e, w) . We store the fixed vertices in a linear size dynamic data structure that supports halfspace emptiness queries, that is, a dynamic convex hull data structure [8]. This data structure uses $\mathcal{O}(m)$ space, and supports $\mathcal{O}(\log m)$ time updates and queries, where m is the number of stored points. The variate vertices are mapped to a point in \mathbb{R}^8 using a function f that is independent of q . We give the precise definition later. We store the resulting points in a dynamic data structure that can answer halfspace emptiness queries [1]. This data structure uses $\mathcal{O}(m \log m)$ space, answers queries in $\mathcal{O}(m^{\frac{3}{4}+\epsilon})$ time and supports updates in $\mathcal{O}(\log^2 m)$ time, where m is the number of points stored. It follows that our representation of V_q uses $\mathcal{O}(n \log^2 n)$ space, and supports updates in amortized $\mathcal{O}(\log^3 n)$ time.

Since all nodes in the data structure have constant in-degree we can make it partially persistent at the cost of $\mathcal{O}(\log^3 n)$ space per update [21]. It follows we can represent the visibility polygons for all cells in \mathcal{S}_ℓ in $\mathcal{O}(n^2 \log^3 n)$ space.

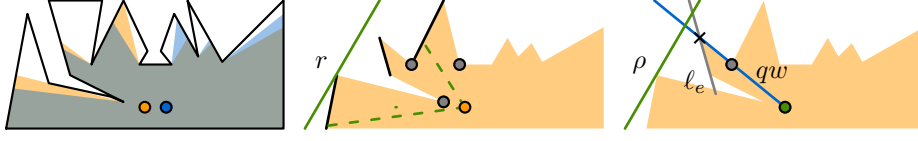
Querying. Given a query q, r we test if the segment r intersects V_q . The main idea is to query our data structure for the part of V_q in the wedge defined by q and r . We then extend r into a line ρ , and test if this line separates a vertex of V_q in this wedge from q . The segment r intersects V_q , and thus there is a time at which r is visible from q , if and only if this is the case.

We can obtain the part of V_q that lies in the wedge defined by q and r , represented by $\mathcal{O}(\log^2 n)$ $BB[\alpha]$ -tree nodes. For each of these nodes we query the associated data structures to test if the halfspace ρ^{-q} not containing q is empty. We can directly query the data structure storing the fixed vertices with ρ^{-q} . To test if there is a variate vertex that lies in ρ^{-q} we map it to a halfspace in \mathbb{R}^8 using a function g .

► **Lemma 11.** *There are functions f and g such that f maps each variate vertex (e, w) to a point $f(e, w) \in \mathbb{R}^8$ and g maps each ρ^{-q} to a halfspace $g(\rho^{-q})$ in \mathbb{R}^8 such that $f(e, w) \in g(\rho^{-q})$ if and only if the location of the variate vertex (e, w) in V_q lies in ρ^{-q} .*

Proof. Let $q = (a_3, a_4)$, and let $\rho = \{x, y \mid 0 = a_1x - a_2 - y\}$ be the supporting line of the trajectory of r . We describe the construction for the case that q lies below ρ and ρ is non-vertical. The other cases can be handled analogously.

23:14 Trajectory Visibility



■ **Figure 10** (left) Two points in orange and blue, which have the same implicit visibility polygon, however there could be several placement of r such that r only intersects one of the two explicit visibility polygons. (middle) Entity r in green, q in orange and the chain of uncertain edges. (right) An illustration of the geometric argument. We compute the intersection between ℓ_e and qw and check if that point lies below or above ρ .

Refer to Figure 10 (right) for an illustration of the proof. For each variate vertex (e, w) in a chain we know that the line qw intersects the line ℓ_e supporting e on the domain of e (this property is guaranteed since (e, w) is a vertex of V_q). Moreover, it is guaranteed that the intersection point between qw and ρ lies on the trajectory of r . It follows that q can see r if and only if, the intersection point (x, y) between qw and ℓ_e lies above ρ . Given ρ, q, w and ℓ_e , we can algebraically compute this intersection point (x, y) . We then substitute the equation for (x, y) into the equation for ρ and the point (x, y) lies above this line if and only if the result is greater than 0:

$$wq := \left\{ x, y \mid 0 = \frac{x_4 - a_4}{x_3 - a_3}x - \frac{x_4 - a_4}{x_3 - a_3}x_3 + x_4 \right\}$$

The lines wq and ℓ_e intersect at the point where their y -coordinate is equal and therefore:

$$\begin{aligned} x_1x - x_2 &= \frac{x_4 - a_4}{x_3 - a_3}x - \frac{x_4 - a_4}{x_3 - a_3}x_3 + x_4 \\ (x_3 - a_3)(x_1x - x_2) &= (x_4 - a_4)x - (x_4 - a_4)x_3 + (x_3 - a_3)x_4 \\ (x_3 - a_3)x_1x - (x_4 - a_4)x &= x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4 \end{aligned}$$

From this equation we can extract the coordinates of the intersection point (x, y) between wq and ℓ_e :

$$\begin{aligned} x &= \frac{x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4}{(x_3 - a_3)x_1 - (x_4 - a_4)} \\ y &= x_1 \frac{x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4}{(x_3 - a_3)x_1 - (x_4 - a_4)} - x_2 \end{aligned}$$

Lastly we substitute the algebraic expression for (x, y) into the formula for ρ and we linearize the predicate:

$$\begin{aligned} 0 &\geq a_1(x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4) - \\ &\quad x_2 - x_1(x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4) + x_2 \\ 0 &\geq [-a_1a_3](x_2) + [a_3](x_1x_2) + [a_1](x_2x_3) + [a_1a_4](x_3) + \\ &\quad [-a_4](x_1x_3) + [-a_1a_3](x_4) + [a_3](x_1x_4) + [-1](x_1x_2x_3) \end{aligned}$$

Thus we found a predicate $F(\vec{x}, \vec{a})$ with:

$$\begin{aligned} (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8) &= (x_2, x_1x_2, x_2x_3, x_3, x_1x_3, x_4, x_1x_4, x_1x_2x_3) \\ (g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8) &= (0, -a_1a_3, a_3, a_1, a_1a_4, -a_4, -a_1a_3, a_3, -1) \end{aligned}$$

It follows that we can map every variate vertex to a point in \mathbb{R}^8 using the f -maps provided by the predicate. Any query consisting of the halfplane ρ^{-q} defined by ρ and q gets mapped to a halfspace in \mathbb{R}^8 . The halfplane ρ^{-q} contains the variate vertex defined by q , w , and e if and only if its representative point lies in this halfspace. ◀

This theorem now immediately follows:

► **Theorem 12.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n^2 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^2 \log^3 n)$ time.*

Polygonal domains. In case P is a polygonal domain we use a similar approach; we build a subdivision \mathcal{S} in which all points in a cell have a visibility polygon V_q with the same combinatorial structure, and then traverse \mathcal{S} while maintaining V_q in a partially persistent data structure. To obtain \mathcal{S} we simply take all $\mathcal{O}(n^2)$ lines defined by pairs of polygon vertices. The subdivision \mathcal{S} is the arrangement of these lines and has $\mathcal{O}(n^4)$ complexity. We obtain a traversal of \mathcal{S} by computing an Euler tour of a spanning tree of the dual of \mathcal{S} . We conclude

► **Theorem 13.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $\mathcal{O}(n^4 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^4 \log^3 n)$ time.*

Next, we investigate the variants where the entities can walk through edges of P , and/or where P is a polygonal domain. We switch to different techniques, focused on representing the set of all potential visibility polygons in P compactly. Our representation supports efficient intersection queries of the visibility polygon of a specific point q with the trajectory of r . We obtain the following result.

► **Theorem 14.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n^2 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^2 \log^3 n)$ time. If P is a polygonal domain, we can still obtain this result using $\mathcal{O}(n^4 \log^3 n)$ space and $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ query time.*

6 A data structure for queries in polygonal domains

Finally, in Appendix C we consider the most general version of the problem, where both entities are moving in a polygonal domain (and/or can move through walls). Now the set of visibility lines from one trajectory to the other does not form a single hourglass but consist of linearly many hourglasses, thus the visibility glass does not dualize to a convex polygon. As a result, the solutions based on partition trees no longer lead to sublinear query times. Instead, we discuss a different approach based on cutting trees, leading to polylogarithmic query time at the cost of much higher space usage. We obtain the following result.

► **Theorem 15.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $\mathcal{O}(n^{3k})$, for some sufficiently large constant k , that allows us to answer trajectory visibility queries in $\mathcal{O}(\log^k n)$ time. Building the data structure takes $\mathcal{O}(n^{3k})$ time.*

References

- 1 P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, April 1995. doi:10.1007/BF01293483.
- 2 Pankaj K. Agarwal, Jivri Matoušek, and Micha Sharir. On range searching with semialgebraic sets. II. *SICOMP*, 42(6):2039–2062, 2013. doi:10.1137/120890855.
- 3 Boris Aronov, Leonidas J Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *DCG*, 27(4):461–483, 2002.
- 4 Michael Ben-Or. Lower bounds for algebraic computation trees. In *STOC*, pages 80–86, 1983.
- 5 Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Comput. Geom.*, 41(3):111–125, 2008. doi:10.1016/j.comgeo.2007.10.003.
- 6 Marshall Bern, David Dobkin, David Eppstein, and Robert Grossman. Visibility with a moving point of view. *Algorithmica*, 11(4):360–378, 1994.
- 7 P. Bovet and S. Benhamou. Spatial analysis of animals’ movements using a correlated random walk model. *J. Theoretical Biology*, 131(4):419–433, 1988. doi:10.1016/S0022-5193(88)80038-9.
- 8 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *FOCS*, pages 617–626, 2002.
- 9 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *IJCGA*, 21(03):253–282, 2011. doi:10.1142/S0218195911003652.
- 10 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *DCG*, 57(1):164–178, 2017.
- 11 Timothy M Chan. Optimal partition trees. *DCG*, 47(4):661–690, 2012.
- 12 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *DCG*, 9(2):145–158, 1993.
- 13 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.
- 14 Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *DCG*, 4:551–581, 1989. doi:10.1007/BF02187747.
- 15 Vasek Chvátal. A combinatorial theorem in plane geometry. *JCTB*, 18(1):39–41, 1975.
- 16 Mark De Berg. *Ray shooting, depth orders and hidden surface removal*, volume 703. Springer Science & Business Media, 1993.
- 17 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry*. Springer, 1997.
- 18 Yago Diez, Matias Korman, André van Renssen, Marcel Roeloffzen, and Frank Staals. Kinetic all-pairs shortest path in a simple polygon. *EuroCG*, pages 21–24, 2017. abstract.
- 19 David P. Dobkin and David G. Kirkpatrick. Fast detection of polyhedral intersection. *TCS*, 27(3):241–253, 1983. ICALP. doi:10.1016/0304-3975(82)90120-7.
- 20 Somayeh Dodge, Robert Weibel, and Ehsan Foroootan. Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Computers, Environment and Urban Systems*, 33(6):419–434, 2009.
- 21 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *JCSS*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.
- 22 Frédo Durand. A multidisciplinary survey of visibility. *ACM Siggraph course notes Visibility, Problems, Techniques, and Applications*, 2000.
- 23 Steve Fisk. A short proof of chvátal’s watchman theorem. *JCTB*, 24(3):374, 1978.
- 24 Leila De Florian and Paola Magillo. Algorithms for visibility computation on terrains: a survey. *Environ. Plann. B*, 30(5):709–728, 2003.
- 25 S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *KDD*, pages 63–72, 1999.

- 26 Subir K. Ghosh and Partha P. Goswami. Unsolved problems in visibility graphs of points, segments, and polygons. *CSUR*, 46(2):22:1–22:29, December 2013. doi:10.1145/2543581.2543589.
- 27 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS.*, pages 1362–1370. Springer, 2017. doi:10.1007/978-3-319-17885-1_823.
- 28 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.
- 29 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. doi:10.1007/BF01840360.
- 30 Eliezer Gurarie, Russel D. Andrews, and Kristin L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology Letters*, 12(5):395–408, 2009. doi:10.1111/j.1461-0248.2009.01293.x.
- 31 Patrick Laube, Marc J. van Kreveld, and Stephan Imfeld. Finding REMO - detecting relative motion patterns in geospatial lifelines. In *SDH*, pages 201–215, 2004. doi:10.1007/3-540-26772-7_16.
- 32 J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 593–604, 2007.
- 33 Xiaojie Li, Xiang Li, Daimin Tang, and Xianrui Xu. Deriving features of traffic flow around an intersection from trajectories of vehicles. In *Proc. 18th International Conference on Geoinformatics*, pages 1–5. IEEE, 2010.
- 34 J. Matoušek. Efficient partition trees. *DCG*, 1992.
- 35 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *DCG*, 10(2):157–182, 1993. doi:10.1007/BF02573972.
- 36 Ether Moet. *Computation and complexity of visibility in geometric environments*. PhD thesis, Utrecht University, 2008.
- 37 Ketan Mulmuley. Hidden surface removal with respect to a moving view point. In *STOC*, pages 512–522. ACM, 1991.
- 38 J. Nievergelt and E. M. Reingold. Binary Search Trees of Bounded Balance. *SICOMP*, 2(1):33–43, 1973.
- 39 Joseph O’Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.
- 40 Michel Pocchiola and Gert Vegter. The visibility complex. *IJCGA*, 6(3):279–308, 1996.
- 41 Andreas Stohl. Computation, accuracy and applications of trajectories – a review and bibliography. *Atmospheric Environment*, 32(6):947–966, 1998. doi:10.1016/S1352-2310(97)00457-3.
- 42 Emo Welzl. Constructing the visibility graph for n -line segments in $o(n^2)$ time. *IPL*, 20(4):167–171, 1985.

A

 Transforming two entities into an algebraic curve.

Throughout this paper, we study the line-of-sight between two entities that each move along a linear trajectory (possibly at different but constant speeds) during the time $t \in [0, 1]$. Consider the line $g(t)$ through the two entities at time t . We can dualize $g(t)$ to a point using classical point-line dualization. In Section 3 we claim that the continuous dualization of the line through the two entities traces a degree-2 curve segment denoted by γ in the dual. Here we show why this is the case:

► **Lemma 1.** *The segment γ is a segment of a quadratic curve with 5 degrees of freedom.*

Proof. For algebraic convenience we say that entity q walks from (a_1, a_2) to $(a_1 + a_3, a_2 + a_4)$ and that entity r walks from (a_5, a_6) to $(a_5 + a_7, a_6 + a_8)$. Note that the speed of entity q is $\|(a_3, a_4)\|$ and that the speed of entity r is $\|(a_7, a_8)\|$. We can parametrize the position of entity q and r on the time $t \in [0, 1]$ as follows:

$$q(t) = \begin{pmatrix} x_{q(t)} \\ y_{q(t)} \end{pmatrix} = \begin{pmatrix} a_1 + a_3t \\ a_2 + a_4t \end{pmatrix} \quad r(t) = \begin{pmatrix} x_{r(t)} \\ y_{r(t)} \end{pmatrix} = \begin{pmatrix} a_5 + a_7t \\ a_6 + a_8t \end{pmatrix} \quad (1)$$

At all times, the line $g(t)$ is the line through the points $q(t)$ and $r(t)$. We say that at all times, $g(t)$ has slope and offset $(\alpha(t), \beta(t))$. The parametrisation of $g(t)$ then becomes:

$$g(t) = \begin{pmatrix} \alpha(t) \\ \beta(t) \end{pmatrix} = \begin{pmatrix} \frac{y_{r(t)} - y_{q(t)}}{x_{r(t)} - x_{q(t)}} \\ \alpha(t) \cdot x_{q(t)} - y_{q(t)} \end{pmatrix} = \begin{pmatrix} \frac{a_6 - a_2 + (a_8 - a_4)t}{a_5 - a_1 + (a_7 - a_3)t} \\ \alpha(t)(a_1 - a_3t) - a_2 - a_4t \end{pmatrix} \quad (2)$$

If the time t lies between 0 and 1, this parametric equation traces our curve segment γ and if we take t over all of \mathbb{R} , the parametric equation traces a full curve which we denote by Γ . To show the degree of the curve Γ we rewrite the parametrized curve to a canonical form that drops the dependence on t . First we take the formula for the β -coordinate and isolate t :

$$t = \frac{\alpha(t)a_1 - a_2 - \beta(t)}{\alpha(t)a_3 + a_4}$$

We then take the formula for the α -coordinate and remove the fraction by multiplying both sides with $((a_5 - a_1) + (a_7 - a_3)t)$. Note that this expression is only zero if the line $g(t)$ is vertical. Refer to below on how to avoid such degeneracies.

$$\alpha(t)(a_5 - a_1) + \alpha(t)(a_7 - a_3)t = (a_6 - a_2) + (a_8 - a_4)t$$

We substitute the value for t into this equation, and remove the fraction by multiplying both sides with $(\alpha(t)a_3 + a_4)$:

$$\begin{aligned} & \alpha(t)(a_5 - a_1) \cdot (\alpha(t)a_3 + a_4) + \alpha(t)(a_7 - a_3) \cdot (\alpha(t)a_1 - a_2 - \beta(t)) = \\ & (a_6 - a_2) \cdot (\alpha(t)a_3 + a_4) + (a_8 - a_4) \cdot (\alpha(t)a_1 - a_2 - \beta(t)) \Rightarrow \\ & \alpha(t)^2 a_3(a_5 - a_1) + \alpha(t)a_4(a_5 - a_1) + \\ & \alpha(t)^2 a_1(a_7 - a_3) - \alpha(t)a_2(a_7 - a_3) - \alpha(t)\beta(t)(a_7 - a_3) = \\ & \alpha(t)a_3(a_6 - a_2) + a_4(a_6 - a_2) + \alpha(t)a_1(a_8 - a_4) - a_2(a_8 - a_4) - \beta(t)(a_8 - a_4) \end{aligned}$$

Lastly we show that this equation provides a linearization as defined in Appendix B by separating polynomials based on $\alpha(t)$ and $\beta(t)$ from polynomials based on $a_1 \dots a_8$.

$$[\alpha(t)^2](a_3(a_5 - a_1) + a_1(a_7 - a_3)) + \quad (3)$$

$$[\alpha(t)](a_4(a_5 - a_1) - a_2(a_7 - a_3) - a_3(a_6 - a_2) - a_1(a_8 - a_4)) - \quad (4)$$

$$[\alpha(t)\beta(t)](a_7 - a_3) + [\beta(t)](a_8 - a_4) + \alpha(t)(a_1(a_8 - a_4)) [1](a_2(a_8 - a_4) - a_4(a_6 - a_2)) \quad (5)$$

◀

Dealing with degeneracies in this paper. Observe that in Equation 2 it is possible to divide by zero. Note that this occurs only if there is a moment in time where the line-of-sight between the two entities is a vertical segment. This situation occurs throughout this paper, and in this case the dual of their line-of-sight is also not well defined. Generally we cannot construct the dual of a visibility glass if in the primal it contains any vertical lines of sight.

This is a common degeneracy with visibility queries and dualization algorithms in computational geometry and it can be solved as follows: For any two (input or query) segments, one can split the time interval into two disjoint intervals, such that in one interval the segment is never vertical and in the second interval the segment is never horizontal. Note that only one split is needed, which can be calculated in constant time, because the entities move linearly. Therefore we solve the algorithmic question or the data structure question by solving two separate inputs or queries, where for the time interval that can contain vertical but not horizontal lines-of-sight, we consider a rotated version of the plane.

Similarly, whenever we consider any visibility glass, we split it into lines that are steeper than $y = x$ and lines that are not. One such set will never contain horizontal lines and the other will never contain vertical lines. Therefore, for each set of lines we can construct the dual of the visibility glass in an appropriate rotated version of the plane.

B Semi-algebraic range searching

Throughout this paper we make extensive use of the semi-algebraic (or linearization) techniques from Agarwal et al. [2]. We describe the technique in detail for completeness.

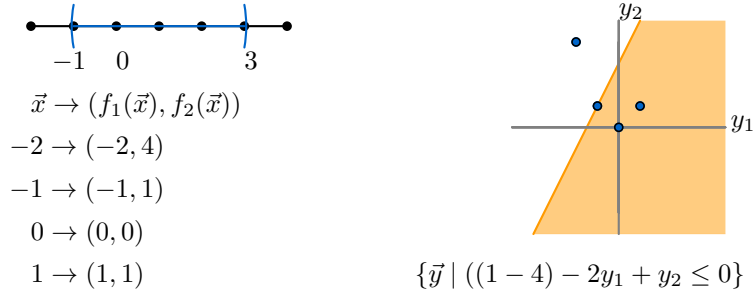
Let X be a set of n geometric objects in \mathbb{R}^d , where each object is parametrized by a vector \vec{x} . If X is a set of points, then the most natural parametrization is a vector of its coordinates. But X could be a more complicated algebraic object such as a line, in that case the most natural parametrization is a two-dimensional vector containing its slope and offset.

We denote by Γ the family of geometric regions (called semi-algebraic ranges) in \mathbb{R}^d where each region $G \in \Gamma$ is bounded by an algebraic curve γ which is parametrized by a vector \vec{a} . Two examples of such a family is the set of all disks and the set of all disks of radius 1. An arbitrary disk can be represented as a vector in many ways: three non-colinear points define a unique circle so one could represent a circle as a six-dimensional vector which specifies the coordinates of these points. However the larger the representation vector, the more complicated the linearization process becomes. The most efficient representation of a circle is by a three-dimensional vector specifying its center and radius. The family of disks of radius 1 has fewer degrees of freedom than the family of all disks, and thus their representation can be more efficiently represented (e.g. as a 2-dimensional vector specifying only its center).

Given X and Γ , we are interested in preprocessing X such that for any range $G \in \Gamma$, we can report which objects of X intersect G . To accomplish this, we first want to derive what we have dubbed a *predicate function* $F(\vec{x}, \vec{a}) \leq 0$. The predicate function F takes any instance of X (parametrized by \vec{x}) and any instance of Γ (parametrized by \vec{a}) and outputs a real number. The object intersects the range if and only if the output value is lesser then or equal to zero. At this point we present an example: let X be a set of two-dimensional points parametrized by $\vec{x} = (x_1, x_2)$ and Γ be the set of arbitrary disks, each parametrized by $\vec{a} = (a_1, a_2, r)$. The disk parametrized by \vec{a} has center (a_1, a_2) and radius r . Any point (x_1, x_2) is contained in this disk if and only if $F(\vec{x}, \vec{a}) = (a_1 - x_1)^2 + (a_2 - x_2)^2 - r^2 \leq 0$ and thus we have found our predicate function.

The predicate function $F(\vec{x}, \vec{a})$ could be seen as a map from the parameter space of our intersection problem to the boolean space $\{0, 1\}$ and thus F partitions our parameter space into areas where the answer is *yes* or areas where the answer is *no*. The idea behind

23:20 Trajectory Visibility



■ **Figure 11** Consider the family Γ of 1-dimensional unit disks. We can parametrize their border by supplying a 1-dimensional center point and a radius $\vec{a} = (a_1, a_2)$. Any 1-dimensional point $\vec{x} = (x_1)$ is contained in a disk $G \in \Gamma$ if and only if $(a_1 - x_1)^2 - a_2^2 \leq 0 \Rightarrow [a_1^2 - a_2^2] + [-2a_1](x_1) + (x_1^2) \leq 0$. If we linearize this predicate function, we get that $(g_0, g_1, g_2) = (a_1^2 - a_2^2, -2a_1, 1)$ and $(f_1, f_2) = (x_1, x_1^2)$. It follows that any intersection query between a disk G and a set of points X can be answered using a halfspace emptiness query. In the figure we show an example for the points $(-2, -1, 0, 1)$ and the disk G with center 1 and radius 2.

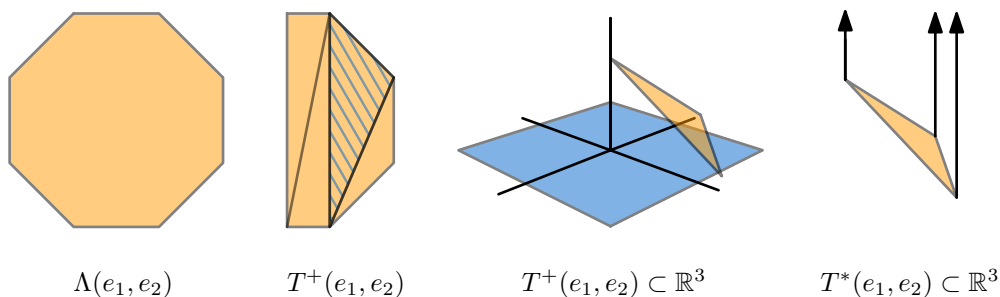
semi-algebraic range searching is that we search for an intersection in this parameter space, as opposed to searching in the space where our problem lives. However, the border of these areas do not have to be particularly nice. In our example, each of the *yes* areas is bounded by a quadratic surface. This is where *linearization* comes in. We transform the parameter space through a polynomial map into a k -dimensional space where the boundary of the *yes* spaces becomes a linear-complexity surface. At this point, we wish to mention that in full generality, this map does not have to be a polynomial map. But for the purpose of this paper, we can restrict the results from [2] so that at all times, all the functions that we regard are bounded degree polynomials.

Given such a linearization, we can solve our problem using halfspace range searching. Specifically, we rewrite the function $F(\vec{x}, \vec{a})$ to the form: $F(\vec{x}, \vec{a}) = g_0(\vec{a}) + \sum_{i=1}^k g_i(\vec{a})f_i(\vec{x})$ where f_i and g_i are polynomials dependent only on \vec{x} and \vec{a} respectively. In our example we had: $F(\vec{x}, \vec{a}) = (a_1 - x_1)^2 + (a_2 - x_2)^2 - r^2 \leq 0$. To linearize this function we need to first expand the squares: $F(\vec{a}, \vec{x}) = a_1^2 - 2a_1x_1 + x_1^2 + a_2^2 - 2a_2x_2 + x_2^2 - r^2$. This immediately gives a straight-forward linearization of seven terms. However, we can reduce the number of terms by grouping variables and writing: $F(\vec{x}, \vec{a}) = [a_1^2 + a_2^2 - r^2] + [-2a_1](x_1) + [-2a_2](x_2) + [1](x_1^2 + x_2^2)$ and obtain a linearization where: $(g_0, g_1, g_2, g_3) = (a_1^2 + a_2^2 - r^2, -2a_1, -2a_2, 1)$ and $(f_1, f_2, f_3) = (x_1, x_2, x_1^2 + x_2^2)$. We get the term g_0 (which is not attached to any polynomial dependent on x) “for free” and this thus becomes a three-dimensional linearization.

Agarwal et al. prove that you can map any d -dimensional point \vec{x} to the k -dimensional point $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \dots, f_k(\vec{x}))$, and any query range to the k -dimensional halfspace $G(\vec{a}) = \left\{ \vec{y} \in \mathbb{R}^k \mid g_0(\vec{a}) + \sum_{i=1}^k g_i(\vec{a})y_i \leq 0 \right\}$ and that \vec{x} intersects G if and only if $f(\vec{x})$ is contained in $G(\vec{a})$. Consider our example. The map f that we found, is the well-known paraboloid projection: We map all the two-dimensional points onto a three-dimensional paraboloid and they are contained in a circle $G \in G$ if and only if the points lie within a halfplane cutting the paraboloid. Refer to Figure 11 for an even shorter example.

C Two moving entities in a polygonal domain

We investigate the two cases where (1) the entities can walk through edges of P and (2) P is a polygonal domain simultaneously. Let k be an unspecified constant. We prove that it is possible to preprocess a polygonal domain P with n vertices in $\mathcal{O}(n^k)$ time, such that for any



■ **Figure 12** Unfortunately we cannot draw figures in four dimensions. So we illustrate the mapping from a visibility glass to a three-dimensional volume instead using the map $F_1(a, b)$ based on only the edge e_1 and not the edge e_2 .

two entities q and r that each traverse a line segment possibly through edges of P , we can determine if there is a moment when q and r are mutually visible in sub-linear time. Our approach almost certainly does not generate an optimal solution with respect to its space requirement and query time. However, it is a non-trivial proof that sub-linear query times are achievable. We obtain these results, by transforming the visibility query in the plane into an intersection query in \mathbb{R}^4 and then immediately applying semi-algebraic range searching.

Let e_1 and e_2 be two edges of P and denote their visibility glass by $L(e_1, e_2)$. We say that e_1 lies on the line $y = x_1x - x_2$ and e_2 lies on the line $y = x_3x - x_4$. Let $\ell :: y = ax - b$ be a line through $L(e_1, e_2)$ with positive slope and let e_1 lie below e_2 along ℓ . The x -coordinate of the intersection between ℓ and the edges is given by $F_1(a, b) = \frac{b-x_2}{a-x_1}$ and $F_2(a, b) = \frac{b-x_4}{a-x_3}$.

► **Observation 3.** *Suppose we have a segment on ℓ that starts at the point q and ends at the point r then this segment is contained in $L(e_1, e_2)$ if and only if $F_1(a, b) \leq x_q \leq x_r \leq F_2(a, b)$.*

This observation leads to the following approach for detecting if there is a line-of-sight between q and r : we construct for each of the n^2 pairs of edges e_1, e_2 , the two-dimensional area $\Lambda^+(e_1, e_2)$ which is the dualization of all positive slope lines through the visibility glass between e_1 and e_2 . For reasons that will become apparent later, we triangulate $\Lambda^+(e_1, e_2)$. Consider any triangle $T^+(e_1, e_2)$ of this triangulation. It represents a collection of positive-slope lines that stab through the visibility glass $L(e_1, e_2)$. We lift $T^+(e_1, e_2)$ to a two-dimensional surface to \mathbb{R}^4 with the map that takes a point (a, b) in $T^+(e_1, e_2)$ and that maps it to the point $(a, b, F_1(a, b), F_2(a, b))$. This creates a two-dimensional surface in \mathbb{R}^4 which has a constant description size. Now consider the following cylinder-like volume in \mathbb{R}^4 : $T^*(e_1, e_2) = \{(a, b, c, d) \in \mathbb{R}^4 \mid (a, b, c', d') \in T^+(e_1, e_2) \wedge c' \leq c \wedge c \leq d \wedge d \leq d'\}$. Any point $(a, b, c, d) \in T^*(e_1, e_2)$, represents a line segment that lies on the line $y = ax - b$, whose start point lies below its end point, and whose start and end points lie between e_1 and e_2 . Refer to Figure 12 for an example of this transformation in \mathbb{R}^3 :

Let q and r be given as two line-segment trajectories that do not intersect (if they do intersect, we can always split the visibility query into constantly many visibility queries). Note that we can split q and r into two sub-segments q' and r' where the entity q always has a lower y -coordinate than entity r and where the line through q and r has positive slope. We denote by γ' the continuous dualization of q' and r' according to equation 3. The two-dimensional curve segment γ' can be mapped to a curve segment in \mathbb{R}^4 with a mapping that is very similar to our earlier transformation. Each point $(a, b) \in \gamma'$ represents a segment following the line $y = ax - b$ between q and r where q must lie below r . We map the point (a, b) to the point (a, b, c, d) where c and d are the x -coordinates of intersections of the line

23:22 Trajectory Visibility

$y = ax - b$ with the trajectories of q and r respectively. Coincidentally, this means that we are mapping a point (a, b) to (a, b, x_q, x_r) . If γ' intersects $T^*(e_1, e_2)$ then at the time of intersection, the two entities realise a line segment that lies within the visibility glass $L(e_1, e_2)$ and it follows that the entities are mutually visible. Both the volume T^* and the query segment γ' can be parametrized with a constant-length parameter, so the predicate that tests their intersection can be linearized to a constant k number of terms.


It follows that we can create a data structure that stores $\mathcal{O}(n^3)$ of these volumes (one for each triangle in both the positive and negative visibility glasses), each represented by a point in \mathbb{R}^k . Specifically, we build a cutting tree on these $\mathcal{O}(n^3)$ points in $\mathcal{O}(n^{3k})$ time. A query supplied as two segments q and r , can be cut into constantly many pairs of segments where for each pair of segments either q is above r or vice versa. For each pair of segments, we derive its corresponding k -dimensional halfspace in $\mathcal{O}(k)$ time and we query the cutting tree in $\mathcal{O}(\log^k n)$ time to see if the halfspace is empty. There is no time when the two entities are mutually visible if and only if each of these queries reports an empty halfspace. Thus we conclude:

► **Theorem 15.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $\mathcal{O}(n^{3k})$, for some sufficiently large constant k , that allows us to answer trajectory visibility queries in $\mathcal{O}(\log^k n)$ time. Building the data structure takes $\mathcal{O}(n^{3k})$ time.*

Simplifying Activity-On-Edge Graphs

David Eppstein

University of California, Irvine, CA, United States
eppstein@uci.edu

Daniel Frishberg 

University of California, Irvine, CA, United States
dfrishbe@uci.edu

Elham Havvaei 

University of California, Irvine, CA, United States
ehavvaei@uci.edu

Abstract

We formalize the simplification of *activity-on-edge* graphs used for visualizing project schedules, where the vertices of the graphs represent project milestones, and the edges represent either tasks of the project or timing constraints between milestones. In this framework, a timeline of the project can be constructed as a leveled drawing of the graph, where the levels of the vertices represent the time at which each milestone is scheduled to happen. We focus on the following problem: given an activity-on-edge graph representing a project, find an equivalent activity-on-edge graph—one with the same critical paths—that has the minimum possible number of milestone vertices among all equivalent activity-on-edge graphs. We provide an $O(mn^2)$ -time algorithm for solving this graph minimization problem.

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

Keywords and phrases directed acyclic graph, activity-on-edge graph, critical path, project planning, milestone minimization, graph visualization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.24

Related Version A related version of the paper is available at <https://arxiv.org/abs/2002.01610>.

Funding This work was supported in part by NSF grants CCF-1618301 and CCF-1616248.

1 Introduction

The *critical path method* is used in project modeling to describe the tasks of a project, along with the dependencies among the tasks; it was originally developed as PERT by the United States Navy in the 1950s [18]. A dependency graph is used to identify bottlenecks, and in particular to find the longest path among a sequence of tasks, where each task has a required length of time to complete (this is known as the *critical path*).

In this paper we consider a phase in planning a given project in which we do not yet know the time lengths of each task. We are interested in the problem of visualizing an abstract timeline of the potential critical paths (i.e., paths that could be critical depending on the lengths of the tasks) of the project, represented abstractly as a partially ordered set of tasks. The most common method of visualizing partially ordered sets, as an *activity-on-node graph* (a transitively reduced directed acyclic graph with a vertex for each task) is unsuitable for this aim, because it represents each task as a point instead of an object that can extend over a span of time in a timeline. To resolve this issue, we choose to represent each task as an edge in a directed acyclic graph. In this framework, the endpoints of the task edges have a natural interpretation, as the *milestones* of the project to be scheduled. Additional *unlabeled edges* do not represent tasks to be performed within the project, but constrain certain pairs of milestones to occur in a certain chronological order. The resulting *activity-on-edge graph*



© David Eppstein, Daniel Frishberg, and Elham Havvaei;
licensed under Creative Commons License CC-BY

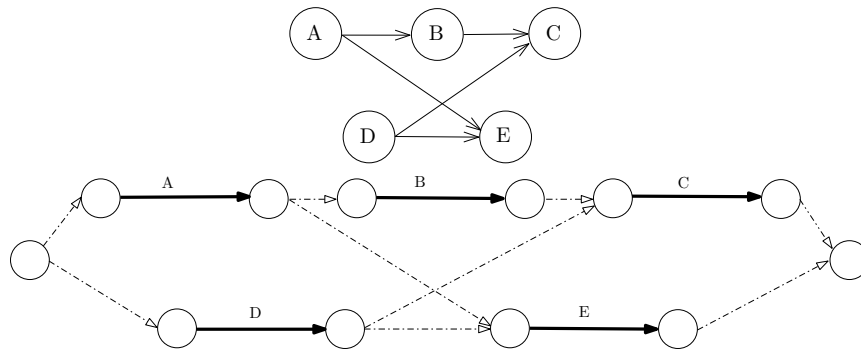
17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 24; pp. 24:1–24:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** An activity-on-node graph, above, and its naively expanded activity-on-edge graph, below, with solid arrows as task edges and empty arrows as unlabeled edges.

can then be drawn in standard upward graph drawing style [1, 6, 8, 10, 11]. Alternatively, once the lengths of the tasks are known and the project has been scheduled, this graph can be drawn in leveled style [12, 16], where the level of each milestone vertex represents the time at which it is scheduled.

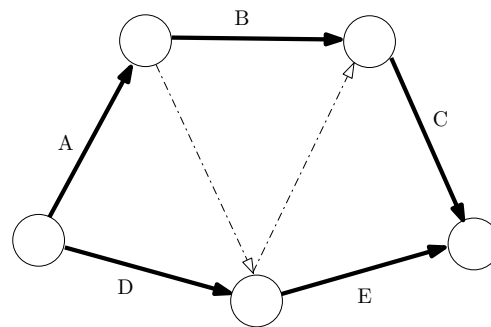
It is straightforward to expand an activity-on-node graph into an activity-on-edge graph by expanding each task vertex of the activity-on-node graph into a pair of milestone vertices connected by a task edge, with the starting milestone of each task retaining all of the incoming unlabeled edges of the activity-on-node graph and the ending milestone retaining all of the outgoing edges. It is convenient to add two more milestones at the start and end of the project, connected respectively to all milestones with no incoming edges and from all milestones with no outgoing edges. The size of the resulting activity-on-edge graph is linear in the size of the activity-on-node graph. An example of such a transformation is depicted in Figure 1.

However, the graphs that result from this naive expansion are not minimal. Often, one can merge some pairs of milestones (for instance the ending milestone of one task and the starting milestone of another task) to produce a simpler activity-on-edge graph (such as the one for the same schedule in Figure 2). Despite having fewer milestones, this simpler graph can be equivalent to the original, in the sense that its potential critical paths (maximal sequences of tasks that belong to a single path in the graph) are the same. By being simpler, this merged graph should aid in the visualization of project schedules. In this paper we formulate and provide an $O(mn^2)$ -time algorithm (where n is the number of milestones and m is the number of unlabeled edges) for the problem of optimal simplification of activity-on-edge graphs.

1.1 New Results

We describe a polynomial-time algorithm that, given an activity-on-edge graph (i.e., a directed acyclic graph with a subset of its edges labeled as tasks), produces a directed acyclic graph that preserves the potential critical paths of the graph and has the minimum possible number of vertices among all critical-path-preserving graphs for the given input. Our algorithm is agnostic about the weights of the tasks. In more general terms, the resulting graph has the following properties:

- The task edges in the given graph correspond one-to-one with the task edges in the new graph.



■ **Figure 2** A simplification of the graph from Figure 1.

- The new graph has the same dependency (reachability) relation among task edges as the original graph.
- The new graph has the same potential critical paths as the original graph.
- The number of vertices of the graph is minimized among all graphs with the first three properties.

Our algorithm repeatedly applies a set of local reduction rules, each of which either merges a pair of adjacent vertices or removes an unlabeled edge, in arbitrary order. When no rule can be applied, the algorithm outputs the resulting graph.

We devote the rest of this section to related work and then describe the preliminaries in Section 2. We then present the algorithm in Section 3 and show in Section 4 that its output preserves the potential critical paths of the input, and in Section 5 that it has the minimum possible number of vertices. We also show that the output is independent of the order in which the rules are applied. We discuss the running time in Section 6 and conclude with Section 7.

1.2 Related work

Constructing clear and aesthetically pleasing drawings of directed acyclic graphs is an old and well-established task in graph drawing, with many publications [5, 6, 13, 19]. The work in this line that is most closely relevant for our work involves upward drawings of unweighted directed acyclic graphs [1, 8, 10, 11] or leveled drawings of directed acyclic graphs that have been given a level assignment [12, 16] (an assignment of a y -coordinate to each vertex, for instance representing its height on a timeline).

Although multiple prior publications use activity-on-edge graphs [3, 7, 15, 17] and even consider graph drawing methods specialized for these graphs [20], we have been unable to locate prior work on their simplification. This problem is related to a standard computational problem, the construction of the transitive reduction of a directed acyclic graph or equivalently the covering graph of a partially ordered set [2]. We note in addition our prior work on augmenting partially ordered sets with additional elements (preserving the partial order on the given elements) in order to draw the augmented partial order as an upward planar graph with a minimum number of added vertices [9].

The PERT method may additionally involve the notion of “float”, in which a given task may be delayed some amount of time (depending on the task) without any effect on the overall time of the project [4, 14]. We do not consider constraints of this form in the present work, although the unlabeled edges of our output can in some sense be seen as serving a similar purpose.

2 Preliminaries

We first define an activity-on-edge graph. The graph can be a multigraph to allow tasks that can be completed in parallel to share both a start and end milestone when possible.

► **Definition 1.** An activity-on-edge graph (AOE) is a directed acyclic multigraph $G = (V, E)$, where a subset of the edges of E , denoted \mathcal{T} , are labeled as task edges. The labels, denoting tasks, are distinct, and we identify each edge in \mathcal{T} with its label.

► **Definition 2.** Given an AOE G with tasks \mathcal{T} , for all $T \in \mathcal{T}$, let $\text{St}_G(T)$ be the start vertex of T , and let $\text{End}_G(T)$ be the end vertex of T .

When the considered graph is clear from context, we omit the subscript G and write $\text{St}(T)$ and $\text{End}(T)$. It may be that $\text{St}(T) = \text{St}(T')$, or $\text{End}(T) = \text{End}(T')$, or $\text{End}(T) = \text{St}(T')$ with $T \neq T'$.

To define potential critical paths formally, we introduce the following notation.

► **Definition 3.** Given an AOE G with tasks \mathcal{T} , for all $T, T' \in \mathcal{T}$ with $T \neq T'$, say that T has a path to T' in G if there exists a path from $\text{End}(T)$ to $\text{St}(T')$, or if $\text{End}(T) = \text{St}(T')$, and write $T \rightsquigarrow_G T'$.

► **Definition 4.** Given an AOE G with tasks \mathcal{T} , a potential critical path is a sequence of tasks $P = (T_1, \dots, T_k)$, where for all $i = 1, \dots, k - 1$, $T_i \rightsquigarrow_G T_{i+1}$, and where P is not a subsequence of any other sequence with this property.

Our algorithm will apply a set of transformation rules to an input AOE of a *canonical* form.

► **Definition 5.** A canonical AOE is an AOE which is naively expanded from an activity-on-node graph.

Every AOE G can be transformed into a canonical AOE with the same reachability relation on its tasks. First, we start by computing the reachability relation of the tasks. The transitive closure of the resulting reachability matrix gives an activity-on-node graph (which is quadratic, in the worst case, in the size of the original AOE). Then, this activity-on-node graph can be converted to a canonical AOE as described in Section 1.

► **Definition 6.** Two AOE graphs G and H are equivalent, i.e. $G \equiv H$, if G and H have the same set of tasks—i.e., there is a label-preserving bijection between the task edges of G and those of H —and, with respect to this bijection, G and H have the same set of potential critical paths.

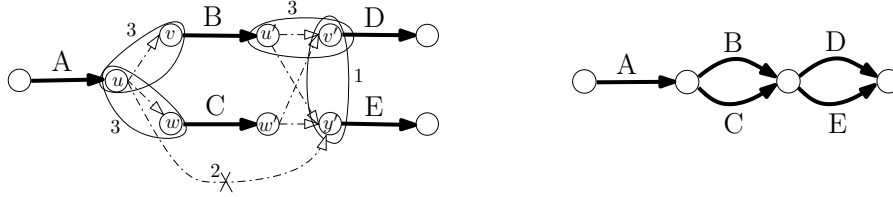
► **Definition 7.** An AOE G is optimal if G minimizes the number of vertices for its equivalence class: i.e., if for every AOE $H \equiv G$, $|V(H)| \geq |V(G)|$.

We now formally define our problem.

Problem 1. Given a canonical AOE G , find some optimal AOE H with $H \equiv G$.

3 Simplification Rules

Our algorithm takes a canonical AOE and greedily applies a set of rules until no more rules can be applied. Given an AOE $G = (V, E)$ and given two distinct vertices $u, v \in V$, the simplification rules used by our algorithm are:



■ **Figure 3** On the left, an AOE in which each of rules 1-3 can be applied, and on the right, the corresponding graph output by the algorithm.

1. if u and v have no outgoing task edges and have precisely the same outgoing neighbors, merge u and v . Symmetrically, if u and v have no incoming task edges and have precisely the same incoming neighbors, merge u and v .
2. If u has an unlabeled edge to v , and u has another path to v , remove the edge (u, v) .
3. If u has an unlabeled edge to v and the following conditions are satisfied, merge u and v :
 - rule 2 is not applicable to the edge (u, v) .
 - if u has an outgoing task, then v has no incoming edge other than (u, v) .
 - if v has an incoming task, then u has no outgoing edge other than (u, v) .
 - every incoming neighbor of v has a path to every outgoing neighbor of u .

Figure 3 depicts an AOE graph and the graph output by the algorithm after applying all possible rules. Vertices u and v can be merged by rule 3, since there is no other path from u to v to apply rule 2 (satisfying the first condition in the application of rule 3), u and v have no outgoing and no incoming task, respectively, and v has no incoming neighbor other than u . Therefore, the conditions of rule 3 are true (the second and third hold vacuously). Further, vertices u' and v' can be merged since the first three conditions for applying rule 3 are satisfied and there exists a path from w' to y' , satisfying the last condition.

It will be convenient for the proofs in Section 5 to give a name to the output of the algorithm:

► **Definition 8.** An output AOE, denoted \mathcal{A} , is any AOE obtained from a canonical AOE G by a sequence of applications of rules 1, 2, and 3, to which none of these rules can still be applied.

We will show (Theorem 19) that \mathcal{A} does not depend on the order in which the rules are applied.

4 Correctness

In this section we prove the correctness of our algorithm (its output graph is equivalent to its input graph).

We begin with preserving potential critical paths. We show that the rules never change the existence or nonexistence of a path from one task to another, and that this implies preservation of potential critical paths.

► **Lemma 9.** Given two AOE's G and H with the same set of tasks \mathcal{T} , G and H have the same reachability relation \rightsquigarrow on the tasks if and only if $G \equiv H$.

Proof. Trivially, we have $T \rightsquigarrow_G T'$ (or $T \rightsquigarrow_H T'$) if and only if T is earlier than T' in some potential critical path of G (or H). Therefore, preservation of potential critical paths is equivalent to preservation of the reachability relation. ◀

► **Lemma 10.** *The output of the algorithm is equivalent to its input.*

Proof. We show the invariant that given tasks T and T' , $T \rightsquigarrow T'$ at a given iteration of the algorithm if and only if $T \rightsquigarrow T'$ at the next iteration. From this, and from the fact that the rules never change the set of tasks, it follows that the output of the algorithm has the same reachability relation on its tasks as the input, and then the lemma follows from Lemma 9.

The invariant is true because merging a pair of vertices (rules 1 and 3) never disconnects a path, and no edge is ever removed (by rule 2) between two vertices unless another path exists between the two vertices. In particular, the end vertex of T still has a path to the start vertex of T' after the application of any of the rules.

For the other direction, removing an edge never introduces a new path. Furthermore, if vertices u and v are merged by applying rule 1, and if some vertex w has a path to some vertex z through the newly merged uv , then the condition of rule 1 ensures that w has a path, through u or v , to z before the merge. Similarly, suppose u and v are merged by applying rule 3. Then if w has a path to z through uv , then (abusing notation) either $w \rightsquigarrow u$ and $v \rightsquigarrow z$ before the merge, so $w \rightsquigarrow z$ (via the edge (u, v)), or for some incoming neighbor x of v and outgoing neighbor y of u , $w \rightsquigarrow x$ and $y \rightsquigarrow z$. In this case, by the conditions of the rule, $w \rightsquigarrow z$ before the merge. ◀

► **Lemma 11.** *Any intermediate graph that results from applying rules of the algorithm to an input canonical AOE graph, is acyclic.*

Proof. Given Definition 1 and Definition 5, the canonical AOE input G is acyclic. Now we show none of the rules can create a cycle after being applied to an intermediate acyclic graph G' . This is obvious for rule 2 as it removes edges. Suppose for a contradiction that merging vertices u and v creates a cycle. The cycle must involve the new vertex resulting from the merge. For rule 1, this implies the existence of a cycle in G' either from u or v to itself which is a contradiction. For rule 3, it implies the existence of a cycle in G' including the unlabeled edge (u, v) or a cycle including an incoming neighbor of v and an outgoing neighbor of u , which is a contradiction. ◀

► **Corollary 12.** *Any graph \mathcal{A} output by the algorithm is acyclic.*

5 Optimality

In this section we prove the optimality of our algorithm: it uses as few vertices as possible. Let \mathcal{A} be any output AOE. Let Opt be any optimal AOE such that $\mathcal{A} \equiv Opt$. Our proof relies on an injective mapping from the vertices of \mathcal{A} to the vertices of Opt . The existence of this mapping shows that \mathcal{A} has at most as many vertices as Opt , and therefore has the optimal number of vertices. Once we have identified the vertices of \mathcal{A} with the vertices of Opt in this way, we show that, for a given input, any two graphs output by the algorithm (but not necessarily Opt) must have the same unlabeled edges. Since the task edges are determined, and since the injective mapping to Opt determines the vertices, determining the unlabeled edges implies the order-independence of our algorithm's choice of simplification rules.

Before defining the mapping between \mathcal{A} and Opt , we establish some facts about the structure of \mathcal{A} .

► **Lemma 13.** *For every unlabeled edge (u, v) in any output AOE \mathcal{A} , there exist tasks T and T' such that $u = \text{End}(T)$ and $v = \text{St}(T')$.*

Proof. By Definition 8, \mathcal{A} is produced by the algorithm from some canonical AOE G . This property holds for G by Definition 5. As every rule of the algorithm either removes an unlabeled edge or merges two vertices, and never creates a new edge or vertex, the proof is complete. \blacktriangleleft

► **Corollary 14.** *Every vertex in an output AOE \mathcal{A} has an incident task edge.*

We can now define a mapping from the vertices of \mathcal{A} to those of Opt :

► **Definition 15.** *Given an output AOE \mathcal{A} with task set \mathcal{T} , and given an optimal AOE Opt with $\mathcal{A} \equiv Opt$, let $M : V(\mathcal{A}) \rightarrow V(Opt)$ be the following mapping: for every $v \in V(\mathcal{A})$:*

- *Let $M(v) = St_{Opt}(T)$, for some $T \in \mathcal{T}$ for which $v = St_{\mathcal{A}}(T)$, if such a task exists.*
- *Let $M(v) = End_{Opt}(T)$, where $v = End_{\mathcal{A}}(T)$, otherwise.*

As shown in Corollary 14, every vertex in \mathcal{A} has an incident task edge, and by Definition 6, \mathcal{A} and Opt have the same set of tasks. Therefore, this mapping is well-defined (up to its arbitrary choices of which task to use for each v). To prove that M is injective, we will use the fact that since $\mathcal{A} \equiv Opt$, \mathcal{A} and Opt have the same reachability relation (by Lemma 9 and Lemma 10).

The heart of the proof that M is injective lies in showing that if two tasks do not share a vertex in \mathcal{A} , the corresponding tasks also do not share the corresponding vertices in Opt . From this it follows that M cannot map distinct vertices in \mathcal{A} to the same vertex in Opt .

► **Lemma 16.** *Given an output AOE \mathcal{A} , and an optimal AOE $Opt \equiv \mathcal{A}$, with task set \mathcal{T} , let T and T' be two distinct tasks in \mathcal{T} . If $St_{\mathcal{A}}(T) \neq St_{\mathcal{A}}(T')$, then $St_{Opt}(T) \neq St_{Opt}(T')$. If $End_{\mathcal{A}}(T) \neq End_{\mathcal{A}}(T')$, then $End_{Opt}(T) \neq End_{Opt}(T')$.*

Proof. Suppose for a contradiction that $St_{\mathcal{A}}(T) \neq St_{\mathcal{A}}(T')$, but $St_{Opt}(T) = St_{Opt}(T')$ (the other case is symmetrical). Let $u = St_{\mathcal{A}}(T)$ and $v = St_{\mathcal{A}}(T')$. Consider the following (exhaustive) cases for u and v :

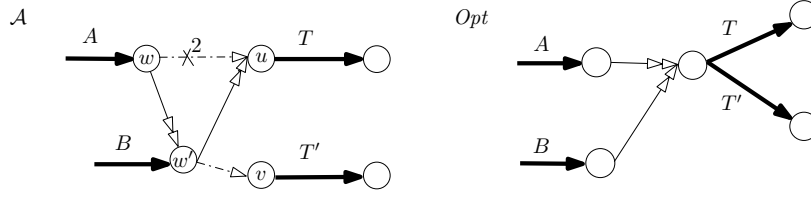
1. u and v have no incoming edges
2. u or v has an incoming unlabeled edge, but neither u nor v has an incoming task edge
3. u or v has an incoming task edge A

In case 1, applying rule 1 results in merging u and v . However, since \mathcal{A} is the output of the algorithm, no rule can be applied to \mathcal{A} . This is a contradiction.

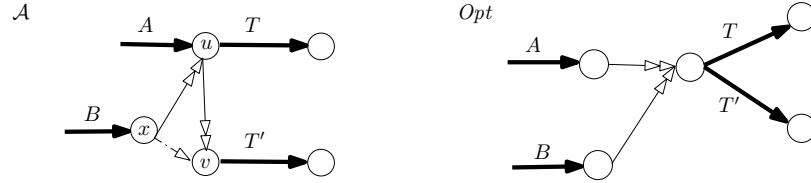
In case 2, u and v cannot have the same incoming neighbors or else rule 1 would apply. We may assume without loss of generality that there exist a vertex w and an unlabeled edge (w, u) , such that the edge (w, v) does not exist. By Lemma 13, there exists a task A where $w = End_{\mathcal{A}}(A)$. Since $A \rightsquigarrow_{\mathcal{A}} T$ and $\mathcal{A} \equiv Opt$ (by Lemma 10), then by Lemma 9, $A \rightsquigarrow_{Opt} T$, so $A \rightsquigarrow_{Opt} T'$, since $St_{Opt}(T) = St_{Opt}(T')$. Again by Lemma 9, $A \rightsquigarrow_{\mathcal{A}} T'$, so there is a path P from w to v . If $|P| = 1$, then this contradicts that (w, v) does not exist. Suppose $|P| > 1$. Then we show there exist some vertex $w' \neq w$ and an unlabeled edge (w', v) . The following cases are exhaustive:

- (a) P contains a path from u to v . As such a path to v exists and v has no incoming task edge, there exist a vertex w' and an unlabeled edge (w', v) ($w' \neq u$), not belonging to P unless rule 3 can be applied to vertex v and its incoming neighbor in path P .
- (b) P does not contain a path from u to v . As $|P| > 1$, an unlabeled edge (w', v) belonging to path P exists.

Given the existence of (w', v) , by Lemma 13, there exists a task B where $w' = End_{\mathcal{A}}(B)$. $B \rightsquigarrow_{\mathcal{A}} T'$, so by reasoning similar to the above, $B \rightsquigarrow_{\mathcal{A}} T$. Then, one can apply rule 2



■ **Figure 4** Lemma 16, case 2, subcase b (double arrows indicate paths).



■ **Figure 5** Lemma 16, case 3, subcase b.

and either remove edge (w', v) in case a or (w, u) in case b (Figure 4); this contradicts the definition of \mathcal{A} .

In case 3, we can assume without loss of generality that u has an incoming task A ; consequently, $u = \text{End}_{\mathcal{A}}(A)$. Then, by Lemma 9, we have $A \rightsquigarrow_{Opt} T'$ and thus $A \rightsquigarrow_{\mathcal{A}} T'$ via a path P . Consider the following cases for P :

- (a) P contains a task edge B
- (b) P is a sequence of unlabeled edges

In case a, by Lemma 9 $B \rightsquigarrow_{Opt} T'$, and thus $B \rightsquigarrow_{Opt} T$, and therefore $B \rightsquigarrow_{\mathcal{A}} T$. This creates a cycle between u and $\text{End}_{\mathcal{A}}(B)$, contradicting Corollary 12.

In case b, illustrated in Figure 5, since rule 3 cannot be applied (if it could, this would contradict the definition of \mathcal{A}), there exist a vertex x not on the path from u to v , and an edge (x, v) (a task edge or an unlabeled edge). Therefore, there exists a task B where either $v = \text{End}_{\mathcal{A}}(B)$ or by Lemma 13, $x = \text{End}_{\mathcal{A}}(B)$. Considering Opt and applying Lemma 9, $B \rightsquigarrow_{Opt} T$ so $B \rightsquigarrow_{\mathcal{A}} T$. This path either creates a cycle in \mathcal{A} or allows for removing edge (x, v) by rule 2, which is a contradiction.

Thus if $\text{St}_{\mathcal{A}}(T) \neq \text{St}_{\mathcal{A}}(T')$, then $\text{St}_{Opt}(T) \neq \text{St}_{Opt}(T')$. ◀

► **Lemma 17.** *Given an output AOE \mathcal{A} , and an optimal AOE $Opt \equiv \mathcal{A}$, with task set \mathcal{T} , let T and T' be two distinct tasks in \mathcal{T} . If $\text{End}_{\mathcal{A}}(T) \neq \text{St}_{\mathcal{A}}(T')$, then $\text{End}_{Opt}(T) \neq \text{St}_{Opt}(T')$.*

Proof. The proof, which is in Appendix A.1, uses essentially the same approach as the proof of Lemma 16: supposing that the two vertices are the same, then using the fact that \mathcal{A} and Opt have the same reachability relation on their tasks, and the definition of \mathcal{A} as having no more rules to apply, to derive a contradiction. ◀

There is one remaining technicality: we have defined an optimal AOE as being acyclic; the question arises whether one could reduce the number of vertices by allowing (unlabeled) cycles. However, this is not the case; it is easy to see that any unlabeled cycle can be merged into one vertex, reducing the number of vertices, without changing the reachability relation on the tasks.

We are ready to prove our main results.

► **Theorem 18.** *Given a canonical AOE G , the algorithm produces an optimal AOE $Opt \equiv G$.*

Proof. Let \mathcal{A} be the output AOE produced by the algorithm on G . Given any optimal AOE Opt and \mathcal{A} , the mapping M in Definition 15 is injective: suppose for a contradiction that u and v are distinct vertices in \mathcal{A} , and $w = M(u) = M(v)$. Then by the definition of M , either u , v , and w have the same incoming task, or u , v , and w have the same outgoing task, or there exist tasks T and T' such that (without loss of generality) $u = \text{End}_{\mathcal{A}}(T)$, $v = \text{St}_{\mathcal{A}}(T')$, and $\text{End}_{Opt}(T) = w = \text{St}_{Opt}(T')$. By Lemmas 16 and 17, all three of these cases imply that $u = v$.

Therefore, $|V(Opt)| = |V(\mathcal{A})|$. Furthermore, $\mathcal{A} \equiv G$, by Lemma 10. The theorem follows. \blacktriangleleft

► **Theorem 19.** *Given an input, the algorithm produces the same output regardless of the order in which the rules are applied.*

Proof. As stated earlier, all task edges of an input canonical AOE G are present in any output of the algorithm and the mapping determines the vertices. Therefore, it suffices to show that any two graphs output by the algorithm have the same set of unlabeled edges. Suppose for a contradiction that \mathcal{A}_1 and \mathcal{A}_2 are two distinct outputs of the algorithm, resulting from applying different sequences of rules. By Theorem 18, the algorithm always produces an optimal AOE. Therefore, $|V_{\mathcal{A}_1}| = |V_{\mathcal{A}_2}| = |V_{Opt}|$. Since $\mathcal{A}_1 \neq \mathcal{A}_2$, there is an unlabeled edge (u, v) in \mathcal{A}_1 (without loss of generality) that is not in \mathcal{A}_2 . By Lemma 13, there exist task edges T and T' such that $u = \text{End}_{\mathcal{A}_1}(T)$ and $v = \text{St}_{\mathcal{A}_1}(T')$. We have $T \rightsquigarrow_{\mathcal{A}_1} T'$. Since by Lemma 9 and Lemma 10, \mathcal{A}_1 and \mathcal{A}_2 both preserve the reachability relation of the tasks of G , we have $T \rightsquigarrow_{\mathcal{A}_2} T'$. Consider the cases for path P from T to T' in \mathcal{A}_2 :

1. There exists a task A in P other than T and T' .
2. Path P is a sequence of unlabeled edges.

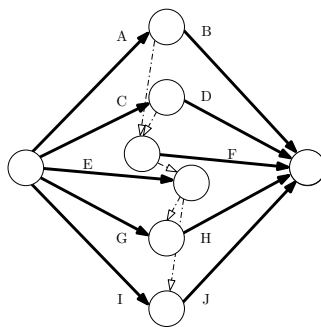
In case 1, we have $T \rightsquigarrow_{\mathcal{A}_2} A \rightsquigarrow_{\mathcal{A}_2} T'$ and therefore, $T \rightsquigarrow_{\mathcal{A}_1} A \rightsquigarrow_{\mathcal{A}_1} T'$. Then by rule 2, one can remove the edge (u, v) , which contradicts the definition of \mathcal{A}_1 .

In case 2, the length of P is at least two, and P contains a vertex w . By Lemma 13, there exist tasks A and B where $w = \text{End}_{\mathcal{A}_2}(A) = \text{St}_{\mathcal{A}_2}(B)$. Now, since $\mathcal{A}_1 \equiv \mathcal{A}_2$, both graphs are optimal, and both graphs are outputs of the algorithm, Lemma 17 implies that $\text{End}_{\mathcal{A}_1}(A) = \text{St}_{\mathcal{A}_1}(B)$. Call this vertex x . Then there exists a path from u to v , through x , by Lemma 9, and one can remove the edge (u, v) by rule 2. This contradicts the definition of \mathcal{A}_1 . \blacktriangleleft

It is tempting to imagine that Theorem 19 implies uniqueness of the optimal AOE. However, this is not the case: the unlabeled edges of an optimal AOE are not determined by our bijection. Figure 6 shows an optimal AOE graph that our algorithm cannot produce (because it violates Lemma 13). One way to see the optimality is to expand the graph naively into a canonical AOE, apply the algorithm, and verify that the resulting number of vertices is the same.

6 Analysis

Let n be the number of vertices in a canonical AOE (which is linear in the number of tasks), and m the number of unlabeled edges. (The number of task edges is $O(m)$.) There are at most $O(n + m)$ iterations in the algorithm, because each iteration either merges two vertices or removes an edge, by applying one of the three rules. This requires finding an edge to remove ($O(m)$ potential edges) or two vertices to merge ($O(n^2)$ potential pairs), then performing the merge or the removal. Intuitively, our algorithm runs in polynomial time as it takes polynomial time to find and apply a rule.



■ **Figure 6** An optimal AOE graph that the algorithm cannot produce.

■ **Algorithm 1** the proposed transformation algorithm.

Data: Canonical AOE G
Result: Optimal AOE Opt

```

1 while true do
2   Initialize and compute the reachability matrix  $M$ ;
3   Remove, by rule 2, all unlabeled edges  $(u, v)$  where  $M[u][v] = 2$ ;
4   if rule 1 applies then
5     | apply rule 1;
6   else if rule 3 applies then
7     | apply rule 3;
8   else
9     | return the graph;

```

We provide a faster implementation of our algorithm than the naive approach. The algorithm transforms a canonical AOE graph G into an optimal AOE graph by applying rules 1, 2 or 3. For simplicity, we label the vertices $1, \dots, n$. At each iteration, compute a reachability matrix M for the current graph. $M[u][v]$ indicates whether there exist zero, one, or more than one paths from u to v . In order to compute M , for all u and v initialize $M[u][v] = 1$ if the edge (u, v) exists. Then sort the vertices in topological order (such an ordering exists according to Lemma 11). For each vertex v in this order, and for each vertex u , set $M[u][v]$ to $\min(2, \sum_{w \in W} (M[u][w]))$, where W is the set of all vertices w such that either $w = v$ or there exists an edge (w, v) . This procedure takes $O(nm)$ time. Algorithm 1 provides a summary.

Given the reachability matrix, an unlabeled edge (u, v) is removed by rule 2 in $O(1)$ time, if $M[u][v] \geq 2$. Therefore, checking rule 2 for all edges takes $O(m)$ time.

Without loss of generality, for rule 1, we only consider merges of pairs of vertices with the same outgoing neighbors. This requires, for each vertex u with no outgoing task edge, a sorted list of outgoing neighbors ($S[u]$). To obtain such lists for all vertices, list unlabeled edges as pairs of vertices and sort all the pairs with two bucket sorts: first over the first elements of the pairs, then over the second elements. Breaking the sorted list into chunks of pairs with the same first element (say u), gives the outgoing neighbors of u , in the second elements of the pairs, in a numerically sorted order. This takes $O(m)$ time. Then find pairs of vertices to merge, if any exist: first, bucket sort vertices based on their out-degree. Vertices in different buckets cannot be merged by rule 1. For each bucket b containing vertices with degree d ($0 \leq d < n$), call MergeDetection(b, d):

```

10 Function MergeDetection(bucket a, i):
11   if  $i = 0$  then
12     |   return bucket  $a$ 
13   else
14     |   bucket sort vertices  $v$  of  $a$  based on  $S[v][i]$ 
15     |   foreach newly created bucket  $a'$  do
16     |     |   MergeDetection( $a', i - 1$ )

```

The vertices in each resulting bucket have the same outgoing neighbors and can be merged by rule 1. As each vertex with degree d appears in one bucket in each of $d + 1$ iterations, this sort takes $O(\sum_v(deg(v))) = O(m)$ time. Upon merging vertices u and v , name the new vertex $\min(u, v)$.

To check rule 3, for each vertex v , compute $I(v)$: the intersection of the reachable sets of the incoming neighbors of v . This takes $O(mn)$ time.

Consider only those unlabeled edges (u, v) that meet the preconditions of rule 3 concerning the existence of outgoing and incoming tasks of u and v respectively. Test whether the last point in rule 3 applies to edge (u, v) by testing in $O(n)$ time whether all outgoing neighbors of u are in $I(v)$.

Computing the reachability matrix takes $O(mn)$ time, and using this matrix to check for rule 2 takes $O(m)$ time per iteration. Checking for rule 1 or 3 takes $O(mn)$ time per iteration. Further, the outer loop in Algorithm 1 runs at most n times as it either merges two vertices or returns the output. This gives a total complexity of $O(mn^2)$ for our algorithm.

7 Conclusion

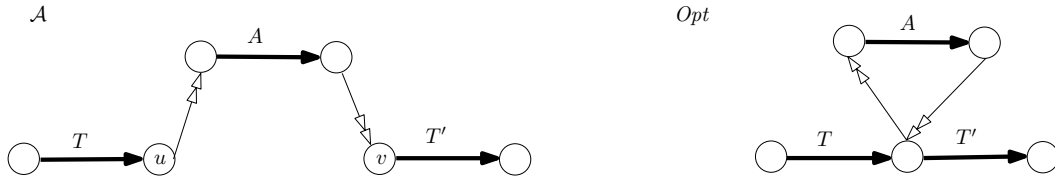
Our algorithm reduces the visual complexity of an activity-on-edge graph, making it easier to understand bottlenecks in a project. The algorithm repeatedly applies simple rules and therefore can be implemented easily. We have shown that the algorithm runs in $O(mn^2)$ time. One question for future work is whether this analysis is tight. Another question is whether some other algorithm could achieve an optimal graph more efficiently.

Furthermore, one can measure the complexity of a graph in other ways. One question for future work is whether one can minimize the number of edges in an AOE graph in polynomial time. Another question is whether one can, in polynomial time, convert an AOE graph G into a graph that (i) has the same potential critical paths as G , and (ii) has a plane drawing with fewer edge crossings than all other graphs satisfying (i). It would also be interesting to implement this algorithm and run it on realistic graphs arising in project planning, and to evaluate the visual complexity of the resulting graphs in terms of the measures described above.

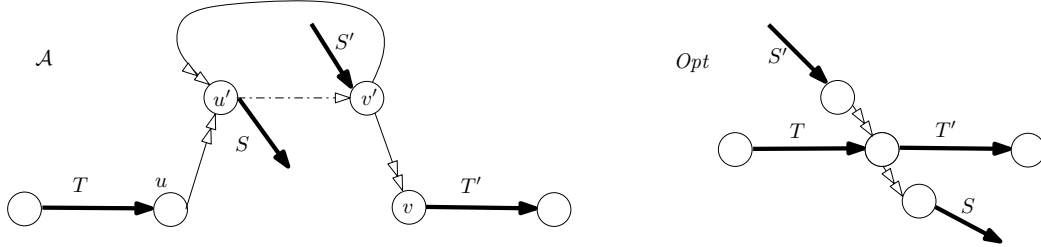
References

- 1 Sarmad Abbasi, Patrick Healy, and Aimal Rextin. Improving the running time of embedded upward planarity testing. *Information Processing Letters*, 110(7):274–278, 2010. doi:10.1016/j.ip1.2010.02.004.
- 2 A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. doi:10.1137/0201008.

- 3 C. Alexander, D. Reese, and J. Harden. Near-critical path analysis of program activity graphs. In *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 308–317. IEEE, 1994. doi:10.1109/mascot.1994.284406.
- 4 David Arditi and Thanat Pattanakitchamroon. Selecting a delay analysis method in resolving construction claims. *International J. Project Management*, 24(2):145–155, 2006. doi:10.1016/j.ijproman.2005.08.005.
- 5 Oliver Bastert and Christian Matuszewski. Layered drawings of digraphs. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*, pages 87–120. Springer-Verlag, 2001. doi:10.1007/3-540-44969-8_5.
- 6 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.
- 7 Pranay Chaudhuri and Ratan K. Ghosh. Parallel algorithms for analyzing activity networks. *BIT*, 26(4):418–429, 1986. doi:10.1007/bf01935049.
- 8 Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete and Computational Geometry*, 7(4):381–401, 1992. doi:10.1007/BF02187850.
- 9 David Eppstein and Joseph A. Simons. Confluent Hasse diagrams. *J. Graph Algorithms Appl.*, 17(7):689–710, 2013. doi:10.7155/jgaa.00312.
- 10 Ashim Garg and Roberto Tamassia. Upward planarity testing. *Order*, 12(2):109–133, 1995. doi:10.1007/BF01108622.
- 11 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.
- 12 Patrick Healy, Ago Kuusik, and Sebastian Leipert. A characterization of level planar graphs. *Discrete Math.*, 280(1-3):51–63, 2004. doi:10.1016/j.disc.2003.02.001.
- 13 Patrick Healy and Nikola S. Nikolov. Hierarchical Graph Drawing. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 409–453. CRC Press, 2014.
- 14 Jerry L. Householder and Hulan E. Rutland. Who owns float? *J. Construction Engineering and Management*, 116(1):130–133, 1990. doi:10.1061/(ASCE)0733-9364(1990)116:1(130).
- 15 Wei Huang and Lixin Ding. Project-scheduling problem with random time-dependent activity duration times. *IEEE Transactions on Engineering Management*, 58(2):377–387, 2011. doi:10.1109/tem.2010.2063707.
- 16 Michael Jünger and Sebastian Leipert. Level Planar Embedding in Linear Time. In Jan Kratochvíl, editor, *Graph Drawing: 7th International Symposium*, volume 1731 of *Lecture Notes in Computer Science*, pages 72–81. Springer-Verlag, 1999. doi:10.1007/3-540-46648-7_7.
- 17 Vidyadhar G. Kulkarni. A compact hash function for paths in PERT networks. *Operations Research Letters*, 3(3):137–140, 1984. doi:10.1016/0167-6377(84)90005-1.
- 18 Donald G. Malcolm, John H. Roseboom, Charles E. Clark, and Willard Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959. doi:10.1287/opre.7.5.646.
- 19 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.
- 20 Zhenming Xu. Automatic layout of information in the AOE network. In *2010 International Conference on Mechanic Automation and Control Engineering*. IEEE, June 2010. doi:10.1109/mace.2010.5536213.



■ **Figure 7** Lemma 17, case 1.



■ **Figure 8** Lemma 17, case 2a.

A Appendix

A.1 Proof of Lemma 17

► **Lemma 17.** *Given an output AOE \mathcal{A} , and an optimal AOE $Opt \equiv \mathcal{A}$, with task set \mathcal{T} , let T and T' be two distinct tasks in \mathcal{T} . If $\text{End}_{\mathcal{A}}(T) \neq \text{St}_{\mathcal{A}}(T')$, then $\text{End}_{Opt}(T) \neq \text{St}_{Opt}(T')$.*

Proof. Suppose for a contradiction that $u = \text{End}_{\mathcal{A}}(T) \neq v = \text{St}_{\mathcal{A}}(T')$, but $\text{End}_{Opt}(T) = \text{St}_{Opt}(T')$. Since $T \rightsquigarrow_{Opt} T'$, then by Lemma 9, $T \rightsquigarrow_{\mathcal{A}} T'$, so u has a path P to v . Consider the following possible cases for path P :

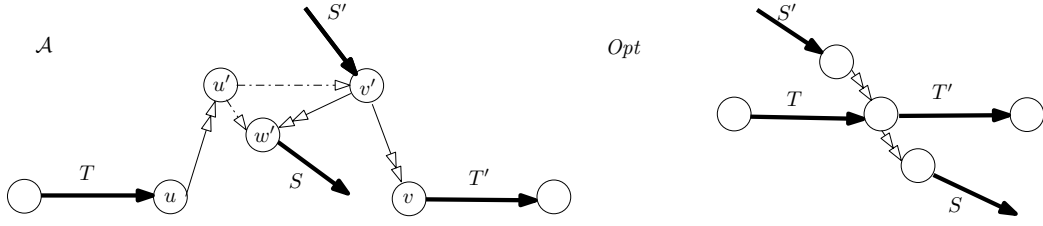
1. There exists a task A in P .
2. P only consists of unlabeled edges. Consider the cases for any unlabeled edge (u', v') in P :
 - a. There exist incident tasks S and S' , pointing away from and toward u' and v' , respectively.
 - b. There exists an incident unlabeled edge (u', w') pointing away from u' and an incident task S' pointing toward v' .
 - c. There exists an incident task S pointing away from u' and an incident unlabeled edge (w', v') pointing toward v' .
 - d. There exists an incident unlabeled edge (u', w') pointing away from u' and an incident unlabeled edge (x', v') pointing toward v' . Vertices u' and v' have no outgoing or incoming task edges, respectively.

These cases are exhaustive as path P either has a task or it is a sequence of unlabeled edges. Further, for case 2, suppose for an unlabeled edge (u', v') , none of the subcases of 2a, 2b, 2c and 2d holds. Then, by rule 3, one can merge vertices u' and v' ; this contradicts the definition of \mathcal{A} .

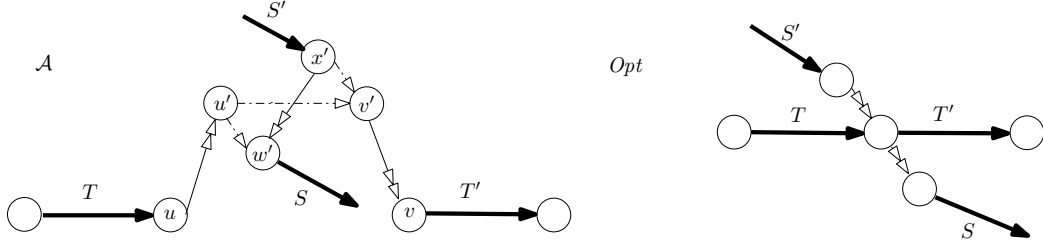
In case 1, shown in Figure 7, we have $T \rightsquigarrow_{\mathcal{A}} A \rightsquigarrow_{\mathcal{A}} T'$ so by Lemma 9, $T \rightsquigarrow_{Opt} A \rightsquigarrow_{Opt} T'$. Therefore, there is a path in Opt (through A) from $\text{End}_{Opt}(T)$ to $\text{St}_{Opt}(T')$. Since $\text{End}_{Opt}(T) = \text{St}_{Opt}(T')$, this path creates a cycle in Opt . However, Opt is an AOE graph and is therefore acyclic by Definition 1.

In case 2a, shown in Figure 8, $S' \rightsquigarrow_{\mathcal{A}} T'$, so by Lemma 9, $S' \rightsquigarrow_{Opt} T'$, i.e. there is a path from S' to $\text{St}_{Opt}(T')$. Similarly, $T \rightsquigarrow_{\mathcal{A}} T'$, so there is a path from $\text{End}_{Opt}(T)$ to S . Since

24:14 Simplifying Activity-On-Edge Graphs



■ **Figure 9** Lemma 17, case 2b.



■ **Figure 10** Lemma 17, case 2d.

$\text{St}_{\text{Opt}}(T') = \text{End}_{\text{Opt}}(T)$, this implies $S' \rightsquigarrow_{\text{Opt}} S$, so by Lemma 9, $S' \rightsquigarrow_{\mathcal{A}} S$. Therefore, there is a path in \mathcal{A} from $\text{End}_{\mathcal{A}}(S')$ to $\text{St}_{\mathcal{A}}(S)$. This path creates a cycle in \mathcal{A} and contradicts Corollary 12.

In case 2b, shown in Figure 9, by Lemma 13, there exists a task S where $w' = \text{St}_{\mathcal{A}}(S)$. Since we have $T \rightsquigarrow_{\mathcal{A}} S$ and $S' \rightsquigarrow_{\mathcal{A}} T'$ and $\text{End}_{\text{Opt}}(T) = \text{St}_{\text{Opt}}(T')$, by Lemma 9, we have $S' \rightsquigarrow_{\text{Opt}} S$. Therefore, there is a path in \mathcal{A} from $v' = \text{End}_{\mathcal{A}}(S')$ to $w' = \text{St}_{\mathcal{A}}(S)$. This path either creates a cycle between u' and v' , contradicting Corollary 12 or by rule 2, one can remove edge (u', w') , which is a contradiction by the definition of \mathcal{A} .

Case 2c is almost identical to case 2b, and again leads to the existence of a path from S' to S (similarly defined), resulting in either a cycle or an application of rule 2.

In case 2d, shown in Figure 10, by Lemma 13, there exist task edges S and S' where $w' = \text{St}_{\mathcal{A}}(S)$ and $x' = \text{End}_{\mathcal{A}}(S')$, and unlabeled edges (x', v') and (u', w') . We have $S' \rightsquigarrow_{\text{Opt}} S$, then by Lemma 9, $S' \rightsquigarrow_{\mathcal{A}} S$. Therefore, there is a path in \mathcal{A} from $\text{End}_{\mathcal{A}}(S')$ to $\text{St}_{\mathcal{A}}(S)$. This path either creates a cycle between u' and v' , contradicting Corollary 12 or by rule 3, one can merge u' and v' in \mathcal{A} , which is a contradiction by the definition of \mathcal{A} .

Thus if $\text{End}_{\mathcal{A}}(T) \neq \text{St}_{\mathcal{A}}(T')$, then $\text{End}_{\text{Opt}}(T) \neq \text{St}_{\text{Opt}}(T')$. ◀

Generalized Metric Repair on Graphs

Chenglin Fan

Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA
cxf160130@utdallas.edu

Anna C. Gilbert

Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA
annacg@umich.edu

Benjamin Raichel

Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA
benjamin.raichel@utdallas.edu

Rishi Sonthalia

Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA
rsonthal@umich.edu

Gregory Van Buskirk

Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA
greg.vanbuskirk@utdallas.edu

Abstract

Many modern data analysis algorithms either assume or are considerably more efficient if the distances between the data points satisfy a metric. However, as real data sets are noisy, they often do not possess this fundamental property. For this reason, Gilbert and Jain [10] and Fan et al. [9] introduced the closely related *sparse metric repair* and *metric violation distance* problems. Given a matrix, representing *all* distances, the goal is to repair as few entries as possible to ensure they satisfy a metric. This problem was shown to be APX-hard, and an $O(OPT^{1/3})$ -approximation was given, where OPT is the optimal solution size.

In this paper, we generalize the problem, by describing distances by a possibly *incomplete* positively weighted graph, where again our goal is to find the smallest number of weight modifications so that they satisfy a metric. This natural generalization is more flexible as it takes into account different relationships among the data points. We demonstrate the inherent combinatorial structure of the problem, and give an approximation-preserving reduction from MULTICUT, which is hard to approximate within any constant factor assuming UGC. Conversely, we show that for any fixed constant ζ , for the large class of ζ -chordal graphs, the problem is fixed parameter tractable, answering an open question from previous work. Call a cycle *broken* if it contains an edge whose weight is larger than the sum of all its other edges, and call the amount of this difference its *deficit*. We present approximation algorithms, one depending on the maximum number of edges in a broken cycle, and one depending on the number of distinct deficit values, both quantities which may naturally be small. Finally, we give improved analysis of previous algorithms for complete graphs.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Approximation, FPT, Hardness, Metric Spaces

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.25

Funding C. Fan, B. Raichel, and G. Van Buskirk supported by NSF awards 1566137 and 1750780.

1 Introduction

Given a set of distances determined by a collection of data points, one of the most basic questions we can ask is whether the distances satisfy a metric. This basic property is fundamental to a large number of computational geometry and machine learning tasks such as metric learning, dimensionality reduction, and clustering (see for example [17, 2]). It



© Chenglin Fan, Anna C. Gilbert, Benjamin Raichel, Rishi Sonthalia, and Gregory Van Buskirk; licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 25; pp. 25:1–25:22

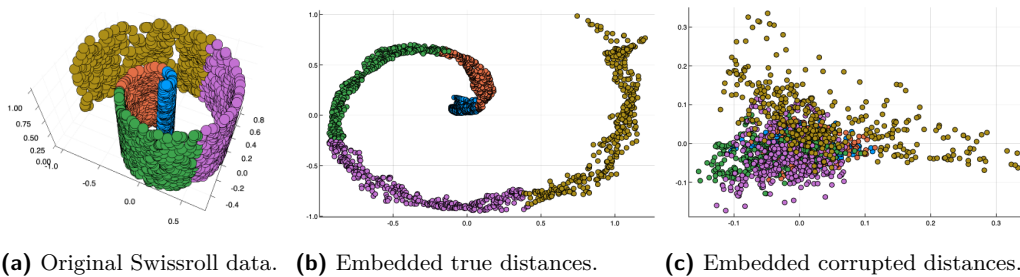
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is fortuitous when the underlying distances arise from a metric space or are at least well modeled by one, as certain tasks become provably easier over metric data (e.g., approximating the optimal TSP tour), and moreover it allows us to use a number of computational tools such as metric embeddings. However, due to noise, missing data, and other corruptions, in practice these distances often do not adhere to a metric.

As a motivating example, consider the following standard manifold learning task ([3, 13, 15]). Given a high dimensional data set, we wish to uncover its intrinsic lower dimensional structure, allowing us to visualize and to understand the geometry of the data. Isomap [15] is one of the standard embedding tools used to find this lower dimensional structure, and Figure 1 shows how Isomap nicely recovers the 2d spiral when embedding a 3d Swiss roll data set. However, as shown on the right in Figure 1, if we perturb even a small fraction of the distances this structure is lost in the embedding produced by Isomap.



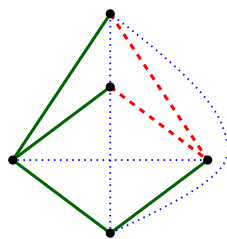
■ **Figure 1** (a) 2000 data points in the Swissroll. For (b) and (c) we took the pairwise distance matrix and added $2\mathcal{N}(0, 1)$ noise to 5% of the distances. We then constructed the 30-nearest-neighbor graph G from these distances, where roughly 8.5% of the edge weights of G were perturbed. For (b) we used the true distances on G as the input to ISOMAP. For (c) we used the perturbed distances.

Motivated by the above applications, the problem of minimally fixing the distances to uncover the data metric was previously considered. Specifically, Fan et al. [9] and Gilbert and Jain [10] respectively formulated the *Metric Violation Distance (MVD)* and the *Sparse Metric Repair (SMR)* problems, where in both cases one is given a *full* distance matrix, and the goal is to modify as few entries as possible so that the repaired distances satisfy a metric.

More generally, however, the underlying distance graph will be incomplete as data may be missing or the constructed distance graph is inherently sparse, as the above manifold example demonstrates. Working directly with this incomplete graph is not only computationally more desirable when the graph is sparse, but also may be necessary to uncover the ground truth. For example, observe that for any graph we can attempt to fill in its missing edges by assigning them weights according to their shortest path distance. Thus naively one could attempt to fix the input graph, by solving MVD/SMR on this complete graph, and afterwards dropping any selected edges that were not in the original graph. Figure 2 shows that doing so, however, can produce radically different and sub-optimal solutions.

Thus to appropriately capture this more general problem, we define the *Graph Metric Repair* problem as the natural graph theoretic generalization of the MVD and SMR problems:

Given a positively weighted undirected graph $G = (V, E, w)$ and a set $\Omega \subseteq \mathbb{R}$, find the smallest set of edges $S \subseteq E$ such that by modifying the weight of each edge in S , by adding a value from Ω , the new distances satisfy a metric.



■ **Figure 2** Original graph is the four solid green edges of weight 1, and two dashed red edges of weight 4. Added blue dotted edges all have weight 2. The original graph is repaired by increasing the lower left green edge, but the optimal solution in the complete graph decreases the two red edges.

The additional graph structure introduced in the generalized problem lets us incorporate different types of relationships amongst data points and gives us more flexibility in its structure, and hence avails itself to be applicable to a richer class of problems. This general graph structure also elucidates the deep connections to cutting problems, which underlie several results in this paper, and which were not previously observed in [9, 10]. In particular, as discussed below, our problem is closely related to **MULTICUT** (Problem 4.1), a generalization of the standard s - t cut problem to multiple s - t pairs, as well as **LB-CUT** (Problem 4.2), where only s - t paths with lengths up to a given threshold L must be cut.

It should also be noted that Graph Metric Repair, as well as MVD and SMR, are related to a large number of other previously studied problems. A short list includes: metric nearness, seeking the metric minimizing the sum of distance value changes [5]; metric embedding with outliers, seeking the fewest points whose removal creates a metric [14]; matrix completion, seeking to fill missing matrix entries to produce a low rank [6]; and many more. See [9] for a more detailed discussion of these and other problems.

Contributions and Results. The main contributions of this paper are as follows:

- We transition all previously known structural results about SMR and MVD to the new graph theoretic version. In particular, we provide a *characterization* for the support of solutions to the increase ($\Omega = \mathbb{R}_{\geq 0}$) and general ($\Omega = \mathbb{R}$) versions of the problem. Furthermore, we provide a new structural result showing that the increase only problem reduces to the general one, where it is unknown if such a result holds for SMR and MVD.
- For any fixed constant ζ , by parameterizing on the size of the optimal solution, we present a *fixed parameter tractable* algorithm for the case when G is ζ -chordal. This not only answers an open question posed by [9] for complete graphs, but significantly extends it to the larger ζ -chordal case (see [7] for characterizations of such graphs, many of which are the complements of a variety of families of graphs). Moreover, we get an upper bound on the number of optimal supports, as each one is seen by some branch of the algorithm.
- We give polynomial-time approx-preserving reductions from **MULTICUT** and **LB-CUT** to graph metric repair. This connection to the well studied **MULTICUT** problem is interesting in its own right, but by [8] it also implies graph metric repair is NP-hard, and cannot be approximated within any constant factor assuming the Unique Games Conjecture (UGC).
- We give approximation algorithms, parameterized by different measures of how far the input is from a metric. Significantly, our approximations mirror our hardness results. Call a cycle *broken* if it contains an edge whose weight is larger than the sum of all its other edges, and call the amount of this difference its *deficit*. We give an L -approximation, where L is the maximum number of edges in a broken cycle, while **LB-CUT** gives $\Omega(\sqrt{L})$ -hardness. We give an $O(\kappa \log n)$ -approximation, where κ is the number of distinct cycle deficit values, while in general the best known approximation for **MULTICUT** is $O(\log n)$.

- Finally, we give improved analysis of previous algorithms for the complete graph case. To keep the focus on our main results, this entire section has been moved to Appendix C.

2 Preliminaries

2.1 Notation and problem definition

Let us start by defining some terminology. Throughout the paper, the input is an undirected and weighted graph $G = (V, E, w)$. A subgraph $C = (V', E')$ is called a k -cycle if $|V'| = |E'| = k$, and the subgraph is connected with every vertex having degree exactly 2. We often overload this notation and use C to denote either the cyclically ordered list of vertices or edges from this subgraph. Let $C \setminus e$ denote the set of edges of C after removing the edge e , and $\pi(C \setminus e)$ denote the corresponding induced path between the endpoints of e .

A cycle C is **broken** if there exists an edge $h \in C$ such that $w(h) > \sum_{e \in C \setminus h} w(e)$. In this case, we call the edge h the **heavy** edge of C , and all other edges of C are called **light** edges. We call a set of edges a **light cover** if it contains at least one light edge from each broken cycle. Similarly, we call it a **regular cover** if it contains at least one edge from each broken cycle. We say that a weighted graph $G = (V, E, w)$ **satisfies a metric** if there are no broken cycles. Finally, let $\text{Sym}_n(\Omega)$ be the set of $n \times n$ symmetric matrices with entries drawn from $\Omega \subseteq \mathbb{R}$. Note that the weight function w can be viewed as an $n \times n$ symmetric matrix (missing edges get weight ∞), and thus for any $W \in \text{Sym}_n(\Omega)$, the matrix sum $w + W$ defines a new weight function. Now we can define the generalized graph metric repair problem as follows. In the following, $\|W\|_0$ is the number of non-zero entries in the matrix W , i.e., the ℓ_0 pseudonorm when viewing the matrix W as a vector.

► **Problem 2.1.** *Given $\Omega \subseteq \mathbb{R}$ and a positively weighted graph $G = (V, E, w)$ we want to find*

$$\arg \min_{W \in \text{Sym}(\Omega)} \|W\|_0 \text{ such that } G = (V, E, w + W) \text{ satisfies a metric, or return NONE,}$$

if no such W exists. Denote this problem as graph metric repair or $MR(G, \Omega)$.

A matrix W is an *optimal solution* if it realizes the $\arg \min$ in the above, and is a *solution* (without the *optimal* prefix) if $G = (V, E, w + W)$ satisfies a metric, but $\|W\|_0$ is not required to be minimum. The **support** of a matrix $W \in \text{Sym}(\Omega)$, denoted S_W , is the set of edges corresponding to non-zero entries in W . As we will see in Proposition 7, given a support for a solution W , we can easily find satisfying entries. Thus, the main difficulty lies in finding the support. Throughout we use OPT to denote the size of the support of an optimal solution.

We also need the following basic graph theory definitions: K_n is the complete graph on n vertices. C_n is the cycle n vertices. A **chord** of a cycle is an edge connecting two non-adjacent vertices. For a given value ζ , a graph G is called a ζ -**chordal** if the size of the largest chordless cycle in G is $\leq \zeta$.

Let the **deficit** of a broken cycle C , denoted $\delta(C)$, be the weight of its heavy edge minus the sum of the weights of all other edges in C . Similarly, $\delta(G)$ denotes the maximum of $\delta(C)$ over all broken cycles. Finally, let $L + 1$ be the maximum number of edges in a broken cycle (i.e., L counts the light edges). Note δ and L are both parameters measuring the extent to which cycles are broken, δ with respect to weights and L with respect to the number of edges.

In several places we compute all pairs shortest paths (APSP). Let T_{APSP} denote the time to do so, where $T_{\text{APSP}} = O(mn + n^2 \log n)$ using Dijkstra's algorithm and Fibonacci heaps.

2.2 Previous results

Fan et al. [9] and Gilbert and Jain [10] studied the special case of $\text{MR}(G, \Omega)$ where $G = K_n$. Three sub-cases based on Ω were considered, namely $\Omega = \mathbb{R}_{\leq 0}$ (decrease only), $\mathbb{R}_{\geq 0}$ (increase only), and \mathbb{R} (general). Various structural, hardness, and algorithmic results were presented for these cases. In particular, the major results from these previous works are as follows. (Note the notation and terminology here differs slightly from [9, 10].)

► **Theorem 1.** [9, 10] *The problem $\text{MR}(K_n, \mathbb{R}_{\leq 0})$ can be solved in $O(T_{\text{APSP}})$ time.*

► **Theorem 2.** [9] *For a complete positively weighted graph $K_n = (V, E, w)$ and $S \subseteq E$:
nosep S is a regular cover if and only if S is the support to a solution to $\text{MR}(K_n, \mathbb{R})$.
nosep S is a light cover if and only if S is the support to a solution to $\text{MR}(K_n, \mathbb{R}_{\geq 0})$.*

► **Theorem 3.** [9, 10] *Given the support S of a solution to $\text{MR}(K_n, \mathbb{R}_{\geq 0})$ or $\text{MR}(K_n, \mathbb{R})$, in polynomial time one can find a weight assignment to the edges in S which is a solution.*

[10] *Moreover, for $\text{MR}(K_n, \mathbb{R}_{\geq 0})$, if $K_n - S$ is connected, then for any edge $uv \in S$, setting the weight of uv to be the shortest distance between u and v in $K_n - S$ is a solution.*

► **Theorem 4.** [9] *The problems $\text{MR}(K_n, \mathbb{R}_{\geq 0})$ and $\text{MR}(K_n, \mathbb{R})$ are APX-Complete, and moreover permit $O(\text{OPT}^{1/3})$ approximation algorithms.*

3 Transitioning to Graph Metric Repair

In this section we generalize Theorems 1, 2, and 3 to the case when G is any graph, and additionally show that for general graphs $\text{MR}(G, \mathbb{R}_{\geq 0})$ reduces to $\text{MR}(G, \mathbb{R})$. Subsequently, in the later sections of paper, we provide a number of new stronger hardness and approximation results for $\text{MR}(G, \mathbb{R}_{\geq 0})$ and $\text{MR}(G, \mathbb{R})$ for general graphs, as well as an FPT algorithm for ζ -chordal graphs, in effect generalizing and strengthening Theorem 4, and answering previously unresolved questions.

For $\text{MR}(G, \mathbb{R}_{\leq 0})$ we have the following generalization of Theorem 1. Moreover, we observe the hardness proof of [9] implies if weights are allowed to increase even by a single value, the problem is APX-Complete. The proof of the theorem below follows fairly directly from previous work, and so has been moved to Appendix A.1, which contains additional corollaries.

► **Theorem 5.** *The problem $\text{MR}(G, \mathbb{R}_{\leq 0})$ can be solved in $O(T_{\text{APSP}})$ time.*

Moreover, the problem becomes hard if even a single positive value is allowed. That is, if $0 \in \Omega$ and $\Omega \cap \mathbb{R}_{> 0} \neq \emptyset$ then $\text{MR}(G, \Omega)$ is APX-Complete.

3.1 Structural results

Theorem 2 suggests that the problem is mostly combinatorial in nature. We shall see that, in general, the difficult part of the problem is finding the support of an optimal solution. Next, we present a characterization of the support of all solutions to the graph metric repair problem, generalizing Theorems 2, 3. It should be noted that the proof of the following is significantly simpler than the proof of Theorem 2 in [9]. The key insight is:

- (i) If the shortest path between two adjacent vertices is not the edge connecting them, then this edge is the heavy edge of a broken cycle.

► **Theorem 6.** *For any positively weighted graph $G = (V, E, w)$ and $S \subseteq E$:*

1. S is a regular cover if and only if S is the support to a solution to $\text{MR}(G, \mathbb{R})$.
2. S is a light cover if and only if S is the support to a solution to $\text{MR}(G, \mathbb{R}_{\geq 0})$.

Proof. First, assume that S is the support of a solution to $\text{MR}(G, \mathbb{R})$ ($\text{MR}(G, \mathbb{R}_{\geq 0})$). Suppose C is a broken cycle in G . If S does not contain any (light) edges from C , then changing (increasing) the weights on S could not have fixed C . Hence, S must be a regular (light) cover thus proving the “if” direction of both parts of the theorem.

For the “only if” direction, we are given a regular (light) cover $S \subseteq E$ which we use to define a graph $\hat{G} = (V, E \setminus S, w)$. Note that since S is either a regular or light cover, S contains at least one edge from all broken cycles of G . Thus, since \hat{G} is G with the edges of S removed, \hat{G} has no broken cycles. Therefore, the shortest path between all adjacent vertices in \hat{G} is the edge connecting them.

Now we define another graph $G' = (V, E, w')$ where $w'(e) = w(e)$ for all $e \in E \setminus S$ and for all $e \in S$, $w'(e)$ is the length of the shortest path between its end points in \hat{G} or $\|w\|_\infty$ (the maximum edge weight in \hat{G}) if no path exists.

To prove **1.**, it suffices to show G' satisfies a metric, since G' is G with only weights from edges in S modified. For any edge $e \in E$, if $w'(e)$ is the shortest path between its nodes in G' then e is not a heavy edge in G' . Therefore, edges that are in both G' and \hat{G} and edges that are in G' whose weight was set to length of the shortest path between its end points in \hat{G} are not heavy edges. Thus, we only need to look at edges in G' whose weight is $\|w\|_\infty$. These are edges that connect two disconnected components in \hat{G} . Thus, any cycle in G' with such an edge must involve another edge between components which also has weight $\|w\|_\infty$. However, a cycle with two edges of maximum weight cannot be broken, and thus such edges cannot be heavy in G' . Therefore, there are no heavy edges in G' , and so G' satisfies a metric.

To prove **2.**, it now suffices to show that for all $e \in E$, we have that $w'(e) \geq w(e)$. For all $e \in E \setminus S$, we know that $w'(e) = w(e)$. Now, suppose for contradiction that for some $e \in S$, we have $w'(e) < w(e)$. Note if we set $w'(e) = \|w\|_\infty$, then we cannot have $w'(e) < w(e)$. Thus, $w'(e)$ must be the weight of the shortest path between the end points of e in \hat{G} . Let P be this shortest path in \hat{G} . This implies G has a broken cycle $C = P \cup \{e\}$ for which e is the heavy edge. Since S is a light cover, it has a light edge from each broken cycle. So, S must have a light edge from C , but then P could not have existed in \hat{G} , a contradiction. Hence, $w'(e) \geq w(e)$ and we have an increase only solution with such a set S . ◀

Furthermore, given a weighted graph G and a potential support S_W for a solution W , in $O(T_{\text{APSP}})$ time we can determine if there exists a valid (increase only or general) solution on that support, and if so, find one. This is a generalization of Theorem 3, interestingly improving upon the linear programming approach of [9]. Its proof is related to the above theorem, and again uses insight i, though due to space has been moved to Appendix A.2.

■ **Algorithm 1** Verifier.

```

1: function VERIFIER( $G = (V, E, w), S$ )
2:    $M = \|w\|_\infty, \hat{G} = (V, E, \hat{w})$ 
3:   For each  $e \in S$  set  $\hat{w}(e) = M$  and for each  $e \in E \setminus S$ , set  $\hat{w}(e) = w(e)$ 
4:   For each  $(u, v) \in E$ , update  $w(u, v)$  to be length of the shortest path from  $u$  to  $v$  in  $\hat{G}$ 
5:   if Only edges in  $S$  had weights changed (or increased for increase only case) then
6:     return  $w$ 
7:   else
8:     return NULL

```

► **Proposition 7.** *The VERIFIER algorithm, given a weighted graph G and a potential support for a solution S , determines in $O(T_{\text{APSP}})$ time whether there exists a valid (increase only or general) solution on that support and if so finds one.*

3.2 Reducing $\text{MR}(G, \mathbb{R}_{\geq 0})$ to $\text{MR}(G, \mathbb{R})$

We now show that $\text{MR}(G, \mathbb{R}_{\geq 0})$ reduces to $\text{MR}(G, \mathbb{R})$. In later sections, this lets us focus on $\text{MR}(G, \mathbb{R})$ for our algorithms and $\text{MR}(G, \mathbb{R}_{\geq 0})$ for our hardness results. Note that whether an analogous statement holds for the previously studied $G = K_n$ case, is not known, and the following does not immediately imply this as it does not construct a complete graph.

► **Theorem 8.** *There is an approximation-preserving, polynomial-time reduction from $\text{MR}(G, \mathbb{R}_{\geq 0})$ to $\text{MR}(G, \mathbb{R})$.*

Proof. Let $G = (V, E, w)$ be an instance of $\text{MR}(G, \mathbb{R}_{\geq 0})$. Find the set $H = \{(s_1, t_1), \dots, (s_{|H|}, t_{|H|})\}$ of heavy edges of all broken cycles by comparing the weight of each edge to the shortest path distance between its endpoints. We now construct an instance, $G' = (V', E', w)$, of $\text{MR}(G, \mathbb{R})$. For all $1 \leq i \leq |H|$ and $1 \leq j \leq |E| + 1$, let $Q = \{v_{ij}\}_{i,j}$ be a vertex set, and let $F_l = \{(s_i, v_{ij})\}_{i,j}$ and $F_r = \{(t_i, v_{ij})\}_{i,j}$ be edge sets. Let $V' = V \cup Q$ and $E' = E \cup F_l \cup F_r$, where all (s_i, v_{ij}) edges in F_l have weight $Z = 1 + \max_{e \in E} w(e)$, and for any i all (t_i, v_{ij}) edges in F_r have weight $Z - w((s_i, t_i))$.

Let C be any broken cycle in G with heavy edge (s_i, t_i) for some i . First, observe that $C' = (C \setminus (s_i, t_i)) \cup \{(s_i, v_{ij}), (t_i, v_{ij})\}$ is a broken cycle with heavy edge (s_i, v_{ij}) , for any j . To see this, note that $w((s_i, v_{ij})) = Z = w((t_i, v_{ij})) + w((s_i, t_i))$. Thus since C is broken,

$$w((s_i, v_{ij})) = w((t_i, v_{ij})) + w((s_i, t_i)) > w((t_i, v_{ij})) + w(C \setminus (s_i, t_i)),$$

and thus by definition C' is broken with heavy edge (s_i, v_{ij}) . Hence each broken cycle C in G , with heavy edge (s_i, t_i) , corresponds to $|E| + 2$ broken cycles in G' , namely, C itself and the cycles obtained by replacing (s_i, t_i) with a pair $(s_i, v_{ij}), (t_i, v_{ij})$, for any j .

We now show the converse, that any broken cycle C' in G' is either also a broken cycle C in G , or obtained from a broken cycle C in G by replacing (s_i, t_i) with $(s_i, v_{ij}), (t_i, v_{ij})$ for some j . First, observe that for any i , any cycle containing the edge (s_i, v_{ij}) must also contain the edge (t_i, v_{ij}) , and moreover, if a cycle containing such a pair is broken, then its heavy edge must be (s_i, v_{ij}) as $w((s_i, v_{ij})) = Z$. Similarly, any cycle containing more than one of these pairs of edges (over all i and j) is not broken, since such cycles then would contain at least two edges with the maximum edge weight Z . So let C' be any broken cycle containing exactly one such $(s_i, v_{ij}), (t_i, v_{ij})$ pair. Note that C' cannot be the cycle $((s_i, v_{ij}), (t_i, v_{ij}), (s_i, t_i))$, as this cycle is not broken because $w((s_i, v_{ij})) = w((t_i, v_{ij})) + w((s_i, t_i))$. Thus, $C = C' \setminus \{(s_i, v_{ij}), (t_i, v_{ij})\} \cup \{(s_i, t_i)\}$ is a cycle, and C' being broken implies C is broken with heavy edge (s_i, t_i) , implying the claim. This holds since

$$w(s_i, t_i) = w((s_i, v_{ij})) - w((t_i, v_{ij})) > w(C' \setminus (s_i, v_{ij})) - w((t_i, v_{ij})) = w(C \setminus (s_i, t_i)).$$

Now consider any optimal solution M to the $\text{MR}(G, \mathbb{R}_{\geq 0})$ instance G , which by Theorem 6 we know is a minimum cardinality light cover of G . By the above, we know that M is also a light cover of G' , and hence is also a regular cover of G' . Thus by Theorem 6, M is a valid solution to the $\text{MR}(G, \mathbb{R})$ instance. Conversely, consider any optimal solution M' to the $\text{MR}(G, \mathbb{R})$ instance G' , which by Theorem 6 is a minimum cardinality regular cover of G' . The claim is that M' is also a light cover of G , and hence is a valid solution to the $\text{MR}(G, \mathbb{R}_{\geq 0})$ instance. To see this, observe that since all broken cycles in G are broken cycles in G' , M' must be a regular cover of all broken cycles in G , and we now argue that it is in fact a light cover. Specifically, consider all the broken cycles in G which have a common heavy edge (s_i, t_i) . Suppose there is some cycle in this set, call it C , which is not light covered by M' . As M' is a regular cover for G' , this implies that for any j , the broken cycle described above determined by removing the edge (s_i, t_i) from C and adding edges

(s_i, v_{ij}) and (t_i, v_{ij}) , must be covered either with (s_i, v_{ij}) or (t_i, v_{ij}) . However, as j ranges over $|E| + 1$ values, and these edge pairs have distinct edges for different values of j , M' has at least $|E| + 1$ edges. This is a clear contradiction with M' being a minimum sized cover, as any light cover of G is a regular cover of G' , and G only has $|E|$ edges in total. ◀

4 Hardness

Previously, [9] gave an approximation-preserving reduction from Vertex Cover to both $\text{MR}(K_n, \mathbb{R})$ and $\text{MR}(K_n, \mathbb{R}_{\geq 0})$. Thus, both are APX-complete, and in particular are hard to approximate within a factor of $2 - \varepsilon$ for any $\varepsilon > 0$, assuming UGC [11]. Since these hardness results were proven for complete graphs, they also immediately apply to the general problems $\text{MR}(G, \mathbb{R})$ and $\text{MR}(G, \mathbb{R}_{\geq 0})$. Here we give stronger hardness results for $\text{MR}(G, \mathbb{R}_{\geq 0})$ and $\text{MR}(G, \mathbb{R})$ by giving approximation-preserving reductions from MULTICUT and LB-CUT.

► **Problem 4.1 (MULTICUT).** *Given an undirected unweighted graph $G = (V, E)$ on $n = |V|$ vertices together with k pairs of vertices $\{s_i, t_i\}_{i=1}^k$, compute a minimum size subset of edges $M \subseteq E$ whose removal disconnects all the demand pairs, i.e., in the subgraph $(V, E \setminus M)$ every s_i is disconnected from its corresponding vertex t_i .*

Chawla *et al.* [8] proved that if UGC is true, then it is NP-hard to approximate MULTICUT within any constant factor $L > 0$, and assuming a stronger version of UGC, within $\Omega(\sqrt{\log \log n})$. (The MULTICUT version in [8] allowed weights, but they remark their hardness proofs extend to the unweighted case.)

► **Theorem 9.** *There is an approximation-preserving, polynomial-time reduction from MULTICUT to $\text{MR}(G, \mathbb{R}_{\geq 0})$.*

Proof. Let $G = (V, E)$ be an instance of MULTICUT with k pairs of vertices $\{s_i, t_i\}_{i=1}^k$. First, if $(s_i, t_i) \in E$ for any i , then that edge must be included in the solution M . Thus, we can assume no such edges exists in the MULTICUT instance, as assuming this can only make it harder to approximate the optimum value of the MULTICUT instance. We now construct an instance of $\text{MR}(G, \mathbb{R}_{\geq 0})$, $G' = (V', E', w)$. Let $V' = V$ and $E' = E \cup \{s_i, t_i\}_{i=1}^k$ where the edges in E have weight one and the edges (s_i, t_i) , for all $i \in [k]$, have weight $n = |V|$.

If a cycle in G' has exactly one edge of weight n , then it must be broken since there can be at most $n - 1$ other edges in the cycle. Conversely, if a cycle C has no edge or more than one edge with weight n , then C does not have a heavy edge, and so is not broken.

Note that the edges from G are exactly the weight one edges in G' , and thus, the paths in G are in one-to-one correspondence with the paths in G' which consist of only weight one edges. Moreover, the weight n edges in G' are in one-to-one correspondence with the (s_i, t_i) pairs from G . Thus, the cycles in G' with exactly one weight n edge followed by paths of all weight one edges connecting their endpoints, which by the above are exactly the set of broken cycles, are in one-to-one correspondence with paths between (s_i, t_i) pairs from G . Therefore, a minimum cardinality subset of edges which light cover all broken cycles, i.e., an optimal $\text{MR}(G, \mathbb{R}_{\geq 0})$ support, corresponds to a minimum cardinality subset of edges from E which cover all paths from s_i to t_i for all i , i.e., an optimal solution to MULTICUT. ◀

► **Problem 4.2 (LB-CUT).** *Given a value L and an undirected unweighted graph $G = (V, E)$ with source s and sink t , find a minimum size subset of edges $M \subseteq E$ such that no s - t -path of length less than or equal to L remains in the graph after removing the edges in M .*

An instance of LB-CUT with length L , is referred to as an instance of L -LB-CUT. For any fixed L , Lee [12] showed that it is hard to approximate L -LB-CUT within a factor of $\Omega(\sqrt{L})$. Using a similar reduction as above, we argue the following.

► **Theorem 10.** *For any fixed value L , there is an approximation-preserving, polynomial-time reduction from L -LB-CUT to $MR(G, \mathbb{R}_{\geq 0})$.*

Proof. Let $G = (V, E)$ be an instance of L -LB-CUT with source s and sink t . First, if $(s, t) \in E$, then that edge must be included in the solution M . Thus we can assume that edge is not in the LB-CUT instance, as assuming this can only make it harder to approximate the optimum value of the LB-CUT instance. We now construct an instance of $MR(G, \mathbb{R}_{\geq 0})$, $G' = (V', E', w)$. Let $V' = V$ and $E' = E \cup \{(s, t)\}$ where the edges in E have weight 1 and the edge (s, t) has weight $L + 1$.

First, observe that any cycle containing the edge (s, t) followed by $\leq L$ unit weight edges is broken, as the sum of the unit weight edges will be $< L + 1 = w((s, t))$. Conversely, any broken cycle must contain the edge (s, t) followed by $\leq L$ unit weight edges. Specifically, if a cycle does not contain (s, t) then it is unbroken since all edges would then have weight 1. Moreover, if a cycle contains (s, t) and $> L$ other edges, then the total sum of those unit edges will be $\geq L + 1 = w((s, t))$.

Note that the edges from G are exactly the weight one edges in G' , and thus the paths in G are in one-to-one correspondence with the paths in G' which consist of only weight one edges. Moreover, the edge (s, t) in G' corresponds with the source and sink from G . Thus by the above, the broken cycles in G' are in one-to-one correspondence with s - t -paths with length $\leq L$ in G . Therefore, a minimum cardinality subset of edges which light cover all broken cycles, i.e., an optimal support to $MR(G, \mathbb{R}_{\geq 0})$, corresponds to a minimum cardinality subset of edges from E which cover all paths from s to t of length $\leq L$, i.e., an optimal solution to LB-CUT. ◀

In the L -LB-CUT to $MR(G, \mathbb{R}_{\geq 0})$ reduction of Theorem 10, one edge (the s, t pair) has weight $L + 1$ and all other edges have unit weight. Moreover, in the reduction from $MR(G, \mathbb{R}_{\geq 0})$ to $MR(G, \mathbb{R})$ of Theorem 8, the max edge weight increases by 1. Thus, by these reductions, and previous hardness results, we have the following summarizing theorem.

► **Theorem 11.** *$MR(G, \mathbb{R}_{\geq 0})$ and $MR(G, \mathbb{R})$ are APX-complete, and moreover assuming UGC neither can be approximated within any constant factor.*

For any positive integer L , consider the problem defined by the restriction of $MR(G, \mathbb{R})$ to integer weight instances with maximum edge weight L and minimum edge weight 1, or the further restriction of $MR(G, \mathbb{R}_{\geq 0})$ to instances where all weights are 1 except for a single weight L edge. Then assuming UGC these problems are hard to approximate within $\Omega(\sqrt{L})$.

5 Fixed Parameter Analysis for ζ -Chordal Graphs

Let ζ be a fixed constant, and let F_ζ be the family of all ζ -chordal graphs. Here we provide an FPT for $MR(G, \mathbb{R})$ for any $G \in F_\zeta$, parameterized on the optimal solution size OPT .

By Theorem 6, we seek a minimum sized cover of all broken cycles. First, we argue below that if G has a broken cycle, then it has a broken chordless cycle. This seems to imply a natural FPT algorithm for constant ζ . Namely, find an uncovered broken chordless cycle and recursively try adding each one of its edges to our current solution.¹ However, it is possible

¹ One might construe this as FPT kernelization. The edges of the broken chordless cycles do form a kernel but its size is not bounded in our parameter. As an example, take $G = K_n$, set one edge weight to $n + 1$, and all other weights to 1. There are $2n - 3$ edges in the kernel while the optimal solution has size 1.

25:10 Generalized Metric Repair on Graphs

to cover all broken chordless cycles while not covering the broken chorded cycles. These cycles are difficult to cover as they may be much larger than ς , though again by Theorem 6 they must be covered.

Consider an optimal solution W , with support S_W . Suppose that we have found a subset $S \subsetneq S_W$, covering all broken chordless cycles in G . Intuitively, if we add to each edge in S its weight from W , then any remaining broken chordless cycle must be covered further, in effect revealing which edges to consider from the chorded cycles from the original graph G . The challenge, however, is of course that we don't know W a priori. We argue that despite this one can still identify a bounded sized subset of edges containing an edge from a cycle needing to be covered further.

► **Lemma 12.** *If G has a broken cycle, then G has a broken chordless cycle.*

Proof. Let $C = v_1, \dots, v_k$ be the broken cycle in G with the fewest edges, with v_1v_k being the heavy edge. If C is chordless, then the claim holds. Otherwise, this cycle has at least one chord v_iv_j . Now there are two paths P_1 and P_2 from v_i to v_j on the cycle. Let P_1 be the path containing the heavy edge of C . If $w(v_iv_j) > \sum_{e \in P_2} w(e)$, then P_2 together with the edge v_iv_j defines a broken cycle with fewer edges than C . On the other hand, if $w(v_iv_j) \leq \sum_{e \in P_2} w(e)$ then P_1 together with the edge v_iv_j defines a broken cycle with fewer edges than C . In either case we get a contradiction as C was the broken cycle with the fewest edges. ◀

Our FPT is shown in Algorithm 2, where we recursively build a potential support S up to our current guess at the optimal size k . The following lemma is key to arguing correctness.

■ **Algorithm 2** FPT.

```

1: function F( $G, S, k$ )
2:   if  $|S| = k$  then return VERIFIER( $G, S$ )
3:    $P = \emptyset$ 
4:   if there exists a broken chordless cycle  $C$  such that  $C \cap S = \emptyset$  then  $P = C$ 
5:   else
6:     for  $s \subseteq S$  such that  $|s| \leq \varsigma - 1$  do
7:       Let  $\mathcal{C} = \{\text{Chordless cycles } C \text{ such that } C \cap S = s\}$ 
8:        $C_1 \leftarrow \arg \min_{C \in \mathcal{C}} \sum_{e \in C \setminus s} w(e)$ 
9:        $C_2 \leftarrow \arg \max_{C \in \mathcal{C}} w(h) - \sum_{e \in C \setminus (s \cup \{h\})} w(e)$ , where  $h = \arg \max_{f \in C \setminus s} w(f)$ 
10:      Add  $(C_1 \cup C_2) \setminus S$  to  $P$ 
11:   for  $e \in P$  do
12:      $X = \text{F}(G, S \cup \{e\}, k)$ 
13:     if  $X \neq \text{NULL}$  then return  $X$ 
14:   return NULL

15: function FPTWRAPPER( $G$ )
16:   for  $k = 1, 2, \dots$  do
17:      $X = \text{F}(G, \emptyset, k)$ 
18:     if  $X \neq \text{NULL}$  then return  $X$ 

```

► **Lemma 13.** *Consider any optimal solution W and its support S_W to an instance of metric repair for $G = (V, E, w) \in F_\varsigma$. If $S \subsetneq S_W$, then $\text{F}(G, S, \text{OPT})$ adds at least one edge in $S_W \setminus S$ to P .*

Proof. Consider the auxiliary graph $G_S = (V, E, \tilde{w})$, which has the same vertex and edge sets as G , but with the modified weight function:

$$\tilde{w} = \begin{cases} w(e) & e \notin S \\ W(e) + w(e) & e \in S \end{cases}$$

Since $S \subsetneq S_W$, we have that G_S has a broken cycle. Thus, by Lemma 12, G_S has a chordless broken cycle. Suppose there is a chordless broken cycle in G_S that is edge disjoint from S (which occurs if and only if it is also broken in G), in which case, line 4 finds such a cycle. As this is a broken cycle, it must be covered by some edge in $S_W \setminus S$, and thus, we have added some edge in $S_W \setminus S$ to P .

Let us assume otherwise, that any chordless broken cycle in G_S has non-empty intersection with S . Let C be any such chordless broken cycle with $C \cap S \neq \emptyset$. Observe that as C is broken in G_S , it must be that $|C \cap S| < |C|$, as otherwise it would imply W was not a solution. Thus, as $G \in F_\varsigma$, we know that $|C| \leq \varsigma$, and so $|C \cap S| < \varsigma$. This implies in some for loop iteration, $C \in \mathcal{C}$ on line 7.

Let h be the heavy edge, in G_S , of the broken cycle C . We now have two cases:

Case 1: $h \in S$. In this case we have that

$$W(h) + w(h) > \underbrace{\sum_{e \in C \setminus S} w(e)}_{(1)} + \sum_{e \in S} W(e) + w(e).$$

On line 8 we found a cycle C_1 that minimized (1). Thus, since C is broken in G_S , C_1 is also broken in G_S , and so must be covered by some edge in $S_W \setminus S$. Hence, we added some edge in $S_W \setminus S$ to P .

Case 2 $h \notin S$. In this case h has the maximum weight of all edges in $C \setminus s$. We have that

$$w(h) - \underbrace{\sum_{e \in C \setminus (S \cup \{h\})} w(e)}_{(2)} > \sum_{e \in S} W(e) + w(e).$$

On line 9 we found a cycle C_2 maximizing (2). Thus, if C is broken in G_S , then C_2 is broken in G_S , and so must be covered by some edge in $S_W \setminus S$. Hence, we added some edge in $S_W \setminus S$ to P . ◀

► **Lemma 14.** *Any time we call F , we have that $|P| \leq 2\varsigma|S|^\varsigma$*

Proof. Note $|P|$ is upper bounded by ς multiplied by the number of chordless cycles we add. If the conditional on line 4 is true then we add only a single chordless cycle to P . Otherwise, for each $s \subseteq S$ such that $|s| \leq \varsigma - 1$ we find two cycles. There are at most

$$\sum_{i=1}^{\varsigma-1} \binom{|S|}{i} \leq \sum_{i=1}^{\varsigma-1} |S|^i \leq |S|^\varsigma$$

many such subsets, and thus we add at most $2|S|^\varsigma$ many cycles, implying the claim. ◀

► **Theorem 15.** *For any fixed constant ς , Algorithm 2 is an FPT algorithm for $MR(G, \mathbb{R})$ for any $G \in F_\varsigma$, when parameterized by OPT . The running time is $\Theta((2\varsigma OPT^\varsigma)^{OPT+1} n^\varsigma)$.*

25:12 Generalized Metric Repair on Graphs

Proof. FPTWRAPPER iteratively calls $F(G, \emptyset, k)$ for increasing values of k until it returns a non-NULL value. First, we argue that while $k < OPT$, $F(G, \emptyset, k)$ will return NULL. In the initial call to F , we have $S = \emptyset$. F then adds exactly one edge in each recursive call until $|S| = k$, at which point it returns $\text{VERIFIER}(G, S)$. Thus, as $k < OPT$, by proposition 7, NULL is returned.

Now we argue that when $k = OPT$ an optimal solution is returned. Fix any optimal solution W and its support S_W to the given instance G . By Lemma 13, if $S \subsetneq S_W$ (which is true initially as $S = \emptyset$) then at least one edge in $S_W \setminus S$ is added to P . Thus, as F makes a recursive call to $F(G, S \cup \{e\}, k)$ for every edge $e \in P$, in at least one recursive call an edge of S_W is added to S . Thus there is some path in the tree of recursive calls to F in which all $k = OPT$ edges from S_W are added, at which point F returns $\text{VERIFIER}(G, S)$, which returns an optimal solution by proposition 7. (Note this recursive call may not be reached, if a different optimal solution is found first.)

Now we consider bounding the running time. Observe that in each call to F , a set P is constructed, and then recursive calls to $F(G, S \cup \{e\}, k)$ are made for each $e \in P$. By Lemma 14, $|P| \leq 2\zeta|S|^\zeta \leq 2\zeta k^\zeta$ at all times. So in the tree of all recursive calls made by any initial call to $F(G, \emptyset, k)$, the branching factor is always bounded by $2\zeta k^\zeta$, and the depth is k . Thus there are $O((2\zeta k^\zeta)^k)$ nodes in our recursion tree.

Now we bound the time needed for each node in the recursion tree. If VERIFIER is called then it takes $O(T_{\text{APSP}})$ time by proposition 7. Otherwise, note that there are $O(n^\zeta)$ chordless cycles. Thus it takes $O(\zeta n^\zeta)$ time to enumerate and check them on line 4. Similarly $|\mathcal{C}| = O(n^\zeta)$ on line 7, and so the run time of each iteration of the for loop is $O(\zeta n^\zeta)$. There are $O(|S|^\zeta) = O(k^\zeta)$ iterations of the for loop, thus the total time per node is $O(\zeta k^\zeta n^\zeta)$.

Thus the total time for each call to $F(G, \emptyset, k)$ is $O((2\zeta k^\zeta)^k \zeta k^\zeta n^\zeta) = O((2\zeta k^\zeta)^{k+1} n^\zeta)$. Since FPTWRAPPER calls $F(G, \emptyset, k)$ for $k = 1, \dots, OPT$, the overall running time is

$$O\left(\left(\sum_{k=1}^{OPT} (2\zeta k^\zeta)^{k+1}\right) \cdot n^\zeta\right) = O((2\zeta OPT^\zeta)^{OPT+1} n^\zeta). \quad \blacktriangleleft$$

As lemma 13 holds for any optimal solution, the bound on the recursion tree size in the above proof actually bounds the number of optimal solutions.

► **Corollary 16.** *If $G \in \mathcal{F}_\zeta$ then there are at most $(2\zeta OPT^\zeta)^{OPT}$ subsets $S \subset E$ such that S is the support of an optimal solution to $\text{MR}(G, \mathbb{R})$.*

► **Remark 17.** Using the approximation-preserving reduction from $\text{MR}(G, \mathbb{R}_{\geq 0})$ to $\text{MR}(G, \mathbb{R})$ in Theorem 8, the above also yields an FPT for $\text{MR}(G, \mathbb{R}_{\geq 0})$. This holds since the reduction does not change the optimal solution size, nor ζ as it only adds triangles. Alternatively, the above algorithm can be carefully modified to consider light covering broken cycles.

6 Approximation Algorithms

In this section we present approximation algorithms for $\text{MR}(G, \mathbb{R}_{\geq 0})$ and $\text{MR}(G, \mathbb{R})$.

By Theorem 6, the support of an optimal solution to $\text{MR}(G, \mathbb{R})$ is a minimum cardinality regular cover of all broken cycles. This naturally defines a hitting set instance (E, \mathcal{C}) , where the ground set E is the edges from G , and \mathcal{C} is the collection of the subsets of edges determined by broken cycles. Unfortunately, constructing (E, \mathcal{C}) explicitly is infeasible as there may be an exponential number of broken cycles. In general just counting the number of paths in a graph is #P-Hard [16], though it is known how to count paths of length up to roughly $O(\log n)$

using color-coding. (See [1, 4] and references therein.) Moreover, observe our situation is more convoluted as we wish to count only paths corresponding to broken cycles.

Despite these challenges, we argue there is sufficient structure to at least roughly apply the standard greedy algorithms for hitting set. Our first key insight, related to insight i, is:

- (ii) One can always find *some* broken cycle, if one exists, by finding any edge whose weight is more than the shortest path length between its endpoints (using APSP).

Thus we have a polynomial time oracle, returning an arbitrary set in \mathcal{C} . Recall the greedy algorithm for hitting set, which repeatedly picks an arbitrary uncovered set, and adds all its elements to the solution. If $L = \max_{c \in \mathcal{C}} |c|$ is the largest set size, this gives an L -approximation, as each time we take the elements of a set, we get at least one element of the optimal solution. Below we apply this approach to approximate $\text{MR}(G, \mathbb{R})$ and $\text{MR}(G, \mathbb{R}_{\geq 0})$.

We would prefer, however, to have an oracle for the number of broken cycles that an edge $e \in E$ participates in as using such an oracle would yield an $O(\log n)$ -approximation algorithm for $\text{MR}(G, \mathbb{R})$ (regardless of the size of L) by running the standard greedy algorithm for hitting set which repeatedly selects the element that hits the largest number of uncovered sets. Towards this end, we have the following key insight:

- (iii) We can find the *most* broken cycle (i.e., with maximum deficit) and, more importantly, count how many such maximum deficit cycles each edge is in.

To argue that insight iii is true, first we observe that the cycle with the largest deficit value corresponds to a shortest path. This in turn, argued over several lemmas, allows us to quickly get a count when restricting to such cycles. Thus, if κ denotes the number of distinct cycle deficit values, the above insight implies an $O(\kappa \log n)$ -approximation, by breaking the problem into κ hitting set instances, where for each instance we can run the greedy algorithm.

6.1 L -approximation

In this section, we consider the problems defined by restricting $\text{MR}(G, \mathbb{R})$ and $\text{MR}(G, \mathbb{R}_{\geq 0})$ to the subset of instances where the largest number of light edges in a broken cycle is L . We present an $(L + 1)$ -approximation algorithm for $\text{MR}(G, \mathbb{R})$ which runs in $O(T_{\text{APSP}} \cdot \text{OPT})$ time, which also will imply an L -approximation for $\text{MR}(G, \mathbb{R}_{\geq 0})$ with the same running time.

As mentioned above, the main idea comes from insight ii. In particular, the following algorithm, SHORT PATH COVER (SPC), can be easily understood by viewing it as running the standard L -approximation for the corresponding instance (E, \mathcal{C}) of hitting set, where we have an oracle for finding a set $c \in \mathcal{C}$. In the following, APSP is a subroutine returning a shortest path distance function $d(u, v)$, and a function $P(u, v)$ giving the set of edges along *any* shortest path from u to v .

■ **Algorithm 3** Short Path Cover (SPC) for $\text{MR}(G, \mathbb{R})$.

```

1: function SPC( $G = (V, E, w)$ )
2:    $H = (V_H = V, E_H = E, w_H = w)$ 
3:   while True do
4:      $d, P = \text{APSP}(H)$ 
5:     if  $\exists e = (u, v) \in E_H$  such that  $w(e) > d(u, v)$  then  $E_H = E_H \setminus (P(u, v) \cup \{e\})$ 
6:     else return VERIFIER( $G, E \setminus E_H$ )

```

► **Theorem 18.** *SPC gives an $(L + 1)$ -approximation for $\text{MR}(G, \mathbb{R})$ in $O(T_{\text{APSP}} \cdot \text{OPT})$ time.*

Proof. First, note that if there is a broken cycle in H , then for some edge $e = (u, v)$, $w(e) > d(u, v)$, and moreover, in this case $P(u, v) \cup \{e\}$ is a broken cycle. Thus, when the

algorithm terminates there are no broken cycles in H . Also, for any broken cycle in G , if all of its edges are still in H , then it will be a broken cycle in H . Thus, when the algorithm terminates at least one edge from each broken cycle in G is in $E \setminus E_H$, which by Theorem 6 implies $E \setminus E_H$ is a valid support.

Note that removing edges does not create any new broken cycles, thus, any broken cycle in H is also a broken cycle in G . Thus, the support of any optimum solution must contain at least one edge from each broken cycle in H (again by Theorem 6), and so every time we remove the edges of a broken cycle $P(u, v) \cup \{e\}$, we remove at least one optimum edge. As the largest broken cycle length is $L + 1$, this implies overall we get an $(L + 1)$ -approximation. The same argument implies the while loop can get executed at most OPT times, and as APSP takes $O(T_{\text{APSP}})$ time, and line 5 takes $O(m)$ time, we obtain the running time in the theorem statement. ◀

► **Remark 19.** If we modify SPC so that in line 5 we only remove $P(u, v)$ from E_H (rather than $P(u, v) \cup \{e\}$), then by the second part of Theorem 6, the same argument implies that SPC is an L -approximation for $\text{MR}(G, \mathbb{R}_{\geq 0})$ that runs in $O(T_{\text{APSP}} \cdot OPT)$ time.

► **Remark 20.** Theorem 11 restricts $\text{MR}(G, \mathbb{R}_{\geq 0})$ and $\text{MR}(G, \mathbb{R})$ to integer instances with max weight L , implying any broken cycle has $\leq L$ edges. As this is a subset of the instances here, SPC is an L or $L + 1$ approx for instances that are hard to approximate within $\Omega(\sqrt{L})$.

6.2 $O(\kappa \log n)$ -approximation

Using insight iii, our approach is to iteratively cover cycles by decreasing deficit value, ultimately breaking the problem into multiple hitting set instances. We present the algorithm for $\text{MR}(G, \mathbb{R})$ first and then remark on the minor change needed to apply it to $\text{MR}(G, \mathbb{R}_{\geq 0})$.

For any pair of vertices $s, t \in V$, let $d(s, t)$ denote their shortest path distance in G , and $\#\text{sp}(s, t)$ denote the number of shortest paths from s to t . It is straightforward to show that $\#\text{sp}(s, t)$ can be computed in $O(m + n)$ time given all $d(u, v)$ values have been precomputed.

► **Lemma 21.** *Let G be a positively weighted graph, where for all pairs of vertices u, v one has constant time access to the value $d(u, v)$. Then for any pair of vertices s, t , the value $\#\text{sp}(s, t)$ can be computed in $O(m + n)$ time.*

Recall that for a broken cycle C with heavy edge h , the deficit of C is $\delta(C) = w(h) - \sum_{e \in (C \setminus h)} w(e)$. Moreover, $\delta(G)$ denotes the maximum deficit over all cycles in G . For any edge e , define $N_h(e, \alpha)$ to be the number of distinct broken cycles of deficit α whose heavy edge is e . Similarly, let $N_l(e, \alpha)$ denote the number of distinct broken cycles with deficit α which contain the edge e , but where e is not the heavy edge. While for general α it is not clear how to even approximate $N_l(e, \alpha)$ and $N_h(e, \alpha)$, we argue that when $\alpha = \delta(G)$ these values can be computed exactly.

► **Lemma 22.** *For any edge $e = (s, t)$, if $w(e) = d(s, t) + \delta(G)$ then $N_h(e, \delta(G)) = \#\text{sp}(s, t)$, and otherwise $N_h(e, \delta(G)) = 0$.*

Proof. If $w(e) \neq d(s, t) + \delta(G)$, then as $\delta(G)$ is the maximum deficit over all cycles, it must be that $w(e) < d(s, t) + \delta(G)$, which in turn implies any broken cycle with heavy edge e has deficit strictly less than $\delta(G)$. Now suppose $w(e) = d(s, t) + \delta(G)$, and consider any path $p_{s,t}$ from s to t such that e together with $p_{s,t}$ creates a broken cycle with heavy edge e . If $p_{s,t}$ is a shortest path then $w(e) - w(p_{s,t}) = w(e) - d(s, t) = \delta(G)$, and otherwise $w(p_{s,t}) > d(s, t)$ and so $w(e) - w(p_{s,t}) < w(e) - d(s, t) = \delta(G)$. Thus $N_h(e, \delta(G)) = \#\text{sp}(s, t)$ as claimed. ◀

As G is undirected, every edge $e \in E$ correspond to some unordered pair $\{a, b\}$. However, often we write $e = (a, b)$ as an ordered pair, according to some fixed arbitrary total ordering of all the vertices. We point this out to clarify the following statement.

► **Lemma 23.** *Fix any edge $e = (s, t)$, and let $X = \{f = (a, b) \mid w(f) = d(a, s) + w(e) + d(t, b) + \delta(G)\}$, and $Y = \{f = (a, b) \mid w(f) = d(b, s) + w(e) + d(t, a) + \delta(G)\}$. Then*

$$N_l(e, \delta(G)) = \left(\sum_{(a,b) \in X} \#sp(a, s) \cdot \#sp(t, b) \right) + \left(\sum_{(a,b) \in Y} \#sp(b, s) \cdot \#sp(t, a) \right).$$

Proof. Consider any broken cycle C containing $e = (s, t)$, with heavy edge $f = (a, b)$ and where $\delta(C) = \delta(G)$. Such a cycle must contain a shortest path between a and b , as otherwise it would imply $\delta(G) > \delta(C)$. Now if we order the vertices cyclically, then the subset of C 's vertices $\{a, b, s, t\}$, must appear either in the order a, s, t, b or b, s, t, a . In the former case, as the cycle must use shortest paths, $w(f) = d(a, s) + w(e) + d(t, b) + \delta(G)$, and the number of cycles satisfying this is $\#sp(a, s) \cdot \#sp(t, b)$. In the latter case, $w(f) = d(b, s) + w(e) + d(t, a) + \delta(G)$, and the number of cycles satisfying this is $\#sp(b, s) \cdot \#sp(t, a)$. Note also that the set X from the lemma statement is the set of all $f = (a, b)$ satisfying the equation in the former direction, and Y is the set of all $f = (a, b)$ satisfying the equation in the later direction. Thus summing over each relevant heavy edge in X and Y , of the number of broken cycles of deficit $\delta(G)$ which involve that heavy edge and e , yields the total number of broken cycles with deficit $\delta(G)$ containing e as a light edge. ◀

► **Corollary 24.** *Given constant time access to $d(u, v)$ and $\#sp(u, v)$ for any vertices u and v , $N_h(e, \delta(G))$ can be computed in $O(1)$ time and $N_l(e, \delta(G))$ in $O(m)$ time.*

■ **Algorithm 4** Finds a valid solution for $MR(G, \mathbb{R})$.

```

1: function APPROX( $G = (V, E, w)$ )
2:   Let  $S = \emptyset$ 
3:   while True do
4:     For every pair  $s, t \in V$  compute  $d(s, t)$ 
5:     Compute  $\delta(G) = \max_{e=(s,t) \in E} w(e) - d(s, t)$ 
6:     if  $\delta(G) = 0$  then return VERIFIER( $G, S$ )
7:     For every edge  $(s, t) \in E$  compute  $\#sp(s, t)$ 
8:     For every  $e \in E$  compute  $count(e) = N_h(e, \delta(G)) + N_l(e, \delta(G))$ 
9:     Set  $f = \arg \max_{e \in E} count(e)$ 
10:    Update  $S = S \cup \{f\}$  and  $G = G \setminus f$ 

```

► **Theorem 25.** *For any positive integer κ , consider the set of $MR(G, \mathbb{R})$ instances where the number of distinct deficit values is at most κ , i.e., $|\{\delta(C) \mid C \text{ is a cycle in } G\}| \leq \kappa$. Then Algorithm 4 gives an $O((T_{\text{APSP}} + m^2) \cdot OPT \cdot \kappa \log n)$ time $O(\kappa \log n)$ -approximation.*

Proof. Observe that the algorithm terminates only when $\delta(G) = 0$, i.e., only once there are no broken cycles left. As no new edges are added, and weights are never modified, this implies that when the algorithm terminates it outputs a valid regular cover S . (The algorithm must terminate as every round removes an edge.) Therefore, by Theorem 6, S is a valid $MR(G, \mathbb{R})$ support, and so we only need to bound its size.

25:16 Generalized Metric Repair on Graphs

Let the edges in $S = \{s_1, \dots, s_k\}$ be indexed in increasing order of the loop iteration in which they were selected. Let G_1, \dots, G_{k+1} be the corresponding sequence of graphs produced by the algorithm, where $G_i = G \setminus \{s_1, \dots, s_{i-1}\}$. Note that for all i , $G_i = (V, E_i)$ induces a corresponding instance of hitting set, (E_i, \mathcal{C}_i) , where the ground set is the set of edges from the $\text{MR}(G, \mathbb{R})$ instance G_i , and $\mathcal{C}_i = \{E_i(C) \mid C \text{ is a broken cycle in } G_i\}$ (where $E_i(C)$ is the set of edges in C).

Let $D = \{\delta(C) \mid C \text{ is a cycle in } G\}$, where by assumption $|D| \leq \kappa$. Note that any cycle C in any graph G_i , is also a cycle in G . Thus as we never modify edge weights, $\delta(G_1), \dots, \delta(G_{k+1})$ is a non-increasing sequence. Moreover $X = \{\delta(G_i)\}_i \subseteq D$, and in particular $|X| \leq \kappa$. For a given value $\delta \in X$, let $G_\alpha, G_{\alpha+1}, \dots, G_\beta$ be the subsequence of graphs with deficit δ (which is consecutive as the deficit values are non-increasing). Observe that for all $\alpha \leq i \leq \beta$, the edge s_i is an edge from a cycle with deficit $\delta = \delta(G_i)$. So for each $\alpha \leq i \leq \beta$, define a sub-instance of hitting set (E'_i, \mathcal{C}'_i) , where E'_i is the set of edges in cycles of deficit δ from G_i , and \mathcal{C}'_i is the family of sets of edges from each cycle of deficit δ in G_i .

The claim is that for the hitting set instance $(E'_\alpha, \mathcal{C}'_\alpha)$, that $\{s_\alpha, \dots, s_\beta\}$ is an $O(\log n)$ approximation to the optimal solution. To see this, observe that for any $\alpha \leq i \leq \beta$ in line 8, $\text{count}(e)$ is the number of times e is contained in a broken cycle with deficit $\delta = \delta(G_i)$, as by definition $N_h(e, \delta(G_i))$ and $N_l(e, \delta(G_i))$ count the occurrences of e in such cycles as a heavy edge or light edge, respectively. Thus s_i is the edge in E'_i which hits the largest number of sets in \mathcal{C}'_i , and moreover, $(E'_{i+1}, \mathcal{C}'_{i+1})$ is the corresponding hitting set instance induced by removing s_i and the sets it hit from (E'_i, \mathcal{C}'_i) . Thus $\{s_\alpha, \dots, s_\beta\}$ is the resulting output of running the standard greedy hitting set algorithm on $(E'_\alpha, \mathcal{C}'_\alpha)$ (that repeatedly removes the element hitting the largest number of sets), and it is well known this greedy algorithm produces an $O(\log n)$ approximation.

The bound on the size of S now easily follows. Specifically, let $I = \{i_1, i_2, \dots, i_{|X|}\}$ be the collection of indices, where i_j was the first graph considered with deficit $\delta(G_{i_j})$. By the above, S is the union of the $O(\log n)$ -approximations to the sequence of hitting set instance $(E'_{i_1}, \mathcal{C}'_{i_1}), \dots, (E'_{i_{|X|}}, \mathcal{C}'_{i_{|X|}})$. In particular, note that for all i_j , $(E'_{i_j}, \mathcal{C}'_{i_j})$ is a hitting set instance induced from the removal of a subset of edges from the initial hitting set instance (E_1, \mathcal{C}_1) , and then further restricted to sets from cycles with a given deficit value. Thus the size of the optimal solution on each of these instances can only be smaller than on (E_1, \mathcal{C}_1) . This implies that the total size of the returned set S is $O(\text{OPT} \cdot |X| \log n) = O(\text{OPT} \cdot \kappa \log n)$.

As for the running time, first observe that by the above, there are $O(\text{OPT} \cdot \kappa \log n)$ while loop iterations. Next, the single call to `VERIFIER` in line 6 takes $O(T_{\text{APSP}})$. For a given loop iteration, computing all pairwise distances in line 4 also takes $O(T_{\text{APSP}})$ time. Computing the graph deficit in line 5 can then be done in $O(m)$ time. For any given vertex pair s, t , computing $\#\text{sp}(s, t)$ takes $O(m + n)$ time by Lemma 21. Thus computing the number of shortest paths over all edges in line 7 takes $O(m^2 + mn)$ time. For each edge e , by Corollary 24, $\text{count}(e) = N_h(e, \delta(G)) + N_l(e, \delta(G))$ can be computed in $O(m)$ time, and thus computing all counts in line 8 takes $O(m^2)$ time. As the remaining steps can be computed in linear time, each while loop iteration in total takes $O(T_{\text{APSP}} + mn + m^2) = O(T_{\text{APSP}} + m^2)$ time, thus implying the running time bound over all iterations in the theorem statement. ◀

► **Remark 26.** If we modify line 8 to instead set $\text{count}(e) = N_l(e, \delta(G))$, by Theorem 6, we get the same result for $\text{MR}(G, \mathbb{R}_{\geq 0})$. If instead we used the reduction from $\text{MR}(G, \mathbb{R}_{\geq 0})$ to $\text{MR}(G, \mathbb{R})$ of Theorem 8, the graph size increases by a linear factor, giving a slower run time.

References

- 1 N. Alon and S. Gutner. Balanced families of perfect hash functions and their applications. *ACM Trans. Algorithms*, 6(3):54:1–54:12, 2010.
- 2 S. Baraty, D. Simovici, and C. Zara. The impact of triangular inequality violations on medoid-based clustering. In *Foundations of Intelligent Systems*, pages 280–289. Springer, 2011.
- 3 M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003. doi:10.1162/089976603321780317.
- 4 C. Brand, H. Dell, and T. Husfeldt. Extensor-coding. In *Symposium on Theory of Computing (STOC)*, pages 151–164, 2018.
- 5 J. Brickell, I Dhillon, S. Sra, and J. Tropp. The metric nearness problem. *SIAM Journal on Matrix Analysis and Applications*, 30(1):375–396, 2008.
- 6 E. Candès and B. Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, June 2012.
- 7 L. Sunil Chandran, Vadim V. Lozin, and C. R. Subramanian. Graphs of low chordality. *Discrete Mathematics and Theoretical Computer Science*, 7:25–36, 2005.
- 8 S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.
- 9 C. Fan, B. Raichel, and G. Van Buskirk. Metric violation distance: Hardness and approximation. In *Symposium on Discrete Algorithms (SODA)*, pages 196–209, 2018.
- 10 A. Gilbert and L. Jain. If it ain’t broke, don’t fix it: Sparse metric repair. *ArXiv*, 2017.
- 11 S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.
- 12 E. Lee. Improved hardness for cut, interdiction, and firefighter problems. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 92:1–92:14, 2017.
- 13 S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. doi:10.1126/science.290.5500.2323.
- 14 A. Sidiropoulos, D. Wang, and Y. Wang. Metric embeddings with outliers. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 670–689, 2017.
- 15 J. Tenenbaum, V. Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. doi:10.1126/science.290.5500.2319.
- 16 L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
- 17 F. Wang and J. Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data Mining and Knowledge Discovery*, 29(2):534–564, March 2015.

A Transitioning to Graph Metric Repair

A.1 The decrease only case

For the problem $\text{MR}(G, \mathbb{R}_{\leq 0})$, consider the following simple algorithm, used in previous works for the special case when $G = K_n$.

■ **Algorithm 5** Decrease Metric Repair (DMR).

-
- 1: **function** DMR($G = (V, E, w)$)
 - 2: Let $W = w$
 - 3: For any edge $e = uv \in E$, set $W(e) =$ weight of a shortest path between u and v
 - 4: **return** $W - w$
-

► **Theorem 5.** *The problem $MR(G, \mathbb{R}_{\leq 0})$ can be solved in $O(T_{\text{APSP}})$ time.*

Moreover, the problem becomes hard if even a single positive value is allowed. That is, if $0 \in \Omega$ and $\Omega \cap \mathbb{R}_{>0} \neq \emptyset$ then $MR(G, \Omega)$ is APX-Complete.

Proof. For the first part, let $e \in G$ be an edge whose edge weight is bigger than the shortest path between the two end points of e . Then in this case e is the heavy edge in a broken cycle. Hence, any decrease only solution must decrease this edge. Thus all edges decreased by DMR are edges that must be decreased.

By the same reasoning we see that this new weighted graph has no broken cycles. Thus, we see that our algorithm gives a sparsest solution to $MR(G, \mathbb{R}_{\leq 0})$ in $\Theta(T_{\text{APSP}})$ time.

For the second part, the reduction is the same as that of Fan et al. [9]. However, we make the observation that for any value $\alpha > 0$, by appropriately scaling the weights of the reduction in Fan et al. [9], $MR(G, \mathbb{R}_{\leq 0})$ is still APX-Hard in the extreme case when $\Omega = \{0, \alpha\}$. ◀

► **Corollary 27.** *For any $G = (V, E, w)$ DMR returns the smallest solution for any ℓ_p norm for $p \in [1, \infty)$.*

Proof. The proof of Theorem 5 actually shows that there is a unique support for the sparsest solution, i.e., the set of all heavy edges. In fact any decrease only solution must contain all of these edges in its support. We can also see that DMR decreases these by the minimum amount so that the cycles are unbroken. Thus, this solution is in fact the smallest for any ℓ_p norm. ◀

A.2 Structural results

► **Proposition 7.** *The VERIFIER algorithm, given a weighted graph G and a potential support for a solution S , determines in $O(T_{\text{APSP}})$ time whether there exists a valid (increase only or general) solution on that support and if so finds one.*

Proof. Let $G = (V, E, w)$ be the original graph and let M be the maximum edge weight from the graph G . The algorithm defines a new graph $\hat{G} = (V, E, \hat{w})$, with the following weights

$$\hat{w}(e) = \begin{cases} w(e) & e \notin S \\ M & e \in S \end{cases}$$

For each $e = (v_1, v_2) \in E$, line 4 sets $w(e)$ to be the weight of the shortest path in \hat{G} from v_1 to v_2 . Thus, at the end of the algorithm $w(e)$ satisfies the shortest path metric of \hat{G} . As the algorithm outputs w if and only if only edge weights in S are modified (increased), it suffices to argue S is a regular cover (light cover) if and only if only edge weights in S are modified (increased).

Assume that S is a regular or light cover. We argue line 4 only updates the weights of the edges in S . Note that $G \setminus S$ has no broken cycles. Thus, for any $e = (v_1, v_2) \in G \setminus S$ we have that the shortest path from v_1 to v_2 must be e . Now consider any path P from v_1 to v_2 in \hat{G} . If $P \cap S = \emptyset$, then $w(P) \geq w(e)$. On the other hand if $P \cap S \neq \emptyset$, then let $\tilde{e} \in P \cap S$. Then, we have that

$$w(P) \geq w(\tilde{e}) = M \geq w(e)$$

Thus, in either case, $w(P) \geq w(e)$. Hence for all $e \in G \setminus S$ we do not change its weight.

If S is a light cover, we also need to argue that the weights only increased. Let $e = (v_1, v_2) \in S$. Let P be a path of smallest weight in \hat{G} . Suppose $P \cap S \neq \emptyset$, then, we have that $w(P) \geq M \geq w(e)$. Thus, in this case we could not have decreased the weight. Thus, assume that $P \cap S = \emptyset$. If we still have that $w(P) \geq w(e)$, then we could not have decreased the weight. Thus, let us further assume that $w(P) < w(e)$. In this case, P along with e form a broken cycle in G , with e as the heavy edge. But then since S is a light cover, we have that $P \cap S \neq \emptyset$. Thus, we have a contradiction and this case cannot occur. Thus, if S is a light cover, then we only increase the edge weights.

Now assume S is not a regular cover (light cover). Then there exists a broken cycle C such that none of its (light) edges are in S . Thus, there is a broken cycle C in \hat{G} . Let e be the heavy edge of C , then on line 4 the weight of e will be decreased, and thus our algorithm will return NULL. ◀

The next theorem shows that once we know the support, the set of all possible solutions on that support is a nice space.

► **Theorem 28.** *For any weighted graph G and support S we have that the set of solutions with support S is a closed convex subset of $\mathbb{R}^{n \times n}$. Additionally, if $G - S$ is a connected graph or we require an upper bound on the weight of each edge, then the set of solutions is compact.*

Proof. Let x_{ij} for $1 \leq i, j \leq n$ be our coordinates. Then the equations $x_{ij} = c_{ij}$ for (i, j) not in the support and $x_{ij} \leq x_{ik} + x_{kj}$ define a closed convex set. Thus, we see the first part. For the second part we just need to see that set is bounded to get compactness. If we have that $G - S$ is connected then for all $e \in S$ there is a path between end points of e in $G - S$. Thus, the weight of this path is an upper bound. On the other hand 0 is always a lower bound. Thus, we get compactness if $G - S$ is connected. ◀

B Approximation Algorithms

Here we give the missing proofs from our $O(\kappa \log n)$ -approximation algorithm.

► **Lemma 21.** *Let G be a positively weighted graph, where for all pairs of vertices u, v one has constant time access to the value $d(u, v)$. Then for any pair of vertices s, t , the value $\#sp(s, t)$ can be computed in $O(m + n)$ time.*

Proof. Let $V = \{v_1, v_2, v_3, \dots, v_n\}$, and let $N(v_i)$ denote the set of neighbors of v_i . Define $X_i = \{v_j \in N(v_i) \mid w(v_i, v_j) + d(v_j, t) = d(v_i, t)\}$, that is, X_i is the set of neighbors of v_i where there is a shortest path from t to v_i passing through that neighbor. Thus we have,

$$\#sp(v_i, t) = \sum_{v_j \in X_i} \#sp(v_j, t).$$

Note that any shortest path from v_i to t can only use vertices v_j which are closer to t than v_i . So consider a topological ordering of the vertices, where edges are conceptually oriented from smaller to larger $d(v_i, t)$ values. Thus if we compute the $\#sp(v_i, t)$ values in increasing order of the index i , then each $\#sp(v_i, t)$ value can be computed in time proportional to the degree of v_i , and so the overall running time is $O(m + n)$. ◀

► **Corollary 24.** *Given constant time access to $d(u, v)$ and $\#sp(u, v)$ for any vertices u and v , $N_h(e, \delta(G))$ can be computed in $O(1)$ time and $N_l(e, \delta(G))$ in $O(m)$ time.*

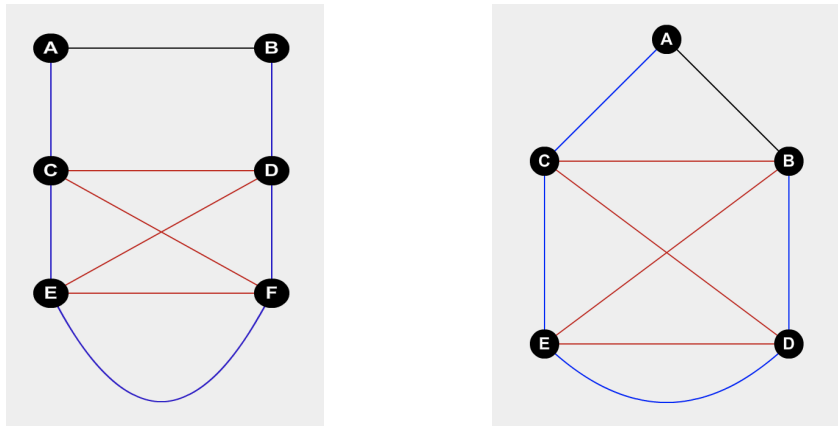
Proof. By Lemma 22, in constant time we can check whether $w(e) = d(s, t) + \delta(G)$, in which case set $N_h(e, \delta(G)) = \#\text{sp}(s, t)$, and otherwise set $N_h(e, \delta(G)) = 0$. By Lemma 23, we can compute $N_i(e, \delta(G))$ with a linear scan of the edges, where for each edge f in constant time we can compute whether $w(f) = d(a, s) + w(e) + d(t, b) + \delta(G)$ and if so add $\#\text{sp}(a, s) \cdot \#\text{sp}(t, b)$ to the sum over X , and if $w(f) = d(b, s) + w(e) + d(t, a) + \delta(G)$ add $\#\text{sp}(b, s) \cdot \#\text{sp}(t, a)$ to the sum over Y . ◀

C Improved Analysis for Complete Graphs

Here we consider the special case when $G = K_n$, improving parts of the analysis from [9, 10]. First, we consider the $O(OPT^{1/3})$ -approximation algorithm of [9], which works for both $\text{MR}(K_n, \mathbb{R})$ and $\text{MR}(K_n, \mathbb{R}_{\geq 0})$. The running time of this algorithm is $\Theta(n^6)$, since at some point it enumerates all cycles of length ≤ 6 . With a more careful analysis, we observe it suffices to consider cycles of length ≤ 5 , improving the running time to $\Theta(n^5)$. For $\text{MR}(K_n, \mathbb{R}_{\geq 0})$ we consider a simple, appealing algorithm with good empirical performance from [10], referred to as IOMR-FIXED. We prove that unfortunately it is an $\Omega(n)$ approximation.

C.1 5 Cycle Cover

Here we argue the running time of the $O(OPT^{1/3})$ -approximation algorithm of [9], which works for both $\text{MR}(K_n, \mathbb{R})$ and $\text{MR}(K_n, \mathbb{R}_{\geq 0})$, can be improved from $\Theta(n^6)$ to $\Theta(n^5)$. The algorithm presented in [9] has 3 major steps. The first two steps are used to approximate the support of the optimal solution and then the last step is actually used to find a solution given this support. We shall focus on the first 2 steps as these are where we make modifications.



■ **Figure 3** Left: Embedding from [9]. Right: Our modified embedding for a smaller cycle. Here the black edge is the heavy edge. The blue edges are the light edges and the red edges are the embedded 4 cycle. The curved blue edge indicates that there are more vertices along that path.

First Step: In the first step, [9] find a cover for all broken cycles of length $\leq m$. In particular, the authors use the case when $m = 6$. As described in [9], we can obtain an $m - 1$ approximation of the optimal cover for all broken cycles of length $\leq m$ in $O(n^m)$ time. Denote this cover by $S_{\leq m}$.

Second Step: For this step, we need to first define unit cycles. Given a broken cycle C with heavy edge h , let e be a chord of C . Then e divides C into 2 cycles, one that contains h , denoted $\text{heavy}(C, e)$ and one that does not contain h denoted $\text{light}(C, e)$. We say this cycle is a unit cycle if for all chords e , e is not the heavy edge of $\text{light}(C, e)$.

From the definition of a unit cycle, a light cover of all unit cycles light covers all broken cycles. Hence, step 2 of the algorithm from [9] light covers all unit cycles not covered by $S_{\leq 6}$ as follows. Let C be such a unit cycle. Now we know that C has at least 7 edges. Consider the red C_4 shown in Figure 3. We know that for each $e \in C_4$, we have that $\text{heavy}(C, e)$ is a broken cycle with at most 6 edges. Hence, we must have at least 1 edge in $S_{\leq 6}$. But since C has no light edges in $S_{\leq 6}$, we must have $e \in S_{\leq 6}$. Thus, we know all edges in C_4 are edges in $S_{\leq 6}$. Moreover, observe that either chord of C_4 is a light edge of C . Thus it suffices to compute a cover with least one chord of every four cycle from the edges in $S_{\leq 6}$, a step which the authors in [9] denote $\text{chord4}(S_{\leq 6})$.

In Figure 3, we observe that the same 4 cycle can be embedded in a 6 cycle instead of a 7 cycle. Thus, our modified algorithm is shown in Algorithm 6.

■ **Algorithm 6** 5-Cycle Cover.

```

1: function 5 CYCLE COVER( $G = (V, E, w)$ )
2:   Compute a regular cover of  $S_{\leq 5}$  of all broken cycles with  $\leq 5$  edges
3:   Compute a cover  $S_c = \text{chord4}(S_{\leq 5})$ 
4:   return VERIFIER( $G, S_c \cup S_{\leq 5}$ )

```

C.2 IOMR-fixed

We will now show that IOMR-FIXED is an $\Omega(n)$ approximation algorithm. The algorithm presented in Gilbert and Jain [10] is as follows:

■ **Algorithm 7** IOMR Fixed.

Require: $D \in \text{Sym}_n(\mathbb{R}_{\geq 0})$

```

1: function IOMR-FIXED( $D$ )
2:    $\hat{D} = D$ 
3:   for  $k \leftarrow 1$  to  $n$  do
4:     for  $i \leftarrow 1$  to  $n$  do
5:        $\hat{D}_{ik} = \max(\hat{D}_{ik}, \max_{j < i}(\hat{D}_{ij} - \hat{D}_{jk}))$ 
6:   return  $\hat{D} - D$ 

```

► **Lemma 29.** *For every n , there exists a weighted graph G such that IOMR-FIXED repairs $\binom{n-1}{2}$ edge weights while an optimal solutions repairs at most $(n-2)$ edge weights.*

Proof. Consider a matrix D where

$$D_{ij} = \begin{cases} 0 & \text{if } i \neq 1, j \neq 1 \\ 2^i & \text{if } j = 1, i > 1 \\ 2^j & \text{if } i = 1, j > 1 \end{cases}$$

This matrix D will be the weight matrix for the input graph K_n .

First, we claim that all entries of the form D_{s1} will never be updated as entries will only be updated the first time they are seen. Thus

$$D_{s1} = \max(D_{s1}, \max_{t < s}(D_{s1} - D_{1t})) = \max(2^s, \max_{t < s}(2^s - 2^t)) = 2^s$$

25:22 Generalized Metric Repair on Graphs

Now we just have to verify that the rest of the non-diagonal entries are updated. Let us look at the first time an entry D_{rs} is updated. (Here $r < s$.) Then we have that

$$\begin{aligned}\hat{D}_{rs} &= \max(D_{rs}, \max_{t < s}(D_{st} - D_{tr})) = \max_{t < s}(D_{st} - D_{tr}) && \text{[Since } D_{rs} = 0\text{]} \\ &\geq D_{s1} - D_{1r} = 2^s - s^r > D_{rs}.\end{aligned}$$

Thus all other non-diagonal entries will be updated the first time seen. Thus, for the solution $W = \hat{D} - D$ that IOMR-FIXED returns, we see that $W_{ij} > 0$ for exactly all $1 < i, j \leq n$ and $i \neq j$. Thus, we repaired $\binom{n-1}{2}$ edge weights.


Finally, a sparser increase only solution W can be obtained as follows. For all $s > 1$ we set

$$W_{1s} = W_{s1} = 2^n - D_{s1}$$

and all other entries of W are 0. This then gives us the desired result. ◀

► **Corollary 30.** *IOMR-FIXED is an $\Omega(n)$ approximation algorithm.*

Maximum Edge-Colorable Subgraph and Strong Triadic Closure Parameterized by Distance to Low-Degree Graphs

Niels Grüttemeier 

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
niegru@informatik.uni-marburg.de

Christian Komusiewicz 

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
komusiewicz@informatik.uni-marburg.de

Nils Morawietz

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany
morawietz@informatik.uni-marburg.de

Abstract

Given an undirected graph G and integers c and k , the MAXIMUM EDGE-COLORABLE SUBGRAPH problem asks whether we can delete at most k edges in G to obtain a graph that has a proper edge coloring with at most c colors. We show that MAXIMUM EDGE-COLORABLE SUBGRAPH admits, for every fixed c , a linear-size problem kernel when parameterized by the edge deletion distance of G to a graph with maximum degree $c - 1$. This parameterization measures the distance to instances that, due to Vizing's famous theorem, are trivial yes-instances. For $c \leq 4$, we also provide a linear-size kernel for the same parameterization for MULTI STRONG TRIADIC CLOSURE, a related edge coloring problem with applications in social network analysis. We provide further results for MAXIMUM EDGE-COLORABLE SUBGRAPH parameterized by the vertex deletion distance to graphs where every component has order at most c and for the list-colored versions of both problems.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Graph coloring, social networks, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.26

Related Version A full version of the paper is available at <https://arxiv.org/abs/2002.08659>.

1 Introduction

Edge coloring and its many variants form a fundamental problem family in algorithmic graph theory [3, 12, 13, 14]. In the classic EDGE COLORING problem, the input is a graph G and an integer c and the task is to decide whether G has a *proper edge coloring*, that is, an assignment of colors to the edges of a graph such that no pair of incident edges receives the same color, with at most c colors. The number of necessary colors for a proper edge coloring of a graph G is closely related to the degree of G : Vizing's famous theorem states that any graph G with maximum degree Δ can be edge-colored with $\Delta + 1$ colors [27], an early example of an additive approximation algorithm. Later it was shown that EDGE COLORING is NP-hard for $c = 3$ [13], and in light of Vizing's result it is clear that the hard instances for $c = 3$ are exactly the subcubic graphs. Not surprisingly, the NP-hardness extends to every fixed $c \geq 3$ [21].

In the more general MAXIMUM EDGE-COLORABLE SUBGRAPH (ECS) problem, we are given an additional integer k and want to decide whether we can delete at most k edges in the input graph G so that the resulting graph has a proper edge coloring with c colors.



© Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 26; pp. 26:1–26:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ECS is NP-hard for $c = 2$ [6] and it has received a considerable amount of interest for small constant values of c such as $c = 2$ [6, 18], $c = 3$ [18, 19, 23], and $c \leq 7$ [15]. Feige et al. [6] mention that ECS has applications in call admittance in telecommunication networks. Given the large amount of algorithmic literature on this problem, it is surprising that there is, to the best of our knowledge, no work on fixed-parameter algorithms for ECS. This lack of interest may be rooted in the NP-hardness of EDGE COLORING for every fixed $c \geq 3$, which implies that ECS is not fixed-parameter tractable with respect to $k + c$ unless $P=NP$.

Instead of the parameter k , we consider the parameter ξ_{c-1} which we define as the minimum number of edges that need to be deleted in the input graph to obtain a graph with maximum degree $c - 1$. This is a distance-from-triviality parameterization [11]: Due to Vizing's Theorem, the answer is always yes if the input graph has maximum degree $c - 1$. We parameterize by the edge-deletion distance to this trivial case. Observe that the number of vertices with degree at least c is at most $2\xi_{c-1}$. If we consider EDGE COLORING instead of ECS, the instances with maximum degree larger than c are trivial no-instances. Thus, in non-trivial instances, the parameter ξ_{c-1} is essentially the same as the number of vertices that have degree c . This is, arguably, one of the most natural parameterizations for EDGE COLORING. We achieve a kernel that has linear size for every fixed c .

► **Theorem 1.1.** *ECS admits a problem kernel with at most $4\xi_{c-1} \cdot c$ vertices and $\mathcal{O}(\xi_{c-1} \cdot c^2)$ edges that can be computed in $\mathcal{O}(n + m)$ time.*

Herein, n denotes the number of vertices of the input graph G and m denotes the number of edges. This kernel is obtained by making the following observation about the proof of Vizing's Theorem: When proving that an edge can be safely colored with one of c colors, we only need to consider the closed neighborhood of one endpoint of this edge. This allows us to show that all vertices which have degree at most $c - 1$ and only neighbors of degree at most $c - 1$ can be safely removed.

Next, we consider ECS parameterized by the size λ_c of a smallest vertex set D such that deleting D from G results in a graph where each connected component has at most c vertices. The parameter λ_c presents a different distance-from-triviality parameterization, since a graph with connected components of order at most c can trivially be colored with c edge colors. Moreover, observe that λ_c is never larger than the vertex cover number which is a popular structural parameter. Again, we obtain a linear-vertex kernel for λ_c when c is fixed.

► **Theorem 1.2.** *ECS admits a problem kernel with $\mathcal{O}(c^3 \cdot \lambda_c)$ vertices.*

We then consider MULTI STRONG TRIADIC CLOSURE (MULTI-STC) a closely related edge coloring problem with applications in social network analysis [25]. In MULTI-STC, we are given a graph G and two integers k and c and aim to find a coloring of the edges with one *weak* and at most c *strong colors* such that every pair of incident edges that forms an induced path on three vertices does not receive the same strong color and the number of weak edges is at most k . The idea behind this problem is to uncover the different strong relation types in social networks by using the following assumption: if one person has for example two colleagues, then these two people know each other and should also be connected in the social network. In other words, if a vertex has two neighbors that are not adjacent to each other, then this is evidence that either the strong interaction types with these two neighbors are different or one of the interaction types is merely weak.

Combinatorically, there are two crucial differences to ECS: First, two incident edges may receive the same strong color if the subgraph induced by the endpoints is a triangle. Second, instead of deleting edges to obtain a graph that admits such a coloring, we may label edges

■ **Table 1** A summary of our results for the two problems. Herein, ξ_{c-1} denotes the edge-deletion distance to graphs with maximum degree at most $c - 1$, and λ_c denotes the vertex-deletion distance to graphs where every connected component has order at most c .

Parameter	ECS	MULTI-STC
(ξ_{c-1}, c)	$\mathcal{O}(\xi_{c-1}c)$ -vertex kernel (Thm. 1.1)	$\mathcal{O}(\xi_{c-1})$ -edge kernel (Thm. 4.14), if $c \leq 4$
(λ_c, c)	$\mathcal{O}(c^3 \cdot \lambda_c)$ -vertex kernel (Thm. 1.2)	No poly Kernel, even for $c = 1$ [10]

as weak. In ECS this does not make a difference; in MULTI-STC, however, deleting an edge may destroy triangles which would add an additional constraint on the coloring of the two remaining triangle edges.

In contrast to ECS, MULTI-STC is NP-hard already for $c = 1$ [25]. This special case is known as STRONG TRIADIC CLOSURE (STC). Not surprisingly, MULTI-STC is NP-hard for all fixed $c \geq 2$ [1]. Moreover, for $c \geq 3$ MULTI-STC is NP-hard even if $k = 0$, that is, even if every edge has to be colored with a strong color. STC and Multi-STC have received a considerable amount of interest recently [25, 9, 10, 1, 16, 17].

Since the edge coloring for MULTI-STC is a relaxed version of a proper edge coloring, we may observe that Vizing's Theorem implies the following: If the input graph G has degree at most $c - 1$, then the instance is a yes-instance even for $k = 0$. Hence, it is very natural to apply the parameterization by ξ_{c-1} also for MULTI-STC. We succeed to transfer the kernelization result from ECS to MULTI-STC for $c \leq 4$. In fact, our result for $c = 3$ and $c = 4$ can be extended to the following more general result.

► **Theorem 1.3.** *MULTI-STC admits a problem kernel with $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c)$ vertices and $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c^2)$ edges, when limited to instances with $c \geq 3$. The kernel can be computed in $\mathcal{O}(n + m)$ time.*

For $c = 5$, this gives a linear-size kernel for the parameter ξ_3 , for $c = 6$, a linear-size kernel for the parameter ξ_4 and so on. Our techniques to prove Theorem 1.3 are very loosely inspired by the proof of Vizing's Theorem but in the context of MULTI-STC several obstacles need to be overcome. As a result, the proof differs quite substantially from the one for ECS. Moreover, in contrast to ECS, MULTI-STC does not admit a polynomial kernel when parameterized by the vertex cover number [10] which excludes almost all popular structural parameters.

We then show how far our kernelization for ξ_t can be lifted to generalizations of ECS and MULTI-STC where each edge may choose its color only from a specified list of colors, denoted as EDGE LIST ECS (EL-ECS) and EDGE LIST MULTI-STC (EL-MULTI-STC). We show that for ξ_2 we obtain a linear kernel for every fixed c .

► **Theorem 1.4.** *For all $c \in \mathbb{N}$, EL-ECS and EL-MULTI-STC admit an $11\xi_2$ -edge and $10\xi_2$ -vertex kernel for EL-ECS that can be computed in $\mathcal{O}(n^2)$ time.*

For $c = 3$, this extends Theorem 1.1 to the list colored version of ECS. For $c > 3$ parameterization by ξ_2 may seem a bit uninteresting compared to the results for ECS and MULTI-STC. However, Theorem 1.4 is unlikely to be improved by considering ξ_t for $t > 2$.

► **Proposition 1.5.** *EL-ECS and EL-MULTI-STC are NP-hard for all $c \geq 3$ on triangle-free cubic graphs even if $\xi_3 = k = 0$.*

A summary of our results is shown in Table 1. Due to space constraints, the proofs of Theorem 1.4 and Proposition 1.5 and further propositions and lemmas needed to show Theorems 1.1–1.3 are deferred to a full version.

2 Preliminaries

Notation. We consider simple undirected graphs $G = (V, E)$. For a vertex $v \in V$, we denote by $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ the *open neighborhood of v* and by $N_G[v] := N_G(v) \cup \{v\}$ the *closed neighborhood of v* . For a given set $V' \subseteq V$, we define $N_G(V') := \bigcup_{v \in V'} N_G(v)$ as the *neighborhood of V'* . Moreover, let $\deg_G(v) := |N(v)|$ be the *degree* of a vertex v in G and $\Delta_G := \max_{v \in V} \deg_G(v)$ denote the *maximum degree* of G . For any two vertex sets $V_1, V_2 \subseteq V$, we let $E_G(V_1, V_2) := \{\{v_1, v_2\} \in E \mid v_1 \in V_1, v_2 \in V_2\}$ denote the set of edges between V_1 and V_2 . For any vertex set $V' \subseteq V$, we let $E_G(V') := E_G(V', V')$ denote the set of edges between the vertices of V' . The *subgraph induced by a vertex set S* is denoted by $G[S] := (S, E_G(S))$. For a given vertex set $V' \subseteq V$, we let $G - V' := G[V \setminus V']$ denote the graph that we obtain after deleting the vertices of V' from G . We may omit the subscript G if the graph is clear from the context.

A *finite sequence* $A = (a_0, a_1, \dots, a_{r-1})$ of length $r \in \mathbb{N}_0$ is an r -tuple of specific elements a_i (for example vertices or numbers). For given $j \in \{0, \dots, r-1\}$, we refer to the j th element of a finite sequence A as $A(j)$. A *path* $P = (v_0, \dots, v_{r-1})$ is a finite sequence of vertices $v_0, \dots, v_{r-1} \in V$, where $\{v_i, v_{i+1}\} \in E$ for all $i \in \{0, \dots, r-2\}$. A path P is called *vertex-simple*, if no vertex appears twice on P . A path is called *edge-simple*, if there are no distinct $i, j \in \{0, \dots, r-2\}$ such that $\{P(i), P(i+1)\} = \{P(j), P(j+1)\}$. For a given path $P = (P(0), \dots, P(r-1))$ we define the sets $V(P) := \{P(j) \mid j \in \{0, \dots, r-1\}\}$ and $E(P) := \{\{P(j), P(j+1)\} \mid j \in \{0, \dots, r-2\}\}$ as the set of vertices or edges on P .

For the relevant definitions of parameterized complexity such as parameterized reduction and problem kernelization refer to the standard monographs [4, 5, 7, 22].

Problem Definitions. We now formally define the two main problems considered in this work, ECS and MULTI-STC, as well as their extensions to input graphs with edge lists.

► **Definition 2.1.** A c -colored labeling $L = (S_L^1, \dots, S_L^c, W_L)$ of an undirected graph $G = (V, E)$ is a partition of the edge set E into $c+1$ color classes. The edges in S_L^i , $i \in \{1, \dots, c\}$, are strong and the edges in W_L are weak.

1. A c -colored labeling L is a proper labeling if there exists no pair of edges $e_1, e_2 \in S_L^i$ for some strong color i , such that $e_1 \cap e_2 \neq \emptyset$.
2. A c -colored labeling L is an STC-labeling if there exists no pair of edges $\{u, v\} \in S_L^i$ and $\{v, w\} \in S_L^i$ such that $\{u, w\} \notin E$.

We consider the following two problems.

EDGE-COLORABLE SUBGRAPH (ECS)

Input: An undirected graph $G = (V, E)$ and integers $c \in \mathbb{N}$ and $k \in \mathbb{N}$.

Question: Is there a c -colored proper labeling L with $|W_L| \leq k$?

MULTI STRONG TRIADIC CLOSURE (MULTI-STC)

Input: An undirected graph $G = (V, E)$ and integers $c \in \mathbb{N}$ and $k \in \mathbb{N}$.

Question: Is there a c -colored STC-labeling L with $|W_L| \leq k$?

If c is clear from the context, we may call a c -colored labeling just *labeling*. Two labelings $L = (S_L^1, \dots, S_L^c, W_L)$, and $L' = (S_{L'}^1, \dots, S_{L'}^c, W_{L'})$ for the same graph $G = (V, E)$ are called *partially equal* on a set $E' \subseteq E$ if and only if for all $e \in E'$ and $i \in \{1, \dots, c\}$ it holds that $e \in S_L^i \Leftrightarrow e \in S_{L'}^i$. If two labelings L and L' are partially equal on E' we write $L|_{E'} = L'|_{E'}$. For given path $P = (P(0), \dots, P(r-1))$ and labeling $L = (S_L^1, \dots, S_L^c, W_L)$, we define the *color sequence* Q_L^P of P under L as a finite sequence $Q_L^P = (q_0, q_1, \dots, q_{r-2})$ of elements

in $\{0, \dots, c\}$, such that $\{P(i), P(i+1)\} \in S_L^{q_i}$ if $q_i \geq 1$ and $\{P(i), P(i+1)\} \in W_L$ if $q_i = 0$. Throughout this work we call a c -colored STC-labeling L (or proper labeling, respectively) *optimal* if the number of weak edges $|W_L|$ is minimal.

Edge-Deletion Distance to Low-Degree Graphs and Component Order Connectivity. We consider parameters related to the edge deletion-distance ξ_t to low-degree graphs and the vertex-deletion distance λ_t to graphs with small connected components.

First, we define the parameter ξ_t . For a given graph $G = (V, E)$ and a constant $t \in \mathbb{N}$, we call $D_t \subseteq E$ an *edge-deletion set of G and t* if the graph $(V, E \setminus D_t)$ has maximum degree t . We define the parameter ξ_t as the size of the minimum edge-deletion set of G and t . Note that an edge-deletion set of G and t of size ξ_t can be computed in polynomial time [8]. More importantly for our applications, we can compute a 2-approximation D'_t for an edge-deletion set of size ξ_t in linear time as follows: Add for each vertex v of degree at least $t+1$ an arbitrary set of $\deg(v) - t$ incident edges to D'_t . Then $|D'_t| \leq \sum_{v \in V} \max(\deg(v) - t, 0)$. This implies that D'_t is a 2-approximation since $\sum_{v \in V} \max(\deg(v) - t, 0) \leq 2\xi_t$ as every edge deletion decreases the degree of at most two vertices. A given edge-deletion set D_t induces the following important partition of the vertex set V of a graph.

► **Definition 2.2.** Let $t \in \mathbb{N}$, let $G = (V, E)$ be a graph, and let $D_t \subseteq E$ be an edge-deletion set of G and t . We call $\mathcal{C} = \mathcal{C}(D_t) := \{v \in V \mid \exists e \in D_t : v \in e\}$ the set of core vertices and $\mathcal{P} = \mathcal{P}(D_t) := V \setminus \mathcal{C}$ the set of periphery vertices of G .

Note that for arbitrary $t \in \mathbb{N}$ and G we have $|\mathcal{C}| \leq 2|D_t|$ and for every $v \in \mathcal{P}$ it holds that $\deg_G(v) \leq t$. Moreover, every vertex in \mathcal{C} is incident with at most t edges in $E \setminus D_t$. In context of ECS and MULTI-STC, for a given instance (G, c, k) we consider some fixed edge deletion set D_t of the input graph G and some integer t which depends on the value of c .

Second, we define the parameter λ_t . For a given graph $G = (V, E)$ and a constant $t \in \mathbb{N}$, we call $D \subseteq V$ an *order- t component cover* if every connected component in $G - D$ contains at most t vertices. Then, we define the *component order connectivity* λ_t to be the size of a minimum order- t component cover. In context of ECS we study λ_c , for the amount of colors c . A $(c+1)$ -approximation of the minimal order- c -component cover can be computed in polynomial time [20].

Note that the parameters are incomparable in the following sense: In a path P_n the parameter λ_c can be arbitrarily large when n increases while $\xi_{c-1} = 0$ for all $c \geq 3$. In a star S_n the parameter ξ_{c-1} can be arbitrary large when n increases while $\lambda_c = 1$.

3 Problem Kernelizations for Edge-Colorable Subgraph

In this section, we provide problem kernels for ECS parameterized by the edge deletion distance ξ_{c-1} to graphs with maximum degree $c-1$, and the size λ_c of a minimum order- c component cover. We first show that ECS admits a kernel with $\mathcal{O}(\xi_{c-1} \cdot c)$ vertices and $\mathcal{O}(\xi_{c-1} \cdot c^2)$ edges that can be computed in $\mathcal{O}(n+m)$ time. Afterwards, we consider λ_c and show that ECS admits a problem kernel with $\mathcal{O}(c^3 \lambda_c)$ vertices, which is a linear vertex kernel for every fixed value of c . Note that if $c = 1$ we can solve ECS by computing a maximal matching in polynomial time. Hence, we assume $c \geq 2$ for the rest of this section. In this case the problem is NP-hard [6].

3.1 Edge Deletion-Distance to Low-Degree Graphs

The kernelization presented inhere is based on Vizing's Theorem [27]. Note that Vizing's Theorem implies, that an ECS instance (G, c, k) is always a yes-instance if $\xi_{c-1} = 0$. Our kernelization relies on the following lemma. This lemma is a reformulation of a known fact about edge colorings [26, Theorem 2.3] which, in turn, is based on the so-called *Vizing Fan Equation* [26, Theorem 2.1].

► **Lemma 3.1.** *Let $G = (V, E)$ be a graph and let $e := \{u, v\} \in E$. Moreover, let $c := \Delta_G$ and let L be a proper c -colored labeling for the graph $(V, E \setminus \{e\})$ such that $W_L = \emptyset$. If for all $Z \subseteq N_G(u)$ with $|Z| \geq 2$ and $v \in Z$ it holds that $\sum_{z \in Z} (\deg_G(z) + 1 - c) < 2$, then there exists a proper c -colored labeling L' for G such that $W_{L'} = \emptyset$.*

We now use Lemma 3.1 as a plug-in for ECS to prove the next lemma which is the main tool that we need for our kernelization. In the proof, we exploit the fact that, given any proper labeling L for a graph $G = (V, E)$, the labeling $(S_L^1, \dots, S_L^c, \emptyset)$ is a proper labeling for the graph $(V, E \setminus W_L)$.

► **Lemma 3.2.** *Let $L := (S_L^1, S_L^2, \dots, S_L^c, W_L)$ be a proper labeling with $|W_L| = k$ for a graph $G := (V, E)$. Moreover, let $e := \{u, v\} \subseteq V$ such that $e \notin E$ and let $G' := (V, E \cup \{e\})$ be obtained from G by adding e . If for one endpoint $u \in e$ it holds that every vertex $w \in N_{G'}[u]$ has degree at most $c - 1$ in G' , then there exists a proper labeling L' for G' with $|W_{L'}| = k$.*

Proof. Consider the auxiliary graph $G_{\text{aux}} := (V, E \setminus W_L)$. Since L is a proper labeling for G , we conclude that $L_{\text{aux}} := (S_L^1, \dots, S_L^c, \emptyset)$ is a proper labeling for G_{aux} . Let $H_{\text{aux}} := (V, E_H)$ where $E_H := (E \setminus W_L) \cup \{e\}$. In order to prove the lemma, we show that there exists a proper labeling L'_{aux} for H_{aux} such that $W_{L'_{\text{aux}}} = \emptyset$.

To this end, we first consider the maximum degree of H_{aux} . Observe that $\deg_{H_{\text{aux}}}(w) \leq \deg_{G'}(w)$ for all $w \in V$. Hence, the property that $\deg_{G'}(w) \leq c - 1$ for all $w \in N_{G'}[u]$ implies $\Delta_{H_{\text{aux}}} = \max(\Delta_{G_{\text{aux}}}, c - 1)$. Since L_{aux} is a proper c -colored labeling for G_{aux} we know that $\Delta_{G_{\text{aux}}} \leq c$ and therefore we have $\Delta_{H_{\text{aux}}} \leq c$. So, to find a proper c -colored labeling without weak edges for H_{aux} it suffices to consider the following cases.

Case 1: $\Delta_{H_{\text{aux}}} \leq c - 1$. Then, there exists a proper labeling L'_{aux} for H_{aux} such that $W_{L'_{\text{aux}}} = \emptyset$ due to Vizing's Theorem.

Case 2: $\Delta_{H_{\text{aux}}} = c$. In this case we can apply Lemma 3.1: Observe that $(V, E_H \setminus \{e\}) = G_{\text{aux}}$ and L_{aux} is a proper labeling for G_{aux} such that $W_{L_{\text{aux}}} = \emptyset$. Consider an arbitrary $Z \subseteq N_{H_{\text{aux}}}(u)$ with $|Z| \geq 2$ and $v \in Z$. Note that $Z \subseteq N_{H_{\text{aux}}}(u)$ implies $\deg_{H_{\text{aux}}}(z) \leq c - 1$ for all $z \in Z$. It follows that $\sum_{z \in Z} (\deg_{H_{\text{aux}}}(z) + 1 - c) < 2$. Since Z was arbitrary, Lemma 3.1 implies that there exists a proper labeling L'_{aux} for H_{aux} such that $W_{L'_{\text{aux}}} = \emptyset$.

We now define $L' := (S_{L'_{\text{aux}}}^1, S_{L'_{\text{aux}}}^2, \dots, S_{L'_{\text{aux}}}^c, W_L)$. Note that the edge set $E \cup \{e\}$ of G' can be partitioned into W_L and the edges of G'_{aux} . Together with the fact that L'_{aux} is a labeling for G'_{aux} it follows that every edge of G' belongs to exactly one color class of L' . Moreover, it obviously holds that $|W_{L'}| = |W_L| = k$. Since there is no vertex with two incident edges in the same strong color class $S_{L'_{\text{aux}}}^i$, the labeling L' is a proper labeling for G' . ◀

We now introduce the kernelization rule. Recall that \mathcal{C} is the set of vertices that are incident with at least one of the ξ_{c-1} edge-deletions that transform G into a graph with maximum degree $c - 1$. We make use of the fact that edges that have at least one endpoint u that is not in $\mathcal{C} \cup N(\mathcal{C})$ satisfy $\deg(w) \leq c - 1$ for all $w \in N[u]$. Lemma 3.2 guarantees that these edges are not important to solve an instance of ECS.

► **Rule 3.1.** Remove all vertices in $V \setminus (\mathcal{C} \cup N(\mathcal{C}))$ from G .

► **Proposition 3.3.** Rule 3.1 is safe.

Proof. Let $(G' = (V', E'), c, k)$ be the reduced instance after applying Rule 3.1. We prove the safeness of Rule 3.1 by showing that there is a proper labeling with at most k weak edges for G if and only if there is a proper labeling with k weak edges for G' .

(\Rightarrow) Let $L = (S_L^1, S_L^2, \dots, S_L^c, W_L)$ be a proper labeling with $|W_L| \leq k$ for G . Then, obviously $L' := (S_L^1 \cap E', S_L^2 \cap E', \dots, S_L^c \cap E', W_L \cap E')$ is a proper labeling for G' with $|W_{L'}| \leq |W_L| \leq k$.

(\Leftarrow) Conversely, let $L' = (S_{L'}^1, S_{L'}^2, \dots, S_{L'}^c, W_{L'})$ be a proper labeling with $|W_{L'}| \leq k$ for G' . Let $E \setminus E' = \{e_1, e_2, \dots, e_p\}$. We define $p + 1$ graphs $G_0, G_1, G_2, \dots, G_p$ by $G_0 := (V, E')$, and $G_i := (V, E' \cup \{e_1, \dots, e_i\})$ for $i \in \{1, \dots, p\}$. Note that $G_p = G$, $\deg_{G_i}(v) \leq \deg_G(v)$, and $N_{G_i}(v) \subseteq N_G(v)$ for every $i \in \{0, 1, \dots, p\}$, and $v \in V$. We prove by induction over i that all G_i have a proper labeling with at most k weak edges.

Base Case: $i = 0$. Then, since G_0 and G' have the exact same edges, L' is a proper labeling for G_0 with at most k weak edges.

Inductive Step: $0 < i \leq p$. Then, by the inductive hypothesis, there exists a proper labeling L_{i-1} for $G_{i-1} = (V, E' \cup \{e_1, \dots, e_{i-1}\})$ with at most k weak edges. From $E' = E(\mathcal{C} \cup N(\mathcal{C}))$ we conclude $e_i \in E \setminus E(\mathcal{C} \cup N(\mathcal{C})) = E(\mathcal{P}) \setminus E(N(\mathcal{C}))$. Hence, for at least one of the endpoints u of e_i it holds that $N_G[u] \subseteq \mathcal{P}$. Therefore $\deg_G(w) \leq c - 1$ for all $w \in N_G[u]$. Together with the facts that $\deg_{G_i}(w) \leq \deg_G(w)$ and $N_{G_i}(w) \subseteq N_G(w)$ we conclude $\deg_{G_i}(w) \leq c - 1$ for all $w \in N_{G_i}[u]$. Then, by Lemma 3.2, there exists a proper labeling L_i for G_i such that $|W_{L_i}| = |W_{L_{i-1}}| \leq k$. ◀

► **Theorem 1.1.** ECS admits a problem kernel with at most $4\xi_{c-1} \cdot c$ vertices and $\mathcal{O}(\xi_{c-1} \cdot c^2)$ edges that can be computed in $\mathcal{O}(n + m)$ time.

Proof. Let (G, c, k) be an instance of ECS. We apply Rule 3.1 on (G, c, k) as follows: First, we compute a 2-approximation D'_{c-1} of the smallest possible edge-deletion set D_{c-1} in $\mathcal{O}(n + m)$ time as described in Section 2. Let $\mathcal{C} := \mathcal{C}(D'_{c-1})$ and note that $|D'_{c-1}| \leq 2\xi_{c-1}$. We then remove all vertices in $V \setminus (\mathcal{C} \cup N_G(\mathcal{C}))$ from G which can also be done in $\mathcal{O}(n + m)$ time. Hence, applying Rule 3.1 can be done in $\mathcal{O}(n + m)$ time.

We next show that after this application of Rule 3.1 the graph consists of at most $4\xi_{c-1} \cdot c$ vertices and $\mathcal{O}(\xi_{c-1} \cdot c^2)$ edges. Since D'_{c-1} is a 2-approximation of the smallest possible edge-deletion set we have $|\mathcal{C}| \leq 4\xi_{c-1}$. Since every vertex in \mathcal{C} has at most $c - 1$ neighbors in $V \setminus \mathcal{C}$, we conclude $|\mathcal{C} \cup N(\mathcal{C})| \leq 4\xi_{c-1} \cdot c$. In $E(\mathcal{C} \cup N(\mathcal{C}))$ there are obviously the at most $4\xi_{c-1}$ edges of D'_{c-1} . Moreover, each of the at most $4\xi_{c-1} \cdot c$ vertices might have up to $c - 1$ incident edges. Hence, after applying Rule 3.1, the reduced instance has $\mathcal{O}(\xi_{c-1} \cdot c^2)$ edges. ◀

If we consider EDGE COLORING instead of ECS, we can immediately reject if one vertex has degree more than c . Then, since there are at most $|\mathcal{C}| \leq 2\xi_{c-1}$ vertices that have a degree of at least c , Theorem 1.1 implies the following.

► **Corollary 3.4.** Let h_c be the number of vertices with degree c . EDGE COLORING admits a problem kernel with $\mathcal{O}(h_c \cdot c)$ vertices and $\mathcal{O}(h_c \cdot c^2)$ edges that can be computed in $\mathcal{O}(n + m)$ time.

3.2 Component Order Connectivity

In this section we present a problem kernel for ECS parameterized by the number of strong colors c and the component order connectivity λ_c . We prove that ECS admits a problem kernel with $\mathcal{O}(c^3 \cdot \lambda_c)$ vertices, which is a linear vertex kernel for every fixed value of c . Our kernelization is based on the Expansion Lemma [24], a generalization of the Crown Rule [2]. We use the formulation given by Cygan et al. [4].

► **Lemma 3.5** (Expansion Lemma). *Let q be a positive integer and G be a bipartite graph with partite sets A and B such that $|B| \geq q|A|$ and there are no isolated vertices in B . Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ with $N(Y) \subseteq X$. Moreover, there exist edges $M \subseteq E(X, Y)$ such that*

- a) *every vertex of X is incident with exactly q edges of M , and*
- b) *$q \cdot |X|$ vertices in Y are endpoints of edges in M .*

The sets X and Y can be found in polynomial time.

To apply Lemma 3.5 on an instance of ECS, we need the following definition for technical reasons.

► **Definition 3.6.** *For a given graph $G = (V, E)$, let D be an order- c component cover. We say that D is saturated if for every $v \in D$ it holds that $E_G(\{v\}, V \setminus D) \neq \emptyset$.*

Note that every order- c component cover D' can be transformed into a saturated order- c component cover by removing any vertex $v \in D'$ with $N(v) \subseteq D'$ from D' while such a vertex exists. Let $(G = (V, E), c, k)$ be an instance of ECS and let $D \subseteq V$ be a saturated order- c component cover. Furthermore, let $I := V \setminus D$ be the remaining set of vertices.

► **Rule 3.2.** *If there exists a set $J \subseteq I$ such that J is a connected component in G , remove all vertices in J from G .*

Rule 3.2 is safe since $|J| \leq c$ and therefore the graph $G[J]$ has maximum degree $c - 1$ and can be labeled by Vizing's Theorem with c colors. For the rest of this section we assume that (G, c, k) is reduced regarding Rule 3.2. The following proposition is a direct consequence of Lemma 3.5.

► **Proposition 3.7.** *Let $(G = (V, E), c, k)$ be an instance of ECS that is reduced regarding Rule 3.2, let D be a saturated order- c component cover of G , and let $I := V \setminus D$. If $|I| \geq c^2 \cdot |D|$, then there exist nonempty sets $X \subseteq D$ and $Y \subseteq I$ with $N(Y) \subseteq X \cup Y$. Moreover, there exists a set $M \subseteq E(X, Y)$ such that*

- a) *every vertex of X is incident with exactly c edges of M , and*
- b) *$c \cdot |X|$ vertices in Y are endpoints of edges in M and every connected component in $G[Y]$ contains at most one such vertex.*

The sets X and Y can be computed in polynomial time.

Proof. We prove the proposition by applying Lemma 3.5. To this end we define an equivalence relation \sim on the vertices of I : Two vertices $v, u \in I$ are equivalent, denoted $u \sim v$ if and only if u and v belong to the same connected component in $G[I]$. Obviously, \sim is an equivalence relation. For a given vertex $u \in I$, let $[u] := \{v \in I \mid v \sim u\}$ denote the equivalence class of u . Note that $|[u]| \leq c$ since D is an order- c component cover.

We next define the auxiliary graph G_{aux} , on which we will apply Lemma 3.5. Intuitively, we obtain G_{aux} from G by deleting all edges in $E_G(D)$ and merging the at most c vertices in

every equivalence class in I . Formally $G_{\text{aux}} := (D \cup I^*, E_{\text{aux}})$, with $I^* := \{[u] \mid u \in I\}$ and

$$E_{\text{aux}} := \{\{[u], v\} \mid [u] \in I^*, v \in \bigcup_{w \in [u]} (N_G(w) \setminus I)\}.$$

Note that G_{aux} can be computed from G in polynomial time and that $|I| \geq |I^*| \geq \frac{1}{c}|I|$.

Observe that G_{aux} is bipartite with partite sets D and I^* . Since G is reduced regarding Rule 3.2, every $[u] \in I^*$ is adjacent to some $v \in D$ in G_{aux} . Furthermore, since D is saturated, every $v \in D$ is adjacent to some $u \in I$ in G and therefore $\{v, [u]\} \in E_{\text{aux}}$. Hence, G_{aux} is a bipartite graph without isolated vertices. Moreover, from $|I| \geq c^2|D|$ and $|I^*| \geq \frac{1}{c}|I|$ we conclude $|I^*| \geq c \cdot |D|$. By applying Lemma 3.5 on G_{aux} we conclude that there exist nonempty vertex sets $X' \subseteq D$ and $Y' \subseteq I^*$ with $N_{G_{\text{aux}}}(Y') \subseteq X'$ that can be computed in polynomial time such that there exists a set $M' \subseteq E_{G_{\text{aux}}}(X', Y')$ of edges, such that every vertex of X' is incident with exactly c edges of M' , and $c \cdot |X'|$ vertices in Y' are endpoints of edges in M' .

We now describe how to construct the sets X , Y , and M from X' , Y' , and M' . We set $X := X' \subseteq D$, and $Y := \bigcup_{[u] \in Y'} [u] \subseteq I$. We prove that $N_G(Y) \subseteq X \cup Y$. Let $y \in Y$. Note that all neighbors of y in I are elements of Y by the definition of the equivalence relation \sim and therefore

$$N_G(y) \subseteq N_{G_{\text{aux}}}([y]) \cup Y \subseteq X' \cup Y = X \cup Y.$$

Next, we construct $M \subseteq E_G(X, Y)$ from M' . To this end we define a mapping $\pi : M' \rightarrow E_G(X, Y)$. For every edge $\{[u], v\} \in M'$ with $[u] \in Y'$ and $v \in X'$ we define $\pi(\{[u], v\}) := \{w, v\}$, where w is some fixed vertex in $[u]$. We set $M := \{\pi(e') \mid e' \in M'\}$. It remains to show that the statements a) and b) hold for M .

a) Observe that $\pi(\{[u_1], v_1\}) = \pi(\{[u_2], v_2\})$ implies $[u_1] = [u_2]$ and $v_1 = v_2$ and therefore, the mapping π is injective. We conclude $|M| = |M'|$. Moreover, observe that the edges of M have the same endpoints in X as the edges of M' . Thus, since every vertex of X' is incident with exactly c edges of M' it follows that statement a) holds for M .

b) By the conditions a) and b) of Lemma 3.5, no two edges in M' have a common endpoint in Y' . Hence, in every connected component in $G[Y]$ there is at most one vertex incident with an edge in M . Moreover, since $|M| = |M'|$ and there are exactly $c \cdot |X'|$ vertices in Y' that are endpoints of edges in M' we conclude that statement b) holds for M . ◀

The following rule is the key rule for our kernelization.

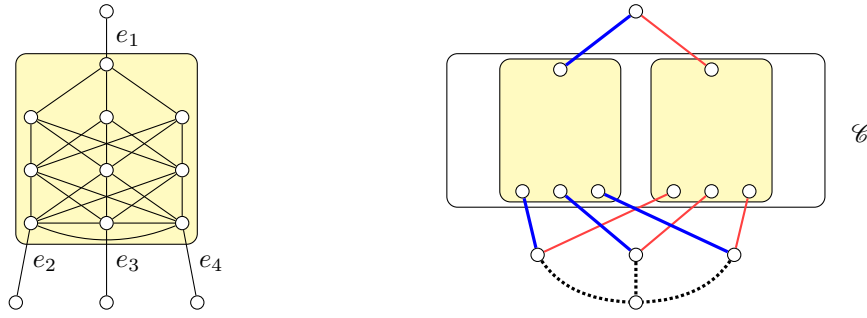
► **Rule 3.3.** *If $|I| \geq c^2 \cdot |D|$, then compute the sets X and Y from Proposition 3.7, delete all vertices in $X \cup Y$ from G , and decrease k by $|E_G(X, V)| - c \cdot |X|$.*

► **Proposition 3.8.** *Rule 3.3 is safe.*

Rules 3.2 and 3.3 together with the fact that we can compute a $(c + 1)$ -approximation of the minimum order- c component cover in polynomial time [20] give us the following.

► **Theorem 1.2.** *ECS admits a problem kernel with $\mathcal{O}(c^3 \cdot \lambda_c)$ vertices.*

Proof. We first consider the running time. We use a $(c + 1)$ -approximation for the minimum order- c component cover and compute an order- c component cover D' in polynomial time [20]. Afterwards we remove any vertex $v \in D'$ with $N(v) \subseteq D'$ from D' while such a vertex exists and we end up with a saturated order- c component cover $D \subseteq D'$. Afterwards, consider Rules 3.2 and 3.3. Obviously, one application of Rule 3.2 can be done in polynomial



■ **Figure 1** Left: A graph where in any STC-labeling with four strong colors and without weak edges, the edges e_1 , e_2 , e_3 , and e_4 are part of the same strong color class. Right: A no-instance of Multi-STC with $c = 4$ and $k = 0$, where Rule 3.1 does not produce an equivalent instance: The inner rectangles correspond to two copies of the gadget on the left. Observe that all blue edges must have a common strong color, and all red edges must have a common strong color distinct that is not blue. Hence, for any STC-labeling of $G[\mathcal{C} \cup N(\mathcal{C})]$ it is not possible to extend the labeling to the dotted edges without violating STC. However, Rule 3.1 converts this no-instance into a yes-instance.

time if D is known. Moreover, Rule 3.3 can also be applied in polynomial time due to Proposition 3.7. Since every application of one of these two rules removes some vertices, we can compute an instance that is reduced regarding Rules 3.2 and 3.3 from an arbitrary input instance of ECS in polynomial time.

We next consider the size of a reduced instance $(G = (V, E), c, k)$ of ECS regarding Rules 3.2 and 3.3. Let $D \subseteq V$ be a $(c+1)$ -approximate saturated order- c component cover, and let $I := V \setminus D$. Since no further application of Rule 3.3 is possible, we conclude $|I| < c^2 \cdot |D|$. Thus, we have $|V| = |I| + |D| < (c^2 + 1) \cdot |D| \leq (c^2 + 1) \cdot (c + 1) \cdot \lambda_c \in \mathcal{O}(c^3 \lambda_c)$. ◀

4 Multi-STC parameterized by Edge Deletion-Distance to Low-Degree Graphs

In this section we provide a problem kernelization for MULTI-STC parameterized by ξ_{c-1} when $c \leq 4$. Before we describe the problem kernel, we briefly show that MULTI-STC does not admit a polynomial kernel for the component order connectivity ξ_{c-1} even if $c = 1$: If $\text{NP} \not\subseteq \text{coNP/poly}$, STC does not admit a polynomial kernel if parameterized by the number of strong edges [10] which – in nontrivial instances – is bigger than the size of a maximal matching M . Since the vertex cover number s is never larger than $2|M|$, this implies that MULTI-STC has no polynomial kernel if parameterized by s unless $\text{NP} \subseteq \text{coNP/poly}$. Since $\lambda_c \leq s$, we conclude that MULTI-STC does not admit a polynomial kernel for λ_c unless $\text{NP} \subseteq \text{coNP/poly}$.

Next, consider parameterization by ξ_{c-1} . Observe that Rule 3.1 which gives a problem kernel for ECS does not work for MULTI-STC; see Figure 1 for an example. Furthermore, for MULTI-STC we need a fundamental new approach: For STC-labelings the maximum degree and the number of colors are not as closely related as in ECS, and therefore, Lemma 3.1 might not be helpful for MULTI-STC. Moreover, in the proof of Lemma 3.2 we exploit that in ECS we may remove weak edges from the instance, which does not hold for MULTI-STC since removing a weak edge may produce P_3 s. However, the results for ECS parameterized by (ξ_{c-1}, c) can be lifted to the seemingly harder MULTI-STC for $c \in \{1, 2, 3, 4\}$. We will first discuss the cases $c = 1$ and $c = 2$. For the cases $c \in \{3, 4\}$ we show the more general statement that MULTI-STC admits a problem kernel with $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c)$ vertices.

If $c = 1$, the parameter $\xi_{c-1} = \xi_0$ equals the number m of edges in G . Hence, MULTI-STC admits a trivial ξ_{c-1} -edge kernel in this case. If $c = 2$, any input graph consists of core vertices \mathcal{C} , periphery vertices in $N(\mathcal{C})$ and isolated vertices and edges. We can compute an equivalent instance in linear time by deleting these isolated components. Afterwards, the graph contains at most $2\xi_{c-1}$ core vertices. Since each of these vertices has at most one neighbor outside \mathcal{C} , we have a total number of $4\xi_{c-1}$ vertices.

To extend this result to $c \in \{3, 4\}$, we now provide a problem kernel for MULTI-STC parameterized by $(c, \xi_{\lfloor \frac{c}{2} \rfloor + 1})$. Let (G, c, k) be an instance of MULTI-STC with edge-deletion set $D := D_{\lfloor \frac{c}{2} \rfloor + 1}$, and let \mathcal{C} and \mathcal{P} be the core and periphery of G . A subset $A \subseteq \mathcal{P}$ is called *periphery component* if it is a connected component in $G[\mathcal{P}]$. Furthermore, for a periphery component $A \subseteq \mathcal{P}$ we define the subset $A^* \subseteq A$ of *close vertices* in A as $A^* := N(\mathcal{C}) \cap A$, that is, the set of vertices of A that are adjacent to core vertices. The key technique of our kernelization is to move weak edges along paths inside periphery components.

► **Definition 4.1.** *Let (G, c, k) be an instance of MULTI-STC with core vertices \mathcal{C} and periphery vertices \mathcal{P} . A periphery component $A \subseteq \mathcal{P}$ is called good, if for every STC-labeling $L = (S_L^1, \dots, S_L^c, W_L)$ for G with $E(A) \subseteq W_L$ there exists an STC-labeling $L' = (S_{L'}^1, \dots, S_{L'}^c, W_{L'})$ for G such that 1. $L'|_{E \setminus E(A)} = L|_{E \setminus E(A)}$, and 2. $W_{L'} \cap E(A) = \emptyset$.*

Intuitively, a good periphery component A is a periphery component where the edges in $E(A)$ can always be added to some strong color classes of an STC-labeling, no matter how the other edges of G are labeled. The condition $E(A) \subseteq W_L$ is a technical condition that makes the proof of the next proposition easier.

► **Proposition 4.2.** *Let (G, c, k) be an instance of MULTI-STC with core vertices \mathcal{C} and periphery vertices \mathcal{P} . Furthermore, let $A \subseteq \mathcal{P}$ be a good periphery component. Then, (G, c, k) is a yes-instance if and only if $(G - (A \setminus A^*), c, k)$ is a yes-instance.*

In the following, we show that for instances (G, c, k) with $c \geq 3$ we can compute an equivalent instance of size $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} c)$. We first consider all cases where $c \geq 3$ is odd. In this case, we can prove that all periphery components are good.

► **Proposition 4.3.** *Let (G, c, k) be an instance of MULTI-STC, where $c \geq 3$ is odd. Moreover, let $A \subseteq \mathcal{P}$ be a periphery component. Then, A is good.*

The Propositions 4.2 and 4.3 guarantee the safeness of the following rule:

► **Rule 4.1.** *If c is odd, remove $A \setminus A^*$ from all periphery components $A \subseteq \mathcal{P}$.*

► **Proposition 4.4.** *Let $(G = (V, E), c, k)$ be an instance of MULTI-STC where $c \geq 3$ is odd. Then, we can compute an instance $(G' = (V', E'), c, k)$ in $\mathcal{O}(n + m)$ time such that $|V'| \leq 2 \cdot \xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot (\lfloor \frac{c}{2} \rfloor + 1)$, and $|E'| \in \mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c^2)$.*

It remains to consider instances where c is an even number and $c \geq 4$. In this case, not every periphery component is good (Figure 1 shows an example), so we need to identify good periphery components more carefully. The first rule removes isolated periphery components.

► **Rule 4.2.** *Remove periphery components $A \subseteq \mathcal{P}$ with $A^* = \emptyset$ from G .*

► **Proposition 4.5.** *Rule 4.2 is safe.*

The intuition for the next lemma is that the small degree of vertices in periphery components can be used to “move” weak edges inside periphery components, the key technique of our kernelization. More precisely, if there is an edge-simple path in a periphery

26:12 Maximum Edge-Colorable Subgraph and Strong Triadic Closure

component, that starts with a weak edge, we can either move the weak edge to the end of that path by keeping the same number of weak edges or find a labeling with fewer weak edges.

► **Lemma 4.6.** *Let $A \subseteq \mathcal{P}$, let L be an STC-labeling of G , and let $e \in W_L \cap E(A)$ be a weak edge in $E(A)$. Furthermore, let $P = (v_1, v_2, \dots, v_{r-1}, v_r)$ be an edge-simple path in $G[A]$ with $\{v_1, v_2\} = e$ and color sequence $Q_L^P = (q_1 = 0, q_2, q_3, \dots, q_{r-1})$ under L . Then, there exists an STC-labeling L' with $L'|_{E \setminus E(P)} = L|_{E \setminus E(P)}$ such that*

$$Q_{L'}^P = (q_2, q_3, \dots, q_{r-1}, 0) \text{ or } |W_{L'}| < |W_L|.$$

Proof. We prove the statement by induction over the length r of P .

Base Case: $r = 2$. Then, $P = (v_1, v_2)$ and $Q_L^P = (0)$. We can trivially define the labeling L' by setting $L' := L$.

Inductive Step: Let $P = (v_1, \dots, v_r)$ be an edge-simple path with color sequence $Q_L^P = (0, q_2, \dots, q_{r-1})$ under L . Consider the edge-simple subpath $P' = (v_1, \dots, v_{r-1})$. By induction hypothesis there exists an STC-labeling L'' for G with $L''|_{E \setminus E(P')} = L|_{E \setminus E(P')}$, such that $Q_{L''}^{P'} = (q_2, q_3, \dots, q_{r-2}, 0)$ or $|W_{L''}| < |W_L|$.

Case 1: $|W_{L''}| < |W_L|$. Then, we define L' by $L' := L''$.

Case 2: $|W_{L''}| \geq |W_L|$. Then, $Q_{L''}^{P'} = (q_2, q_3, \dots, q_{r-2}, 0)$. Since $Q_{L''}^{P'}$ contains the same elements as $Q_L^{P'}$ and $L''|_{E \setminus E(P')} = L|_{E \setminus E(P')}$, we have $|W_{L''}| = |W_L|$.

Case 2.1: There exists an edge $e \neq \{v_{r-1}, v_r\}$ with $e \in S_{L''}^{q_{r-1}}$ that is incident with $\{v_{r-2}, v_{r-1}\}$. From the fact that $\deg(v_{r-2}) \leq \lfloor \frac{c}{2} \rfloor + 1$ and $\deg(v_{r-1}) \leq \lfloor \frac{c}{2} \rfloor + 1$, we conclude that $\{v_{r-2}, v_{r-1}\}$ is incident with at most c other edges of G . Since two of these incident edges have the same strong color q_{r-1} under L'' , the edge $\{v_{r-2}, v_{r-1}\}$ is incident with at most $c-1$ edges of distinct strong colors under L'' . Consequently, there exists a strong color $i \in \{1, \dots, c\}$, such that $\{v_{r-2}, v_{r-1}\}$ can safely be added to the strong color class $S_{L''}^i$ and be removed from $W_{L''}$ without producing any strong P_3 . This way, we transformed L'' into an STC-labeling L' , such that $L'|_{E \setminus E(P')} = L|_{E \setminus E(P')}$ and $|W_{L'}| < |W_L|$.

Case 2.2: No edge $e \neq \{v_{r-1}, v_r\}$ with $e \in S_{L''}^{q_{r-1}}$ is incident with $\{v_{r-2}, v_{r-1}\}$. We then define L' by

$$\begin{aligned} W_{L'} &:= W_{L''} \cup \{\{v_{r-1}, v_r\}\} \setminus \{\{v_{r-2}, v_{r-1}\}\}, \text{ and} \\ S_{L'}^{q_{r-1}} &:= S_{L''}^{q_{r-1}} \cup \{\{v_{r-2}, v_{r-1}\}\} \setminus \{\{v_{r-1}, v_r\}\}. \end{aligned}$$

Note that $Q_{L'}^P = (q_2, q_3, \dots, q_{r-1}, 0)$ and $L'|_{E \setminus E(P)} = L|_{E \setminus E(P)}$. Moreover, since P is edge-simple, the edge $\{v_{r-1}, v_r\}$ does not lie on P' and since $L''|_{E \setminus E(P')} = L|_{E \setminus E(P')}$, it holds that $\{v_{r-1}, v_r\} \in S_{L''}^{q_{r-1}}$. Therefore, every edge has exactly one color under L' . It remains to show that L' satisfies STC. Assume towards a contradiction, that this is not the case. Then, since L'' satisfies STC, there exists an induced P_3 on $\{v_{r-2}, v_{r-1}\} \in S_{L'}^{q_{r-1}}$ and some edge $e \in S_{L'}^{q_{r-1}}$. Since $\{v_{r-1}, v_r\} \in W_{L'}$ and $L'|_{E \setminus \{\{v_{r-2}, v_{r-1}\}, \{v_{r-1}, v_r\}\}} = L''|_{E \setminus \{\{v_{r-2}, v_{r-1}\}, \{v_{r-1}, v_r\}\}}$, the edge $e \neq \{v_{r-1}, v_r\}$ is incident with $\{v_{r-2}, v_{r-1}\}$ and it holds that $e \in S_{L''}^{q_{r-1}}$. This contradicts the condition of Case 2.2. ◀

We will now use Lemma 4.6 to show useful properties of periphery components. First, if there are two weak edges in one periphery component A , we can make these two weak edges incident, which then helps us to define a new labeling that has fewer weak edges in A :

► **Proposition 4.7.** *Let $A \subseteq \mathcal{P}$ be a periphery component and let L be an STC-labeling for G . Then, there exists an STC-labeling L' with $L'|_{E \setminus E(A)} = L|_{E \setminus E(A)}$ and $|W_{L'} \cap E(A)| \leq 1$.*

Proof. If $|W_L \cap E(A)| \leq 1$ the statement already holds for $L' = L$. So, assume there are two distinct edges $e_1, e_2 \in W_L \cap E(A)$. In this case, we construct an STC-labeling which is partially equal to L on $E \setminus E(A)$ and has strictly fewer weak edges in $E(A)$ than L , which then proves the claim.

Since periphery components are connected components in $G[\mathcal{P}]$, there exists an edge-simple path $P = (v_1, \dots, v_r)$ in $G[A]$ such that $e_1 = \{v_1, v_2\}$ and $e_2 = \{v_{r-1}, v_r\}$. Applying Lemma 4.6 on the edge-simple subpath $P' = (v_1, \dots, v_{r-1})$ gives us an STC-labeling L' with $L'|_{E \setminus E(P)} = L|_{E \setminus E(P)}$ such that $|W_{L'}| < |W_L|$ or $Q_{L'}^{P'} = (q_2, q_3, \dots, q_{r-2}, 0)$.

In case of $|W_{L'}| < |W_L|$, nothing more needs to be shown. So, assume $|W_{L'}| = |W_L|$. It follows that $Q_{L'}^{P'} = (q_2, q_3, \dots, q_{r-2}, 0)$ and therefore $Q_{L'}^P = (q_2, q_3, \dots, q_{r-2}, 0, 0)$. Then, e_1 and e_2 are weak under L' . Since $\deg(v_{r-1}) \leq \lfloor \frac{c}{2} \rfloor + 1$ and $\deg(v_r) \leq \lfloor \frac{c}{2} \rfloor + 1$, the edge e_2 is incident with at most c edges. Since at least one of these incident edges is weak, e_2 is incident with at most $c-1$ edges of distinct strong colors. Consequently, there exists a strong color $i \in \{1, \dots, c\}$ such that e_2 can be added to the strong color class $S_{L'}^i$ and deleted from $W_{L'}$ without violating STC. This way, we transformed L' into an STC-labeling L'' such that $L''|_{E \setminus E(A)} = L|_{E \setminus E(A)}$ and $|W_{L''} \cap E(A)| < |W_L \cap E(A)|$. ◀

Next, we use Proposition 4.7 to identify specific good components.

► **Proposition 4.8.** *Let $A \subseteq \mathcal{P}$ be a periphery component such that some edge $\{u, v\} \in E(A)$ forms an induced P_3 with less than c other edges in G . Then, A is good.*

Proof. Let L be an arbitrary STC-labeling for G with $E(A) \subseteq W_L$. We prove that there is an STC-labeling which is partially equal to L on $E \setminus E(A)$ and has no weak edges in $E(A)$.

Let L' be an STC-labeling for G with $L'|_{E \setminus E(A)} = L|_{E \setminus E(A)}$. If $W_{L'} \cap E(A) = \emptyset$, nothing more needs to be shown. So, let $W_{L'} \cap E(A) \neq \emptyset$. By Proposition 4.7 we can assume that there is one unique edge $e \in W_{L'} \cap E(A)$. Since A is a connected component in $G[\mathcal{P}]$, there exists an edge-simple path $P = (v_1, \dots, v_r)$ such that $\{v_1, v_2\} = e$, and $\{v_{r-1}, v_r\} = \{u, v\}$ with $Q_{L'}^P = (0, q_2, \dots, q_{r-1})$. By Lemma 4.6, there exists an STC-labeling L'' with $L''|_{E \setminus E(A)} = L|_{E \setminus E(A)}$ such that $|W_{L''}| < |W_L|$ or $Q_{L''}^P = (q_2, \dots, q_{r-1}, 0)$. In case of $|W_{L''}| < |W_L|$, nothing more needs to be shown. Otherwise, the edge e is weak under L'' . Since e is part of less than c induced P_3 s in G , there exists one strong color $i \in \{1, \dots, c\}$, such that e can safely be added to $S_{L''}^i$ and be removed from $W_{L''}$ without violating STC. This way, we transform L'' into an STC-labeling L''' with $L'''|_{E \setminus E(A)} = L|_{E \setminus E(A)}$ and $W_{L'''} \cap E(A) = \emptyset$.

Since L was arbitrary, the periphery component A is good by definition. ◀

► **Proposition 4.9.** *Let $A \subseteq \mathcal{P}$ be a periphery component such that there exists a vertex $v \in A$ with $\deg_G(v) < \lfloor \frac{c}{2} \rfloor + 1$. Then, A is good.*

Proof. If $|A| = 1$, then A is obviously good, since $E(A) = \emptyset$. Let $|A| \geq 2$. Since A contains at least two vertices and forms a connected component in $G[\mathcal{P}]$ there exists a vertex $u \in A$, such that $\{u, v\} \in E(A)$. Since $\deg_G(v) < \lfloor \frac{c}{2} \rfloor + 1$, and $\deg_G(u) \leq \lfloor \frac{c}{2} \rfloor + 1$, the edge $\{u, v\}$ forms induced P_3 s with less than c other edges in G . Then, by Proposition 4.8 we conclude that A is good. ◀

Propositions 4.2 and 4.9 guarantee the safeness of the following rule.

► **Rule 4.3.** *If there is a periphery component $A \subseteq \mathcal{P}$ with $A \setminus A^* \neq \emptyset$ such that there exists a vertex $v \in A$ with $\deg(v) < \lfloor \frac{c}{2} \rfloor + 1$, then delete $A \setminus A^*$ from G .*

► **Proposition 4.10.** *Let $A \subseteq \mathcal{P}$ be a periphery component such that there exists an edge $\{u, v\} \in E(A)$ which is part of a triangle $G[\{u, v, w\}]$ in G . Then, A is good.*

26:14 Maximum Edge-Colorable Subgraph and Strong Triadic Closure

Proof. Since $u, v \in A$, we know $\deg_G(u) \leq \lfloor \frac{c}{2} \rfloor + 1$ and $\deg_G(v) \leq \lfloor \frac{c}{2} \rfloor + 1$. Since u, v are part of a triangle in G , it follows that $\{u, v\}$ forms an induced P_3 with less than c other edges in G . Then, by Proposition 4.8 we conclude that A is good. \blacktriangleleft

Propositions 4.2 and 4.10 guarantee the safeness of the following rule.

► **Rule 4.4.** *If there is a periphery component $A \subseteq \mathcal{P}$ with $A \setminus A^* \neq \emptyset$ such that there exists an edge $\{u, v\} \in A$ which is part of a triangle $G[\{u, v, w\}]$ in G , then delete $A \setminus A^*$ from G .*

For the rest of this section we consider instances (G, c, k) for MULTI-STC, that are reduced regarding Rules 4.2–4.4. Observe that these instances only contain triangle-free periphery components A where every vertex $v \in A$ has $\deg(v) = \lfloor \frac{c}{2} \rfloor + 1$. Since ECS and MULTI-STC are the same on triangle-free graphs one might get the impression that we can use Vizing’s Theorem to prove that all periphery components in G are good. Consider the example in Figure 1 to see that this is not necessarily the case.

We now continue with the description of the kernel for MULTI-STC. Let (G, c, k) be an instance of MULTI-STC that is reduced regarding Rules 4.2–4.4. We analyze the periphery components of G that contain cycles. In this context, a *cycle* (of length r) is an edge-simple path $P = (v_0, v_1, \dots, v_{r-1}, v_0)$ where the last vertex and the first vertex of P are the same, and all other vertices occur at most once in P . We will see that acyclic periphery components – which are periphery components $A \subseteq \mathcal{P}$ where $G[A]$ is a tree – are already bounded in c and $\xi_{\lfloor \frac{c}{2} \rfloor + 1}$. To remove the other components, we show that periphery components with cycles are always good. To this end we show two lemmas. The intuitive idea behind Lemmas 4.11 and 4.12 is, that we use Lemma 4.6 to rotate weak and strong edge-colors around a cycle.

► **Lemma 4.11.** *Let $A \subseteq \mathcal{P}$ be a periphery component, and let L be an STC-labeling for G . Moreover, let $P = (v_0, v_1, \dots, v_{r-1}, v_0)$ be a cycle in A such that $W_L \cap E(P) \neq \emptyset$ and let $Q_L^P = (q_0, q_1, \dots, q_{r-1})$ be the color sequence of P under L . Then, there exist STC-labelings $L_0, L_1, L_2, \dots, L_{r-1}$ for G such that $L_i|_{E \setminus E(P)} = L|_{E \setminus E(P)}$ and*

$$Q_{L_i}^P(j) = q_{(i+j) \bmod r} \text{ or } |W_{L_i}| < |W_L|$$

for all $i, j \in \{0, \dots, r-1\}$.

Proof. Without loss of generality we assume that $\{v_0, v_1\} \in W_L$ and therefore $q_0 = 0$. We prove the existence of the labelings L_i with $i \in \{0, 1, \dots, r-1\}$ by induction over i .

Base Case: $i = 0$. In this case we set $L_0 := L$.

Inductive Step: By inductive hypothesis, there is a labeling L_{i-1} with $|W_{L_{i-1}}| < |W_L|$ or

$$Q_{L_{i-1}}^P(j) = q_{(i-1+j) \bmod r}.$$

If $|W_{L_{i-1}}| < |W_L|$, then we define L_i by $L_i := L_{i-1}$ and nothing more needs to be shown. Otherwise, we consider $P' = (v_{r-i+1}, v_{r-i+2}, \dots, v_{r-1}, v_0, v_1, \dots, v_{r-i+1})$. Note that P' describes the same cycle as P by rotating the vertices. More precisely,

$$P(j) = P'((j+i-1) \bmod r).$$

Therefore, P' is edge-simple and has the color sequence $Q_{L_{i-1}}^{P'} = (q_0 = 0, q_1, \dots, q_{r-1})$. By Lemma 4.6, there exists an STC-labeling L_i with $L_i|_{E \setminus E(P)} = L_{i-1}|_{E \setminus E(P)}$, such that $|W_{L_i}| < |W_{L_{i-1}}|$ or

$$Q_{L_i}^{P'}(j) = q_{(j+1) \bmod r}.$$

In case of $|W_{L_i}| < |W_{L_{i-1}}|$, nothing more needs to be shown. Otherwise, observe that

$$Q_{L_i}^P(j) = Q_{L_i}^{P'}((j+i-1) \bmod r) = q_{(j+i) \bmod r}$$

which completes the inductive step. \blacktriangleleft

► **Lemma 4.12.** *Let $A \subseteq \mathcal{P}$ be a periphery component, let L be an STC-labeling. Moreover, let $P = (v_0, v_1, \dots, v_{r-1}, v_0)$ be a cycle in A with $W_L \cap E(P) \neq \emptyset$, and let $e_1, e_2 \in E(P)$ with $e_2 \in S_L^q$ for some strong color $q \in \{1, \dots, c\}$. Then, there exists an STC-labeling L' with $L'|_{E \setminus E(P)} = L|_{E \setminus E(P)}$ such that $e_1 \in S_{L'}^q$ or $|W_{L'}| < |W_L|$.*

Proof. Let $Q_L^P := (q_0, q_1, \dots, q_{r-1})$. Without loss of generality assume that $\{v_0, v_1\} \in W_L$ and $e_2 = \{v_t, v_{t+1}\}$ for some $t \in \{1, \dots, r-1\}$. It then holds, that $q_0 = 0$, and $q = q_t$. Furthermore, since $e_1 \in E(P)$ we have $e_1 = \{P(j), P(j+1)\}$ for some $j \in \{0, 1, \dots, r-1\}$.

Consider the STC-labelings $L_0, L_1, L_2, \dots, L_{r-1}$ from Lemma 4.11. If for one such labeling L_i it holds that $|W_{L_i}| < |W_L|$, then nothing more needs to be proven. Otherwise, set $i := (t-j) \bmod r$. We show that $e_1 \in S_{L_i}^{q_t}$ by proving $Q_{L_i}^P(j) = q_t$ as follows:

$$Q_{L_i}^P(j) = q_{(i+j) \bmod r} = q_{((t-j) \bmod r + j) \bmod r} = q_{(t-j+j) \bmod r} = q_t. \quad \blacktriangleleft$$

We next use Lemma 4.12 to prove that periphery components with cycles are good.

► **Proposition 4.13.** *Let $(G = (V, E), c, k)$ be a reduced instance of MULTI-STC regarding rules 4.2–4.4, where $c \geq 4$ is even. Let $A \subseteq \mathcal{P}$ be a periphery component in G such that $A \setminus A^* \neq \emptyset$ and there is a cycle $P = (v_0, v_1, \dots, v_{r-1}, v_0)$ in $G[A]$. Then, A is good.*

Propositions 4.13 and 4.2 imply the safeness of the final rule which together with Rules 4.2–4.4 gives the kernel.

► **Rule 4.5.** *If there is a periphery component $A \subseteq \mathcal{P}$ with $A \setminus A^* \neq \emptyset$ such that there exists a cycle P in $G[A]$, then delete $A \setminus A^*$ from G .*

► **Theorem 4.14.** *MULTI-STC restricted to instances with $c \geq 3$ admits a problem kernel with $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c)$ vertices and $\mathcal{O}(\xi_{\lfloor \frac{c}{2} \rfloor + 1} \cdot c^2)$ edges that can be computed in $\mathcal{O}(n + m)$ time.*

Proof. Throughout this proof let $\xi := 2\xi_{\lfloor \frac{c}{2} \rfloor + 1}$ denote the size of a 2-approximate edge-deletion set $D_{\lfloor \frac{c}{2} \rfloor + 1}$ of G and $\lfloor \frac{c}{2} \rfloor + 1$. We defer the proof of the running time and show that $|V'| \leq (c+7) \cdot \xi$. Let \mathcal{C} be the set of core vertices of G' and \mathcal{P} be the set of periphery vertices of G' . Since $|\mathcal{C}| \leq 2\xi$, and every $v \in \mathcal{C}$ is incident with at most $\frac{c}{2} + 1$ edges, there are $2\xi + 2\xi(\frac{c}{2} + 1) = \xi c + 4\xi$ vertices in $\mathcal{C} \cup N(\mathcal{C})$. It remains to show that there are at most 3ξ non-close vertices in \mathcal{P} . Consider the family $\mathcal{A} := \{A \subseteq \mathcal{P} \mid A \text{ is periphery component with } A \setminus A^* \neq \emptyset\}$ of periphery components.

Since G' is reduced regarding Rules 4.3, 4.4, and 4.5, every $G[A]$ with $A \in \mathcal{A}$ is a tree, where every vertex $v \in A$ has degree $\deg_G(v) = \frac{c}{2} + 1$ in G . We define a *leaf vertex* as a vertex $v \in \bigcup_{A \in \mathcal{A}} A$ with $\deg_{G[\mathcal{P}]}(v) = 1$. Note that these vertices are exactly the leaves of a tree $G[A]$ for some $A \in \mathcal{A}$, and all leaf vertices are close vertices in \mathcal{P} . Let p be the number of leaf vertices. We show that $p \leq 3\xi$. Since (G', c, k) is reduced regarding Rule 4.3, every vertex $v \in \bigcup_{A \in \mathcal{A}} A$ has a degree of $\deg_G(v) = \frac{c}{2} + 1$, hence every leaf vertex has exactly $\frac{c}{2}$ neighbors in \mathcal{C} . We thus have $p \cdot \frac{c}{2} \leq |E(\mathcal{C}, N(\mathcal{C}))| \leq 2\xi(\frac{c}{2} + 1)$, and therefore $p \leq 2\xi + \frac{4\xi}{c} \leq 3\xi$, since $c \geq 4$. Recall that every non-close vertex v in some tree $G[A]$ satisfies $\deg_{G[A]}(v) = \frac{c}{2} + 1 > 2$. Since a tree has at most as many vertices with degree at least three as it has leaves, we conclude $|\bigcup_{A \in \mathcal{A}} A \setminus (\bigcup_{A \in \mathcal{A}} A^*)| \leq 3\xi$. Hence, there are at most 3ξ non-close vertices in \mathcal{P} . Then, G' contains of at most $(c+7) \cdot \xi \in \mathcal{O}(\xi c)$ vertices, as claimed. Since each vertex is incident with at most $\frac{c}{2} + 1$ edges, G' has $\mathcal{O}(\xi c^2)$ edges. \blacktriangleleft

5 Conclusion

In this work, we showed that MAXIMUM EDGE-COLORABLE SUBGRAPH with c colors is tractable on instances that have small edge-deletion distance to graphs whose maximum degree is $c - 1$. This result implies that EDGE COLORING with c colors is fixed-parameter tractable with respect to the combination of c and the number of vertices that have degree c . For MULTI STRONG TRIADIC CLOSURE with c colors, we obtain fixed-parameter algorithms for the same parameter for $c \leq 4$. For MULTI STRONG TRIADIC CLOSURE with $c \geq 5$, the parameter in our fixed-parameter algorithms is the edge-deletion distance to graphs with maximum degree c' for some $c' < c - 1$.

There are several ways of extending our results that seem interesting topics for future research. First, in our fixed-parameter algorithms the value of c is always part of the parameter and it would be very interesting to understand whether this is necessary. For example, is EDGE COLORING fixed-parameter tractable with respect to the number of vertices that have degree at least c alone? Second, our results are obtained via kernelizations. Are there any direct fixed-parameter algorithms that achieve a better running time than using kernelization and brute-force on the kernels? Third, can our results for MULTI STRONG TRIADIC CLOSURE and $c \geq 5$ be improved to fixed-parameter algorithms for the edge-deletion distance to graphs with maximum degree $c - 1$? Moreover, our parameters use the *edge-deletion* distance to tractable special cases. Can one improve these results by obtaining fixed-parameter algorithms for the *vertex-deletion* distance? Finally, in our parameterization for MULTI STRONG TRIADIC CLOSURE we use the fact that MULTI STRONG TRIADIC CLOSURE is polynomial-time solvable on graphs with maximum degree $c - 1$. This is a simple corollary of Vizing's theorem and the fact that every proper edge coloring is a valid coloring for MULTI STRONG TRIADIC CLOSURE. It would be nice to extend the class of tractable instances further in the following sense: For which superclasses of the graphs with maximum degree $c - 1$ does MULTI STRONG TRIADIC CLOSURE remain polynomial-time solvable? Surely, our fixed-parameter algorithms give such superclasses but are there some that can be described without the use of parameters, for example via a characterization of forbidden induced subgraphs of size at most $f(c)$?

References

- 1 Laurent Bulteau, Niels Grüttemeier, Christian Komusiewicz, and Manuel Sorge. Your rugby mates don't need to know your colleagues: Triadic closure with edge colors. In *Proc. 11th CIAC*, volume 11485 of *LNCS*, pages 99–111. Springer, 2019.
- 2 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proc. 30th WG*, volume 3353 of *LNCS*, pages 257–269. Springer, 2004.
- 3 Richard Cole and John E. Hopcroft. On edge coloring bipartite graphs. *SIAM J. Comput.*, 11(3):540–546, 1982.
- 4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 5 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 6 Uriel Feige, Eran Ofek, and Udi Wieder. Approximating maximum edge coloring in multigraphs. In *Proc. 5th APPROX*, volume 2462 of *LNCS*, pages 108–121. Springer, 2002.
- 7 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

- 8 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. 15th STOC*, pages 448–456. ACM, 1983.
- 9 Petr A. Golovach, Pinar Heggernes, Athanasios L. Konstantinidis, Paloma T. Lima, and Charis Papadopoulos. Parameterized aspects of strong subgraph closure. In *Proc. 16th SWAT*, volume 101 of *LIPICs*, pages 23:1–23:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 10 Niels Grüttemeier and Christian Komusiewicz. On the relation of strong triadic closure and cluster deletion. *Algorithmica*, 82(4):853–880, 2020.
- 11 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proc. 1st IWPEC*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004.
- 12 S. Louis Hakimi and Oded Kariv. A generalization of edge-coloring in graphs. *J. Graph Theor.*, 10(2):139–154, 1986.
- 13 Ian Holyer. The NP-Completeness of Edge-Coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.
- 14 Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011.
- 15 Marcin Jakub Kaminski and Lukasz Kowalik. Beyond the Vizing’s bound for at most seven colors. *SIAM J. Discrete Math.*, 28(3):1334–1362, 2014.
- 16 Athanasios L. Konstantinidis, Stavros D. Nikolopoulos, and Charis Papadopoulos. Strong triadic closure in cographs and graphs of low maximum degree. *Theor. Comput. Sci.*, 740:76–84, 2018.
- 17 Athanasios L. Konstantinidis and Charis Papadopoulos. Maximizing the Strong Triadic Closure in Split Graphs and Proper Interval Graphs. In *Proc. 28th ISAAC*, volume 92 of *LIPICs*, pages 53:1–53:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- 18 Adrian Kosowski. Approximating the maximum 2- and 3-edge-colorable subgraph problems. *Discrete Appl. Math.*, 157(17):3593–3600, 2009.
- 19 Lukasz Kowalik. Improved edge-coloring with three colors. *Theor. Comput. Sci.*, 410(38–40):3733–3742, 2009.
- 20 Mithilesh Kumar and Daniel Lokshtanov. A $2\ell k$ kernel for ℓ -component order connectivity. In *Proc. 11th IPEC*, pages 20:1–20:14, 2016.
- 21 Daniel Leven and Zvi Galil. NP completeness of finding the chromatic index of regular graphs. *J. Algorithms*, 4(1):35–44, 1983.
- 22 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- 23 Romeo Rizzi. Approximating the maximum 3-edge-colorable subgraph problem. *Discrete Math.*, 309(12):4166–4170, 2009.
- 24 Elena Prieto Rodríguez. *Systematic kernelization in FPT algorithm design*. PhD thesis, The University of Newcastle, 2005.
- 25 Stavros Sintos and Panayiotis Tsaparas. Using strong triadic closure to characterize ties in social networks. In *Proc. 20th KDD*, pages 1466–1475. ACM, 2014.
- 26 Michael Stiebitz, Diego Scheide, Bjarne Toft, and Lene M Favrholt. *Graph edge coloring: Vizing’s theorem and Goldberg’s conjecture*, volume 75. John Wiley & Sons, 2012.
- 27 Vadim G Vizing. On an estimate of the chromatic class of a p -graph. *Discret Analiz*, 3:25–30, 1964.

Preprocessing Vertex-Deletion Problems: Characterizing Graph Properties by Low-Rank Adjacencies

Bart M. P. Jansen 

Eindhoven University of Technology, The Netherlands
b.m.p.jansen@tue.nl

Jari J. H. de Kroon 

Eindhoven University of Technology, The Netherlands
j.j.h.d.kroon@tue.nl

Abstract

We consider the Π -FREE DELETION problem parameterized by the size of a vertex cover, for a range of graph properties Π . Given an input graph G , this problem asks whether there is a subset of at most k vertices whose removal ensures the resulting graph does not contain a graph from Π as induced subgraph. Many vertex-deletion problems such as PERFECT DELETION, WHEEL-FREE DELETION, and INTERVAL DELETION fit into this framework. We introduce the concept of *characterizing a graph property Π by low-rank adjacencies*, and use it as the cornerstone of a general kernelization theorem for Π -FREE DELETION parameterized by the size of a vertex cover. The resulting framework captures problems such as AT-FREE DELETION, WHEEL-FREE DELETION, and INTERVAL DELETION. Moreover, our new framework shows that the vertex-deletion problem to perfect graphs has a polynomial kernel when parameterized by vertex cover, thereby resolving an open question by Fomin et al. [JCSS 2014]. Our main technical contribution shows how linear-algebraic dependence of suitably defined vectors over \mathbb{F}_2 implies graph-theoretic statements about the presence of forbidden induced subgraphs.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms; Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Graph theory

Keywords and phrases kernelization, vertex-deletion, graph modification, structural parameterization

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.27

Related Version A full version of the paper is available at <https://arxiv.org/abs/2004.08818>.

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).



Acknowledgements We would like to thank Fedor V. Fomin for hosting Jari in Bergen (Norway).

1 Introduction

Background. This paper continues a long line of investigation [2, 3, 13, 15, 19, 27], aimed at answering the following question: how and when can an efficient preprocessing algorithm reduce the size of inputs to NP-hard problems, without changing their answers? This question can be framed and answered using the notion of kernelization, which originated in parameterized complexity theory.

In parameterized complexity theory, the complexity analysis is done not only in the size of the input, but also in terms of another complexity measure related to the input. This complexity measure is called the *parameter*. For graph problems, typical parameters are



© Bart M. P. Jansen and Jari J. H. de Kroon;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the size of a solution, the treewidth of the graph, or the size of a minimum vertex cover (the *vertex cover number*). The latter two are often called structural parameterizations. A kernelization is a polynomial-time preprocessing algorithm with a performance guarantee. It reduces an instance (x, k) of a parameterized problem to an instance (x', k') that has an equivalent YES/NO answer, such that $|x'|$ and k' are bounded by $f(k)$ for some computable function f , called the size of the kernel. If f is a polynomial function, the parameterized problem is said to admit a polynomial kernel. Polynomial kernels are highly sought after, as they allow problem instances to be reduced to a relatively small size.

We investigate polynomial kernels for the class of *graph modification* problems, in an attempt to develop a widely applicable and generic kernelization framework. In graph modification problems, the goal is to make a small number of changes to an input graph to make it satisfy a certain property. Possible modifications are vertex deletions, edge deletions, and edge additions. In this work, we consider the problem of deleting a bounded-size set of vertices such that the resulting graph does not contain certain graphs as an induced subgraph.

The study of kernelization for graph modification problems parameterized by solution size has an interesting and rich history [1, 6, 7, 10, 14, 17, 20, 23]. However, some graph modification problems such as PERFECT VERTEX DELETION [16] and WHEEL-FREE VERTEX DELETION [25] are $W[2]$ -hard parameterized by the solution size and therefore do not admit any kernels unless $FPT = W[2]$. Together with the intrinsic interest in obtaining generic kernelization theorems that apply to a large class of problems with a single parameter, this has triggered research into polynomial kernelization for graph problems under structural parameterizations [4, 13, 15, 19, 26] such as the vertex cover number. The latter parameter is often used for its mathematical elegance, and due to the fact that slightly less restrictive parameters such as the feedback vertex number already cause simple problems such as 3-COLORING not to admit polynomial kernels [18], under the standard assumption $NP \not\subseteq coNP/poly$. This work therefore focuses on the following class of NP-hard [24] parameterized problems, where Π is a fixed (possibly infinite) set of graphs:

Π -FREE DELETION

Parameter: $|X|$

Input: A graph G , a vertex cover X of G , and an integer k .

Question: Does there exist a set $S \subseteq V(G)$ of size at most k such that $G - S$ does not contain any graph from Π as induced subgraph?

The assumption that a vertex cover X is given in the input is for technical reasons. If the problem would be parameterized by an upper-bound on the vertex cover number of the graph, without giving such a vertex cover, then the kernelization algorithm would have to verify that this is indeed a correct upper bound; an NP-hard problem. Instead, in this setting we just want to allow the kernelization algorithm to exploit the structural restriction guaranteed by having a small vertex cover in the graph. We refer to the discussion by Fellows et al. [12, §2.2] for more background. To apply the kernelization algorithms for problems defined in this way, one may simply use a 2-approximate vertex cover as X .

Fomin et al. [13] have investigated characteristics of Π -FREE DELETION problems that admit a polynomial kernel parameterized by the size of a vertex cover. They introduced a generic framework that poses three conditions on the graph property Π , which are sufficient to reach a polynomial kernel for Π -FREE DELETION parameterized by vertex cover. Examples of graph properties that fit in their framework are for instance “having a chordless cycle of length at least 4” or “having an odd cycle”. This results in polynomial kernels for CHORDAL DELETION and ODD CYCLE TRANSVERSAL respectively. INTERVAL DELETION does not fit

■ **Table 1** Kernels obtained by our framework for problems parameterized by a vertex cover X .

Problem	Vertices in kernel
PERFECT DELETION	$\mathcal{O}(X ^5)$
EVEN-HOLE-FREE DELETION (★)	$\mathcal{O}(X ^4)$
AT-FREE DELETION	$\mathcal{O}(X ^9)$
INTERVAL DELETION	$\mathcal{O}(X ^9)$
WHEEL-FREE DELETION	$\mathcal{O}(X ^5)$

in this framework, even though interval graphs are hereditary. Agrawal et al. [1] show that it admits a polynomial kernel parameterized by solution size, and therefore also by vertex cover size. They introduced a linear-algebraic technique, which assigns a vector over \mathbb{F}_2 to each vertex, to find an induced subgraph that preserves the size of an optimal solution by combining several disjoint bases of systems of such vectors. This formed the inspiration for our work, in which we improve the generic kernelization framework of Fomin et al. [13] using linear-algebraic techniques inspired by the kernel [1] for INTERVAL DELETION.

Results. We introduce the notion of *characterizing a graph property Π by low-rank adjacencies*, and use it to generalize the kernelization framework by Fomin et al. [13] significantly. The resulting kernelization algorithms consist of a single, conceptually simple reduction rule for Π -FREE DELETION, whose property-specific correctness proofs show how the linear dependence of suitably defined vectors implies certain graph-theoretic properties. This results in a simpler kernelization for INTERVAL DELETION parameterized by vertex cover compared to the one by Agrawal et al. [1]. More importantly, several vertex-deletion problems whose kernelization complexity was previously open can be covered by the framework. These include AT-FREE DELETION (eliminate all asteroidal triples [22] from the graph), WHEEL-FREE DELETION, and also PERFECT DELETION which was an explicit open question of Fomin et al. [13, §5]. An overview is given in Table 1. Moreover, we give evidence that the distinguishing property of our framework (being able to characterize Π by low-rank adjacencies) is the right one to capture kernelization complexity. While the WHEEL-FREE DELETION problem fits into our framework and therefore has a polynomial kernel, the situation is very different for the related problem ALMOST WHEEL-FREE DELETION (ensure the resulting graph does not contain any wheel, except possibly W_4). We prove the latter problem does not fit into our framework, and that it does not admit a polynomial kernel parameterized by vertex cover, unless $\text{NP} \subseteq \text{coNP/poly}$.

Related work. Even though the vertex cover is generally not small compared to the size of the input graph, it is not always the case that a polynomial kernel parameterized by vertex cover number exists. This was shown by Bodlaender et al. [3]. They showed that for instance the CLIQUE problem that asks whether a graph contains a clique of k vertices, does not admit a polynomial kernel parameterized by the vertex cover size, unless $\text{coNP} \subseteq \text{NP/poly}$.

A graph is perfect if for every induced subgraph H , the chromatic number of H is equal to the size of the largest clique of H . Conjectured by Berge in 1961 and proven in the beginning of this century by Chudnovsky et al. [8], the strong perfect graph theorem states that a graph is perfect if and only if it is Berge. The forbidden induced subgraphs of Berge graphs (and hence of perfect graphs) are C_{2k+1} and \overline{C}_{2k+1} for $k \geq 2$, that is, induced cycles and their edge complements of odd length at least 5. A survey of forbidden subgraph characterizations of some other hereditary graph classes is given in [5, Chapter 7].

Organization. In Section 2 we give preliminaries and definitions used throughout this work. In Section 3 we introduce the framework. In Section 4 we show that several problems such as PERFECT DELETION and INTERVAL DELETION fit in this framework. Finally we conclude in Section 5. For statements marked \star , the proof is deferred to the full version [21].

2 Preliminaries

Notation. For $i \in \mathbb{N}$, we denote the set $\{1, \dots, i\}$ by $[i]$. For a set S , we denote the set of subsets of size at most k by $\binom{S}{\leq k} = \{S' \subseteq S \mid |S'| \leq k\}$. Similarly, $\binom{S}{k}$ denotes the set of subsets of size exactly k . We consider simple graphs that are unweighted and undirected without self-loops. A graph G has vertex and edge sets $V(G)$ and $E(G)$ respectively. An edge between vertices $u, v \in V(G)$ is an unordered pair $\{u, v\}$. For a set of vertices $S \subseteq V(G)$, by $G[S]$ we denote the graph induced by S . For $v \in V(G)$ and $S \subseteq V(G)$, by $G - v$ and $G - S$ we mean the graphs $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus S]$ respectively. We denote the open neighborhood of $v \in V(G)$ by $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$. When clear from context, we sometimes omit the subscript G . For a graph G , let \bar{G} be the edge complement graph of G on the same vertex set, such that for distinct $u, v \in V(G)$ we have $\{u, v\} \in E(\bar{G})$ if and only if $\{u, v\} \notin E(G)$. The path graph on n vertices (v_1, \dots, v_n) is denoted by P_n . Similarly, the n -vertex cycle for $n \geq 3$ is denoted by C_n . When $n \geq 4$, the graph C_n is often called a *hole*. For $n \geq 3$, the wheel W_n of size n is the graph on vertices $\{c, v_1, \dots, v_n\}$ such that (v_1, \dots, v_n) is a cycle and c is adjacent to v_i for all $i \in [n]$. An asteroidal triple (AT) in a graph G consists of three vertices such that every pair is connected by a path that avoids the neighborhood of the third. A vertex cover in a graph G is a set of vertices that contains at least one endpoint of every edge. The minimum size of a vertex cover in a graph G is denoted by $\text{vc}(G)$.

Parameterized complexity. A parameterized problem [9, 11] is a language $Q \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The notion of kernelization is formalized as follows.

► **Definition 1.** Let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem and let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A kernelization for Q of size f is an algorithm that, given an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs in time polynomial in $|x| + k$ an instance (x', k') (known as the kernel) such that $(x, k) \in Q$ if and only if $(x', k') \in Q$ and such that $|x'|, k' \leq f(k)$. If f is a polynomial function, then the algorithm is a polynomial kernelization.

Previous kernelization framework. We state some of the results from the kernelization framework by Fomin et al. [13] that forms the basis of this work. A graph property Π is a (possibly infinite) set of graphs.

► **Definition 2** (Definition 3, [13]). A graph property Π is characterized by $c_\Pi \in \mathbb{N}$ adjacencies if for all graphs $G \in \Pi$, for every vertex $v \in V(G)$, there is a set $D \subseteq V(G) \setminus \{v\}$ of size at most c_Π such that all graphs G' which are obtained from G by adding or removing edges between v and vertices in $V(G) \setminus D$, are also contained in Π .

As an example, the graph property “having a chordless cycle of length at least 4” is characterized by 3 adjacencies. The graph property “not being an interval graph” is not characterized by a finite number of adjacencies. Other examples are given by Fomin et al. [13].

Any finite graph property Π is trivially characterized by $\max_{G \in \Pi} |V(G)| - 1$ adjacencies. We state the following easily verified fact without proof.

► **Proposition 3.** *Let Π' be the set of all graphs that contain a graph from a finite set Π as induced subgraph. Then Π' is characterized by $\max_{G \in \Pi} |V(G)| - 1$ adjacencies.*

A graph G is vertex-minimal with respect to Π if $G \in \Pi$ and for all $S \subsetneq V(G)$ the graph $G[S]$ is not contained in Π . The following framework can be used to get polynomial kernels for the Π -FREE DELETION problem parameterized by vertex cover.

► **Theorem 4** (Theorem 2, [13]). *If Π is a graph property such that:*

- (i) Π is characterized by c_Π adjacencies,
- (ii) every graph in Π contains at least one edge, and
- (iii) there is a non-decreasing polynomial $p: \mathbb{N} \rightarrow \mathbb{N}$ such that all graphs G that are vertex-minimal with respect to Π satisfy $|V(G)| \leq p(\text{vc}(G))$,

then Π -FREE DELETION parameterized by the vertex cover size x admits a polynomial kernel with $\mathcal{O}((x + p(x))x^{c_\Pi})$ vertices.

3 Framework based on low-rank adjacencies

3.1 Incidence vectors and characterizations

As a first step towards our kernelization framework for Π -FREE DELETION, we introduce an incidence vector definition (inc) that characterizes the neighborhood of a given vertex. Compared to the vector encoding used by Agrawal et al. [1] for INTERVAL DELETION, our vector definition differs because it supports arbitrarily large subsets (they consider subsets of size at most two), and because an entry of a vector simultaneously prescribes which neighbors should be present, and which neighbors should *not* be present.

► **Definition 5** (c -incidence vector). *Let G be a graph with vertex cover X and let $c \in \mathbb{N}$. Let $Q', R' \subseteq X$ such that $|Q'| + |R'| \leq c$. We define the c -incidence vector $\text{inc}_{(G,X)}^{c,(Q',R')}(u)$ for a vertex $u \in V(G) \setminus X$ as a vector over \mathbb{F}_2 that has an entry for each $(Q, R) \in X \times X$ with $Q \cap R = \emptyset$ such that $|Q| + |R| \leq c$, $Q' \subseteq Q$ and $R' \subseteq R$. It is defined as follows:*

$$\text{inc}_{(G,X)}^{c,(Q',R')}(u)[Q, R] = \begin{cases} 1 & \text{if } N_G(u) \cap Q = \emptyset \text{ and } R \subseteq N_G(u), \\ 0 & \text{otherwise.} \end{cases}$$

We drop superscript (Q', R') if both Q' and R' are empty sets. The intuition behind the superscript (Q', R') is that it projects the entries of the full incidence vector $\text{inc}_{(G,X)}^c$ to those for supersets of Q', R' . The c -incidence vectors can be naturally summed coordinate-wise. For ease of presentation we do not define an explicit order on the coordinates of the vector, as any arbitrary but fixed ordering suffices.

If the sum of some vectors equals some other vector with respect to a certain graph G , then this equality is preserved when decreasing c or taking induced subgraphs of G .

► **Proposition 6.** *Let G be a graph with vertex cover X , let $c \in \mathbb{N}$, and let $D \subseteq V(G)$ be disjoint from X . If $v \in V(G) \setminus (D \cup X)$ and $\text{inc}_{(G,X)}^c(v) = \sum_{u \in D} \text{inc}_{(G,X)}^c(u)$, then*

- $\text{inc}_{(G,X)}^{c'}(v) = \sum_{u \in D} \text{inc}_{(G,X)}^{c'}(u)$ for any $c' \leq c$, and
- $\text{inc}_{(H,X \cap V(H))}^c(v) = \sum_{u \in D} \text{inc}_{(H,X \cap V(H))}^c(u)$ for any induced subgraph H of G that contains D and v .

Proof. For the first point, observe that for any vertex $v \notin (D \cup X)$, the vector $\text{inc}_{(G,X)}^{c'}(v)$ is simply a projection of $\text{inc}_{(G,X)}^c(v)$ to a subset of its coordinates. Hence if the complete vector of v is equal to the sum of the complete vectors of $u \in D$, then projecting the vector of both v and of the sum to the same set of coordinates, yields identical vectors.

For the second point, observe that since X is a vertex cover of G , we have $N_G(v) \subseteq X$ for all $v \in V(G) \setminus X$. Moreover, if H is an induced subgraph of G containing D and v , then $X_H := X \cap V(H)$ is a vertex cover of H . Hence for any $u \in V(H) \setminus X_H$ the c -incidence vector $\text{inc}_{(H, X \cap V(H))}^c(u)$ is well-defined. If Q, R are disjoint sets for which $\text{inc}_{(H, X \cap V(H))}^c(u)[Q, R]$ is defined, then $Q, R \subseteq X_H$, so the adjacencies between u and $Q \cup R$ in the induced subgraph H are identical to those in G , which implies $\text{inc}_{(G, X)}^c(u)[Q, R] = \text{inc}_{(H, X \cap V(H))}^c(u)[Q, R]$. Hence when we replace a c -incidence vector with subscript (G, X) by a vector with subscript $(H, X \cap V(H))$, we essentially project the vector to a subset of its coordinates without changing any values. For the same reason as above, this preserves the fact that the vectors of D sum to that of v . \blacktriangleleft

We are ready to introduce the main definition, namely characterization of a graph property Π by rank- c adjacencies for some $c \in \mathbb{N}$. In our framework, this replaces characterization by c adjacencies in the framework of Fomin et al. [13] (Theorem 4).

► Definition 7 (rank- c adjacencies). *Let $c \in \mathbb{N}$ be a natural number. Graph property Π is characterized by rank- c adjacencies if the following holds. For each graph H , for each vertex cover X of H , for each set $D \subseteq V(H) \setminus X$, for each $v \in V(H) \setminus (D \cup X)$, if*

■ $H - D \in \Pi$, and

■ $\text{inc}_{(H, X)}^c(v) = \sum_{u \in D} \text{inc}_{(H, X)}^c(u)$ when evaluated over \mathbb{F}_2 ,

then there exists $D' \subseteq D$ such that $H - v - (D \setminus D') \in \Pi$. If there always exists such set D' of size 1, then we say Π is characterized by rank- c adjacencies with singleton replacements.

Intuitively, the definition demands that if we have a set D such that $H - D \in \Pi$, and the c -incidence vectors of D sum to the vector of some vertex v over \mathbb{F}_2 , then there exists $D' \subseteq D$ such that removing v from $H - D$ and adding back D' results in a graph that is still contained in Π . For example, in Section 4.1 we show that the graph property “containing an odd hole or odd-anti-hole” is characterized by rank-4 adjacencies. Using our framework, this leads to a polynomial kernel for PERFECT DELETION parameterized by vertex cover. Other examples of graph properties which are characterized by a rank- c adjacencies for some $c \in \mathcal{O}(1)$ include “containing a cycle” and “being wheel-free”. On the other hand, we will show in Theorem 25 that the property “containing an induced wheel whose size is 3 or at least 5” cannot be characterized by rank- c adjacencies for any finite c .

3.2 A generic kernelization

Our kernelization framework for Π -FREE DELETION relies on a single reduction rule presented in Algorithm 1. It assigns an incidence vector to every vertex outside the vertex cover and uses linear algebra to select vertices to store in the kernel. Let us therefore recall the relevant algebraic background. A *basis* of a set S of d -dimensional vectors over a field \mathbb{F} is a minimum-size subset $B \subseteq S$ such that all $\mathbf{v} \in S$ can be expressed as linear combinations of elements of B , i.e., $\mathbf{v} = \sum_{\mathbf{u} \in B} \alpha_{\mathbf{u}} \cdot \mathbf{u}$ for a suitable choice of coefficients $\alpha_{\mathbf{u}} \in \mathbb{F}$. When working over the field \mathbb{F}_2 , the only possible coefficients are 0 and 1, which gives a basis B of S the stronger property that any vector $\mathbf{v} \in S$ can be written as $\sum_{\mathbf{u} \in B'} \mathbf{u}$, where $B' \subseteq B$ consists of those vectors which get a coefficient of 1 in the linear combination.

Our reduction algorithm repeatedly computes a basis of the incidence vectors of the remaining set of vertices, and stores the vertices corresponding to the basis in the kernel.

■ **Algorithm 1** Reduce (Graph G , vertex cover X of G , $\ell \in \mathbb{N}$, $c \in \mathbb{N}$).

```

1: Let  $Y_1 := V(G) \setminus X$ .
2: for  $i \leftarrow 1$  to  $\ell$  do
3:   Let  $V_i = \{\text{inc}_{(G,X)}^c(y) \mid y \in Y_i\}$  and compute a basis  $B_i$  of  $V_i$  over  $\mathbb{F}_2$ .
4:   For each  $\mathbf{v} \in B_i$ , choose a unique vertex  $y_{\mathbf{v}} \in Y_i$  such that  $\mathbf{v} = \text{inc}_{(G,X)}^c(y_{\mathbf{v}})$ .
5:   Let  $A_i := \{y_{\mathbf{v}} \mid \mathbf{v} \in B_i\}$  and  $Y_{i+1} = Y_i \setminus A_i$ .
6: end for
7: return  $G[X \cup \bigcup_{i=1}^{\ell} A_i]$ 

```

► **Proposition 8.** For a fixed $c \in \mathbb{N}$, Algorithm 1 runs in polynomial time in terms of ℓ and the size of the graph, and returns a graph on $\mathcal{O}(|X| + \ell \cdot |X|^c)$ vertices.

Proof. Observe that for each i , the vectors in V_i have at most $2^c \cdot \binom{|X|}{\leq c} = \mathcal{O}(|X|^c)$ entries and therefore the rank of the vector space is $\mathcal{O}(|X|^c)$. Hence each computed basis contains $\mathcal{O}(|X|^c)$ vectors. For constant c , this means that each basis can be computed in polynomial time using Gaussian elimination. The remaining operations can be done in polynomial time in terms of ℓ and the size of the graph. Since $|A_i| \in \mathcal{O}(|X|^c)$ for each $i \in [\ell]$, the resulting graph has $\mathcal{O}(|X| + \ell \cdot |X|^c)$ vertices. ◀

► **Theorem 9.** If Π is a graph property such that:

- (i) Π is characterized by rank- c adjacencies,
- (ii) every graph in Π contains at least one edge, and
- (iii) there is a non-decreasing polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ such that all graphs G that are vertex-minimal with respect to Π satisfy $|V(G)| \leq p(\text{vc}(G))$,

then Π -FREE DELETION parameterized by the vertex cover size x admits a polynomial kernel on $\mathcal{O}((x + p(x)) \cdot x^c)$ vertices.

Proof. Consider an instance (G, X, k) of Π -FREE DELETION. Note that if $k \geq |X|$, then we can delete the entire vertex cover to get an edgeless graph, which is Π -free by (ii), and therefore we may output a constant size YES-instance as the kernel. If $k < |X|$, let G' be the graph obtained by the procedure $\text{REDUCE}(G, X, \ell := k + 1 + p(|X|), c)$. By Proposition 8 this can be done in polynomial time and the resulting graph contains $\mathcal{O}(|X| + p(|X|)) \cdot |X|^c$ vertices. All that is left to show is that the instance (G', X, k) is equivalent to the original instance. Since G' is an induced subgraph of G , it follows that if (G, X, k) is a YES-instance, then so is (G', X, k) . In the other direction, suppose that (G', X, k) is a YES-instance with solution S . We show that S also is a solution for the original instance.

For the sake of contradiction assume that this is not the case. Then the graph $G - S$ contains an induced subgraph that belongs to Π . Let P be a minimal set of vertices of $G - S$ for which $G[P] \in \Pi$ and that minimizes $|P \setminus V(G')|$. Since S is a solution for (G', X, k) , it follows that there exists a vertex $v \in P \setminus V(G')$. Moreover we have that $v \notin X$, since the graph G' returned by Algorithm 1 contains all vertices of X . The set $P \cap X$ is a vertex cover for $G[P]$, therefore by property (iii) we have that $|P| \leq p(\text{vc}(G[P])) \leq p(|X|)$. Since the vertex sets A_1, \dots, A_{ℓ} computed in the REDUCE operation are disjoint, and since $|S| \leq k$, it follows that there exists an $i \in [k + 1 + p(|X|)]$ such that the set of vertices A_i corresponding to basis B_i is disjoint from both S and P .

As $v \notin V(G')$ implies $v \notin \bigcup_{i=1}^{\ell} A_i$, in each iteration of line 3 the vectors of the computed vertex set A_i span the vector of v . Hence, since we work over \mathbb{F}_2 , there exists $D \subseteq A_i \subseteq V(G')$ such that $\text{inc}_{(G,X)}^c(v) = \sum_{u \in D} \text{inc}_{(G,X)}^c(u)$. Consider the graph $H := G[P \cup D]$. Since H is an induced subgraph that includes D and D is disjoint from X , by Proposition 6 it follows that $\text{inc}_{(H, X \cap V(H))}^c(v) = \sum_{u \in D} \text{inc}_{(H, X \cap V(H))}^c(u)$. Moreover $H - D \in \Pi$ as $H - D = G[P]$.

By the definition of rank- c adjacencies it follows that there exists $D' \subseteq D$ such that $P' = H - v - (D \setminus D') \in \Pi$. But since $|P' \setminus V(G')| < |P \setminus V(G')|$, this contradicts the minimality of P . Therefore S must be a solution for the original instance. \blacktriangleleft

3.3 Properties of low-rank adjacencies

In this section we present several technical lemmata dealing with low-rank adjacencies. These will be useful when applying the framework to various graph properties. The next lemma shows that if Π is characterized by low-rank adjacencies with singleton replacements, then the edge-complement graphs are as well.

► **Lemma 10.** *Let Π be a graph property that is characterized by rank- c adjacencies with singleton replacements. Let $\bar{\Pi}$ be the graph property such that $G \in \Pi$ if and only if $\bar{G} \in \bar{\Pi}$. Then $\bar{\Pi}$ is characterized by rank- c adjacencies with singleton replacements.*

Proof. Let H be a graph with vertex cover X . Let $D \subseteq V(H) \setminus X$ be a set such that $H - D \in \bar{\Pi}$. Consider some vertex $v \in V(H) \setminus (D \cup X)$ such that $\text{inc}_{(H,X)}^c(v) = \sum_{u \in D} \text{inc}_{(H,X)}^c(u)$. Let $X' = V(H) \setminus (D \cup \{v\})$.

▷ **Claim 11.** We have $\text{inc}_{(H,X')}^c(v) = \sum_{u \in D} \text{inc}_{(H,X')}^c(u)$.

Proof. Since vertices outside X are independent, neither v nor any vertex in D is adjacent to any vertex in $X' \setminus X$. So for any disjoint $Q, R \subseteq X'$ with $R \cap (X' \setminus X) \neq \emptyset$ we have $\text{inc}_{(H,X')}^c(v)[Q, R] = \text{inc}_{(H,X')}^c(u)[Q, R] = 0$ for all $u \in D$ by definition, while for $R \cap (X' \setminus X) = \emptyset$ we have $\text{inc}_{(H,X')}^c(u)[Q, R] = \text{inc}_{(H,X)}^c(u)[Q \cap X, R]$ for any $u \in D \cup \{v\}$. \blacktriangleleft

Let H' be obtained from H by (1) taking the edge complement, and then (2) turning $H'[D \cup \{v\}]$ back into an independent set (the complement made it a clique). Note that X' is a vertex cover of H' .

▷ **Claim 12.** We have $\text{inc}_{(H',X')}^c(v) = \sum_{u \in D} \text{inc}_{(H',X')}^c(u)$.

Proof. Immediate from Claim 11 since $\text{inc}_{(H',X')}^c(u)[Q, R] = \text{inc}_{(H,X')}^c(u)[R, Q]$ for all $u \in D \cup \{v\}$. \blacktriangleleft

Observe that $H' - D$ is the edge-complement of $H - D$, so $H' - D \in \Pi$. Together with the previous claim, since Π is characterized by rank- c adjacencies with singleton replacements, it follows that there exists $v' \in D$ such that $G' := H' - v - (D \setminus \{v'\}) \in \Pi$. Since G' contains only a single vertex of $\{v\} \cup D$, none of its edges were edited during step (2) above, so that $G := H - v - (D \setminus \{v'\})$ is the edge-complement of G' , implying $G \in \bar{\Pi}$. This shows that $\bar{\Pi}$ is characterized by rank- c adjacencies with singleton replacements. \blacktriangleleft

Lemma 13 proves closure under taking the union of two characterized properties.

► **Lemma 13.** *Let Π and Π' be graph properties characterized by rank- c_Π and rank- $c_{\Pi'}$ adjacencies (with singleton replacements), respectively. Then the property $\Pi \cup \Pi'$ is characterized by rank- $\max(c_\Pi, c_{\Pi'})$ adjacencies (with singleton replacements).*

Proof. Consider a graph H with vertex cover X and set $D \subseteq V(H) \setminus X$ such that $H - D \in \Pi \cup \Pi'$. Let $v \in V(H) \setminus (D \cup X)$ be some vertex such that $\text{inc}_{(H,X)}^{\max(c_\Pi, c_{\Pi'})}(v) = \sum_{u \in D} \text{inc}_{(H,X)}^{\max(c_\Pi, c_{\Pi'})}(u)$. By Proposition 6, we have $\text{inc}_{(H,X)}^{c_\Pi}(v) = \sum_{u \in D} \text{inc}_{(H,X)}^{c_\Pi}(u)$. If $H - D \in \Pi$, then there exists $D' \subseteq D$ such that $H - v - (D \setminus D') \in \Pi$ and hence, $H - v - (D \setminus D') \in \Pi \cup \Pi'$ (in case of singleton replacements, D' is replaced by $\{v'\}$ for some $v' \in D$). The case $H - D \in \Pi'$ is symmetric. \blacktriangleleft

While the *intersection* of two graph properties which are characterized by a *finite number* of adjacencies is again characterized by a finite number of adjacencies [13, Proposition 4], the same does not hold for low-rank adjacencies; there is no analog of Lemma 13 for intersections.

In a graph G , we say that vertices u and v *share adjacencies* to a set S , if $N_G(u) \cap S = N_G(v) \cap S$. The following lemma states that when we have a set D whose c -incidence vectors sum to the vector of v , then for any set S of size up to c there exists a nonempty subset $D' \subseteq D$ whose members all share adjacencies with v to S .

► **Lemma 14.** *Let G be a graph with vertex cover X , let $D \subseteq V(G)$ be disjoint from X , and let $c \in \mathbb{N}$. Consider a vertex $v \in V(G) \setminus (D \cup X)$. If $\text{inc}_{(G,X)}^c(v) = \sum_{u \in D} \text{inc}_{(G,X)}^c(u)$, then for any set $S \subseteq V(G)$ with $|S| \leq c$ there exists $D' \subseteq D$, such that:*

- $|D'| \geq 1$ is odd,
- each vertex $u \in D'$ shares adjacencies with v to S , and
- $\text{inc}_{(G,X)}^{c,(Q',R')}(v) = \sum_{u \in D'} \text{inc}_{(G,X)}^{c,(Q',R')}(u)$, where $Q' = (S \setminus N_G(v)) \cap X$ and $R' = S \cap N_G(v)$.

Proof. For any vertex $d \in D$ that does not share adjacencies with v to S , the vector $\text{inc}_{(G,X)}^{c,(Q',R')}(d)$ is the vector containing only zeros. Let $D' \subseteq D$ be the set of vertices that do share adjacencies with v to S . Clearly $\text{inc}_{(G,X)}^{c,(Q',R')}(v) = \sum_{u \in D'} \text{inc}_{(G,X)}^{c,(Q',R')}(u)$, as removing all-zero vectors does not change the sum. Since $\text{inc}_{(G,X)}^{c,(Q',R')}(v)[Q', R'] = 1$ and $\text{inc}_{(G,X)}^{c,(Q',R')}(u)[Q', R'] = 1$ for all $u \in D'$, $|D'| \geq 1$ must be odd. ◀

Our framework adapts Theorem 4 by replacing characterization by c adjacencies by rank- c adjacencies. From the following statement we can conclude that our framework extends Theorem 4.

► **Lemma 15.** *A graph property Π characterized by c adjacencies is also characterized by rank- c adjacencies with singleton replacements.*

Proof. Let Π be a graph property characterized by c adjacencies. We show that Π is characterized by rank- c adjacencies. Let G be a graph with vertex cover X and $D \subseteq V(G) \setminus X$ be a set such that $G - D \in \Pi$. Let $v \in V(G) \setminus (D \cup X)$ be a vertex such that $\text{inc}_{(G,X)}^c(v) = \sum_{u \in D} \text{inc}_{(G,X)}^c(u)$.

Since Π is characterized by c adjacencies, there exists a set B of size at most c such that all graphs obtained by changing adjacencies between v and $V(G) \setminus B$ are also contained in Π . By Lemma 14 there exists $w \in D$ that shares adjacencies with v to B . Now consider the graph $G - v - (D \setminus \{w\})$. This graph is isomorphic to $G - D$ where w is matched to v and the adjacencies between v and $V(G) \setminus B$ are changed. But then by the definition of characterization by c adjacencies it follows that $G - v - (D \setminus \{w\}) \in \Pi$. ◀

4 Using the framework

In this section we give some results using our framework, which are listed in Table 1. We give polynomial kernels for PERFECT DELETION, AT-FREE DELETION, INTERVAL DELETION, EVEN-HOLE-FREE DELETION, and WHEEL-FREE DELETION parameterized by vertex cover.

4.1 Perfect Deletion

Let Π_P be the set of graphs that contain an odd hole or an odd anti-hole. The Π_P -FREE DELETION problem is known as the PERFECT DELETION problem. It was mentioned as an open question by Fomin et al. [13], since one can show that Π_P is not characterized by a

finite number of adjacencies. In this section we show that Π_P is characterized by rank-4 adjacencies with singleton replacements. Following this result, we show that it admits a polynomial kernel using Theorem 9. First, we give a lemma that will be helpful in the proof later on. We say that a vertex *sees an edge* if it is adjacent to both of its endpoints.

► **Lemma 16.** *Let G be a graph, $P = (v_1, \dots, v_n)$ where $n \geq 4$ is even be an induced path in G , and let y be a vertex not on P that is adjacent to both endpoints of P and sees an even number of edges of P . Then $G[V(P) \cup \{y\}]$ contains an odd hole as induced subgraph.*

Proof. We prove the claim by induction on n . Consider the case that $n = 4$. If y would be adjacent to exactly one of v_2 or v_3 , then y would see a single edge $\{v_1, v_2\}$ or $\{v_3, v_4\}$ respectively. If y would be adjacent to both v_2 and v_3 , then y would see all three edges of P . Since y sees an even number of edges of P , it follows that y is only adjacent to v_1 and v_4 . Then $G[V(P) \cup \{y\}]$ induces an odd hole.

In the remaining case we assume that the claim holds for $n' < n$, where $n \geq 6$ and both n' and n are even. Suppose that y sees both edges $\{v_1, v_2\}$ and $\{v_{n-1}, v_n\}$, then $P' = (v_2, \dots, v_{n-1})$ is an induced path on an even number of vertices such that y is adjacent to both of its endpoints and y sees an even number edges in P' . By the induction hypothesis $G[V(P') \cup \{y\}]$ contains an odd hole, therefore $G[V(P) \cup \{y\}]$ contains an odd hole as well. If y does not see both $\{v_1, v_2\}$ and $\{v_{n-1}, v_n\}$, then assume without loss of generality that y does not see the last edge $\{v_{n-1}, v_n\}$. Let v_j for $1 \leq j < n - 1$ be the largest index before n for which y is adjacent to v_j . If j is odd, then $G[\{v_j, \dots, v_n, y\}]$ induces an odd hole. Otherwise $P' = (v_1, \dots, v_j)$ is an induced path on an even number of vertices, y is adjacent to both of its endpoints, and y sees an even number of edges in P' ; hence the induction hypothesis applies. In all cases we get that $G[V(P) \cup \{y\}]$ contains an odd hole. ◀

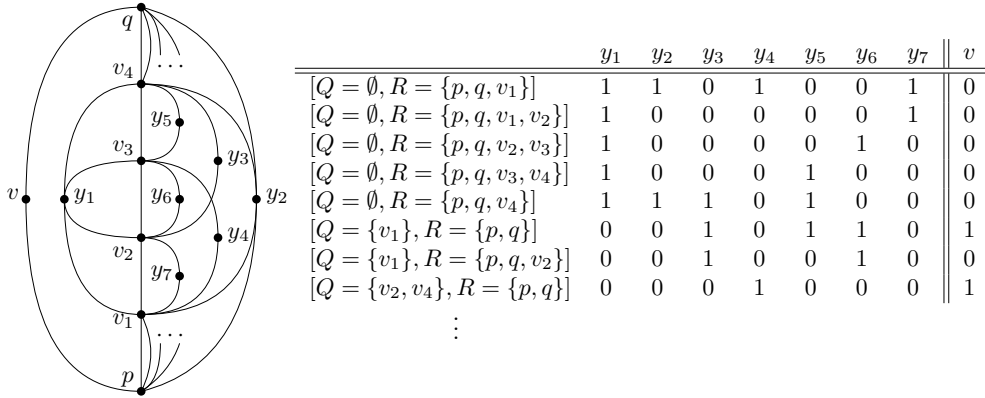
Before we show the proof that Π_P is characterized by rank-4 adjacencies with singleton replacements, we give some intuition for the replacement argument. Suppose we want to replace a vertex v of some odd hole, and we have a set D where each vertex in D is adjacent to both neighbors of v in the hole. Furthermore, the 4-incidence vectors of D sum to the vector of v . Then there must exist some vertex in D that sees an even number of edges of the induced path between the neighbors of v . This together with Lemma 16 would result in a graph that contains an odd hole. Figure 1 adds to this intuition.

Let Π_{OH} be the set of graphs that contain an odd hole. We show that Π_{OH} is characterized by rank-4 adjacencies with singleton replacements.

► **Theorem 17.** *Π_{OH} is characterized by rank-4 adjacencies with singleton replacements.*

Proof. Consider some graph H with vertex cover X and let $D \subseteq V(H) \setminus X$ such that $H - D \in \Pi_{OH}$. Let v be an arbitrary vertex in $V(H) \setminus (D \cup X)$ such that $\text{inc}_{(H,X)}^4(v) = \sum_{u \in D} \text{inc}_{(H,X)}^4(u)$. We show that $H - v - (D \setminus \{v'\}) \in \Pi_{OH}$ for some $v' \in D$.

Let C be an odd hole in $H - D$. If $v \notin V(C)$, then for every $v' \in D$ we have $H - v - (D \setminus \{v'\}) \in \Pi_{OH}$. So suppose that $v \in C$. Let $C = (v, p, v_1, \dots, v_{n-3}, q)$, where $|V(C)| = n$. Consider the induced path $P = (p, v_1, \dots, v_{n-3}, q)$. We have that v is adjacent to p and q . Let $D' \subseteq D$ be a set that shares adjacencies with v to $\{p, q\}$ such that $|D'| \geq 1$ is odd and $\text{inc}_{(H,X)}^{4,(\emptyset, \{p,q\})}(v) = \sum_{u \in D'} \text{inc}_{(H,X)}^{4,(\emptyset, \{p,q\})}(u)$. Such set exists by Lemma 14. Since C is an odd hole, $|V(P)|$ is even. Hence by Lemma 16, $G[V(P) \cup \{u\}]$ contains an odd hole if there exists some $u \in D'$ that sees an even number of edges of P . Suppose for the sake of contradiction that every vertex in D' sees an odd number of edges of P . Let E_u be the set of edges in P that are seen by $u \in D'$. Then $\sum_{u \in D'} |E_u|$ is odd as it is a sum of an odd number of odd numbers. Let $D'_{\{u,w\}} \subseteq D'$ be the set of vertices that see edge $\{u, w\} \in E(P)$. In order



■ **Figure 1** Graph G with vertex cover $X = \{p, q, v_1, \dots, v_4\}$, containing an odd hole $H = \{v, p, v_1, \dots, v_4, q\}$, such that $\text{inc}_{(G,X)}^{4,(\emptyset, \{p,q\})}(v) = \sum_{i=1}^7 \text{inc}_{(G,X)}^{4,(\emptyset, \{p,q\})}(y_i)$. All edges $\{p, y_i\}$ and $\{q, y_i\}$ for $i \in [7]$ exist, but not all are drawn. The table shows entries of the vectors $\text{inc}_{(G,X)}^{4,(\emptyset, \{p,q\})}(u \notin X)$. Vertex y_2 sees an even number of edges ($\{p, v_1\}$ and $\{v_4, q\}$), and (v_1, \dots, v_4, y_2) is an odd hole.

to satisfy $\sum_{u \in D'} \text{inc}^{(\emptyset, \{p,q\})}(u)[\emptyset, \{p, q, u, w\}] = \text{inc}^{(\emptyset, \{p,q\})}(v)[\emptyset, \{p, q, u, w\}] = 0$ over \mathbb{F}_2 , for $\{u, w\} \in E(P)$, we require $|D'_{\{u,w\}}|$ to be even. But then $\sum_{e \in E(P)} |D'_e| = \sum_{u \in D'} |E_u|$ would also need to be an even number. This contradicts the fact that $\sum_{u \in D'} |E_u|$ is odd. Therefore there must exist some $u \in D'$ that sees an even number of edges in P . ◀

Let Π_{OAH} be the set of graphs that contain an odd anti-hole. Then $\Pi_P = \Pi_{OH} \cup \Pi_{OAH}$. From applications of Lemma 10 and Lemma 13 we get the following.

► **Corollary 18.** *Graph properties Π_{OH} , Π_{OAH} , and Π_P are characterized by rank-4 adjacencies with singleton replacements.*

► **Theorem 19.** *PERFECT DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^5)$ vertices.*

Proof. By Corollary 18 we have that Π_P is characterized by rank-4 adjacencies with singleton replacements. Each graph in Π_P contains at least one edge. For each odd hole or odd anti-hole H , we have $|V(H)| \leq 2 \cdot \text{vc}(H)$. Therefore by Theorem 9 it follows that Π_P -FREE DELETION and hence PERFECT DELETION parameterized by vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^5)$ vertices. ◀

A variation of Theorem 17 presented in the full version [21] shows that the set Π_{EH} of graphs containing an even hole are characterized by rank-3 adjacencies, which leads to a kernel for EVEN-HOLE-FREE DELETION parameterized by the size of a vertex cover of $\mathcal{O}(|X|^4)$ vertices.

4.2 AT-free Deletion

In his dissertation, Köhler [22] gives a forbidden subgraph characterization of graphs without asteroidal triples. This forbidden subgraph characterization consists of 15 small graphs on 6 or 7 vertices each, chordless cycles of length at least 6, and three infinite families often called asteroidal witnesses. Let Π_{AT} be the set of graphs that contain an asteroidal triple. A technical case analysis leads to the following results.

► **Theorem 20 (★).** *Π_{AT} is characterized by rank-8 adjacencies with singleton replacements.*

► **Theorem 21.** *AT-FREE DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^9)$ vertices.*

Proof. Every graph in Π_{AT} contains at least one edge. By Theorem 20 it follows that Π_{AT} is characterized by rank-8 adjacencies with singleton replacements. Each small graph has at most 7 vertices. For each cycle C , we have $|V(C)| \leq 2 \cdot \text{vc}(C)$. Finally each asteroidal witness consists of an induced path with 2 or 3 additional vertices, hence there exists $c \in \mathbb{N}$ such that for $G \in \Pi_{AT}$, $|V(G)| \leq c \cdot \text{vc}(G)$. Hence by Theorem 9, AT-FREE DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^9)$ vertices. ◀

4.3 Interval Deletion

INTERVAL DELETION does not fit in the framework of Fomin et al. [13], since one can show that its forbidden subgraph characterization is not characterized by a finite number of adjacencies. It was shown to admit a polynomial kernel by Agrawal et al. [1]. We show that our framework captures this result. Consider the graph property $\Pi_{IV} = \Pi_{AT} \cup \Pi_{C_{\geq 4}}$, where Π_{AT} is the set of graphs that contain an asteroidal triple as in Section 4.2 and $\Pi_{C_{\geq 4}}$ is the set of graphs that contain an induced cycle of length at least 4. Making a graph Π_{IV} -free makes it chordal and AT-free, therefore Π_{IV} -FREE DELETION corresponds to INTERVAL DELETION.

► **Theorem 22.** *INTERVAL DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^9)$ vertices.*

Proof. Every graph in Π_{IV} contains at least one edge. By Theorem 20, Π_{AT} is characterized by rank-8 adjacencies. Furthermore, $\Pi_{C_{\geq 4}}$ is characterized by 3 adjacencies as shown by Fomin et al. [13, Proposition 3], and therefore by Lemma 15 also by rank-3 adjacencies. Therefore by Lemma 13, it follows that Π_{IV} is also characterized by rank-8 adjacencies. Each vertex minimal graph in $\Pi_{C_{\geq 4}}$ is a cycle C , for which we have $|V(C)| \leq 2 \cdot \text{vc}(C)$. Recall that $\Pi_{AT} = \Pi_S \cup \Pi_{C_{\geq 6}} \cup \Pi_{AW}$. Each vertex minimal graph in Π_S contains at most 7 vertices. Finally each asteroidal witness consists of an induced path with 2 or 3 additional vertices, hence there exists $c \in \mathbb{N}$ such that for $G \in \Pi_{IV}$, $|V(G)| \leq c \cdot \text{vc}(G)$. Therefore by Theorem 9, INTERVAL DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^9)$ vertices. ◀

4.4 (Almost) Wheel-free Deletion

Let $\Pi_{W_{\geq 3}}$ be the set of graphs that contain a wheel of size at least 3 as induced subgraph. Then WHEEL-FREE DELETION corresponds to $\Pi_{W_{\geq 3}}$ -FREE DELETION. We present a characterization by rank-4 adjacencies.

► **Theorem 23 (★).** *$\Pi_{W_{\geq 3}}$ is characterized by rank-4 adjacencies.*

Every graph that contains a wheel contains at least one edge. For every wheel W_n , we have $|V(W_n)| \leq 2 \cdot \text{vc}(W_n)$. Therefore by Theorem 9 we obtain:

► **Theorem 24.** *WHEEL-FREE DELETION parameterized by the size of a vertex cover admits a polynomial kernel on $\mathcal{O}(|X|^5)$ vertices.*

It turns out that this good algorithmic behavior is very fragile. Let $\Pi_{W_{\neq 4}}$ be the set of graphs that contain a wheel of size 3, or at least 5. Then $\Pi_{W_{\neq 4}}$ -FREE DELETION corresponds to ALMOST WHEEL-FREE DELETION. While $\Pi_{W_{\geq 3}}$ can be characterized by rank-4 adjacencies, the following shows that $\Pi_{W_{\neq 4}}$ is not characterized by adjacencies of any finite rank, and therefore does not fall within the scope of our kernelization framework.

► **Theorem 25 (★).** $\Pi_{W \neq 4}$ is not characterized by rank- c adjacencies for any $c \in \mathbb{N}$.

This is not a deficiency of our framework; we prove that the problem does not have any polynomial compression, and therefore no polynomial kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

► **Theorem 26 (★).** *ALMOST WHEEL-FREE DELETION* parameterized by vertex cover does not admit a polynomial compression unless $\text{coNP} \subseteq \text{NP}/\text{poly}$.

This suggests that the condition of being characterized by low-rank adjacencies is the right way to capture kernelization complexity.

5 Conclusion

We have presented a framework that can be used to obtain polynomial kernels for the Π -FREE DELETION problem parameterized by the size of a vertex cover, based on the novel concept of characterizations by low-rank adjacencies. Our framework significantly extends the scope of the earlier framework of Fomin et al. [13]. In addition to the examples given in Table 1, the framework can be applied to obtain kernels for a wide range of vertex-deletion problems. Using the fact that graph properties characterized by low-rank adjacencies are closed under taking a union (Lemma 13), together with the characterizations by low-rank adjacencies developed here, and characterizations by few adjacencies by Fomin et al. [13, Table 1], we obtain the following.

► **Corollary 27.** *Let \mathcal{F} be a hereditary graph class defined by an arbitrary combination of the following properties: being wheel-free, being odd-hole-free, being odd-anti-hole-free, being even-hole-free, being AT-free, being bipartite, being $C_{\geq c}$ -free for some fixed $c \in \mathbb{N}$, being H -minor-free for some fixed graph H , being H -free for some fixed graph H containing at least one edge, and having a Hamiltonian cycle (respectively, path). Then the problem of testing whether an input graph G can be turned into a member of \mathcal{F} by removing at most k vertices, has a polynomial kernel parameterized by vertex cover.*

It would be interesting to see whether the exponents given by Table 1 are tight (cf. [15]).

References

- 1 Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1711–1730, 2019. URL: <http://dl.acm.org/citation.cfm?id=3310435.3310538>.
- 2 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.
- 3 H.L. Bodlaender, B.M.P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.
- 4 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.
- 5 A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999. doi:10.1137/1.9780898719796.
- 6 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly(k)

- kernel. In Hans L. Bodlaender and Michael A. Langston, editors, *Proceedings of the 2nd Second International Workshop on Parameterized and Exact Computation, IWPEC 2006*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2006. doi:10.1007/11847250_18.
- 7 Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye. A polynomial kernel for diamond-free editing. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.10.
 - 8 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Ann. Math. (2)*, 164(1):51–229, 2006. doi:10.4007/annals.2006.164.51.
 - 9 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
 - 10 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Erik Jan van Leeuwen, and Marcin Wrochna. Polynomial kernelization for removing induced claws and diamonds. *Theory Comput. Syst.*, 60(4):615–636, 2017. doi:10.1007/s00224-016-9689-x.
 - 11 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.
 - 12 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013. doi:10.1016/j.ejc.2012.04.008.
 - 13 Fedor V. Fomin, Bart M.P. Jansen, and Michał Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *Journal of Computer and System Sciences*, 80(2):468–495, 2014. doi:10.1016/j.jcss.2013.09.004.
 - 14 Fedor V. Fomin, Daniel Lokshantov, Neeldhara Misra, and Saket Saurabh. Planar \mathcal{F} -deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.
 - 15 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshantov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Trans. Algorithms*, 13(3):35:1–35:35, 2017. doi:10.1145/3029051.
 - 16 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.
 - 17 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.
 - 18 Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Inf. Comput.*, 231:70–88, 2013. doi:10.1016/j.ic.2013.08.005.
 - 19 Bart M. P. Jansen and Astrid Pieterse. Optimal data reduction for graph coloring using low-degree polynomials. *Algorithmica*, 81(10):3865–3889, 2019. doi:10.1007/s00453-019-00578-5.
 - 20 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discrete Math.*, 32(3):2258–2301, 2018. doi:10.1137/17M112035X.
 - 21 Bart M.P. Jansen and Jari J.H. de Kroon. Preprocessing vertex-deletion problems: Characterizing graph properties by low-rank adjacencies. *CoRR*, abs/2004.08818, 2020. arXiv:2004.08818.
 - 22 Ekkehard Köhler. *Graphs without asteroidal triples*. PhD thesis, TU Berlin, 1999.
 - 23 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

- 24 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.
- 25 Daniel Lokshtanov. Wheel-free deletion is $W[2]$ -hard. In Martin Grohe and Rolf Niedermeier, editors, *Proceedings of the Third International Workshop on Parameterized and Exact Computation, IWPEC 2008*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 2008. doi:10.1007/978-3-540-79723-4_14.
- 26 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. *Theor. Comput. Sci.*, 494:99–111, 2013. doi:10.1016/j.tcs.2013.01.029.
- 27 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

Locality Sensitive Hashing for Set-Queries, Motivated by Group Recommendations

Haim Kaplan

School of Computer Science, Tel Aviv University, Israel
haimk@tau.ac.il

Jay Tenenbaum

School of Computer Science, Tel Aviv University, Israel
jaytenenbaum@mail.tau.ac.il

Abstract

Locality Sensitive Hashing (LSH) is an effective method to index a set of points such that we can efficiently find the nearest neighbors of a query point. We extend this method to our novel Set-query LSH (SLSH), such that it can find the nearest neighbors of a set of points, given as a query.

Let $s(x, y)$ be the similarity between two points x and y . We define a similarity between a set Q and a point x by aggregating the similarities $s(p, x)$ for all $p \in Q$. For example, we can take $s(p, x)$ to be the angular similarity between p and x (i.e., $1 - \frac{\angle(x, p)}{\pi}$), and aggregate by arithmetic or geometric averaging, or taking the lowest similarity.

We develop locality sensitive hash families and data structures for a large set of such arithmetic and geometric averaging similarities, and analyze their collision probabilities. We also establish an analogous framework and hash families for distance functions. Specifically, we give a structure for the euclidean distance aggregated by either averaging or taking the maximum.

We leverage SLSH to solve a geometric extension of the approximate near neighbors problem. In this version, we consider a metric for which the unit ball is an ellipsoid and its orientation is specified with the query.

An important application that motivates our work is group recommendation systems. Such a system embeds movies and users in the same feature space, and the task of recommending a movie for a group to watch together, translates to a set-query Q using an appropriate similarity.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Data structures design and analysis; Information systems \rightarrow Information retrieval

Keywords and phrases Locality sensitive hashing, nearest neighbors, similarity search, group recommendations, distance functions, similarity functions, ellipsoid

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.28

Related Version A full version of the paper is available at <https://arxiv.org/abs/2004.07286>.

Funding This work was supported by ISF grant no. 1595/19 and GIF grant no. 1367.

Acknowledgements We want to thank Prof. Micha Sharir and Prof. Edith Cohen for the fruitful discussions.

1 Introduction

The focus of this paper is on similarity search for queries which are sets of points (set-queries), where we aim to efficiently retrieve points with a high aggregated similarity to the points of the set-query.



© Haim Kaplan and Jay Tenenbaum;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 28; pp. 28:1–28:18

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Efficient similarity search for massive databases is central in many application areas, such as recommendation systems, content-based image or audio retrieval, machine learning, pattern recognition, and data analysis. The database is often composed of high-dimensional feature vectors of documents, images, etc., and we are interested in finding the near neighbors of a query vector.

Traditional tree-based indexing mechanisms do not scale well to higher dimensions, a phenomenon known as the “curse of dimensionality”. To cope with this curse of dimensionality, Indyk and Motwani [11, 10] introduced Locality Sensitive Hashing (LSH), a framework based on hash functions for which the probability of hash collision is higher for similar points than for dissimilar points.

Using such hash functions, one can determine near neighbors by hashing the query point and retrieving the data points stored in its bucket. Typically, multiple LSH functions are concatenated to reduce false positives, and multiple hash tables are needed to reduce false negatives. This gives rise to a data structure which satisfies the following property: for any query point q , if there exists an S -similar data point to q in the database, it retrieves (with constant probability) some cS -similar data point to q for some constant $0 < c < 1$. This data structure is parameterized by a parameter $\rho = \frac{\log(p_1)}{\log(p_2)} < 1$, where p_1 is the minimal collision probability for any two points of similarity at least S , and p_2 is the maximal collision probability for any two points of similarity at most cS . The data structure can be built in time and space $O(1/p_1 \cdot n^{1+\rho})$, and its query time is $O(1/p_1 \cdot n^\rho \log_{1/p_2}(n))$.

Since the seminal paper of Indyk and Motwani [11, 10], many extensions have been considered for the LSH framework [16]. A notable extension is the work of Shrivastava and Li [22], which study the inner product similarity $ip\text{-sim}(x, y) = x^T y$. They find near neighbors for the inner product similarity by extending the LSH framework to allow asymmetric hashing schemes (ALSH) [20], in which we hash the query and the data points using different hash functions. There is also an analogous LSH framework for distance functions, based on hash functions for which the probability of hash collision is higher for near points than for far points. An important distance function to which the LSH framework has been applied is the ℓ_p distance [19]. Datar et al. [8] study the ℓ_p distance for $p \in (0, 2]$, and present a hash based on p -stable distributions. Andoni and Indyk [2] give a near-optimal (data oblivious) scheme for $p = 2$. Recently, several theoretically superior data dependent schemes have been designed [3, 4].

A noteworthy application of LSH is for *recommendation systems* [15], which are required to recommend points that are similar feature-wise to the user. *Group recommendation systems* [14, 17] are recommendation systems which provide recommendations, not only to an individual, but also to a whole group of people, and are gaining popularity in recent years. The need in such systems arises in many scenarios: when searching for a movie or a TV show for friends to watch together [21, 23], a travel destination for a family to spend a holiday break in [13, 18], or a good restaurant for a group of tourists to have lunch in [5]. In the literature of group recommendation systems, Jameson et al. [14] survey various techniques to aggregate individual user-point similarities s to a group-point similarity s^* . The most famous aggregation techniques are the *average similarity* which defines the aggregated similarity to be $s^*(Q, x) = \frac{1}{|Q|} \sum_{q \in Q} s(q, x)$, and the *center similarity* (sometimes called *Least-Misery*) which defines the aggregated similarity to be $s^*(Q, x) = \min_{q \in Q} (s(q, x))$.

Most of the work to date on group recommendations is experimental on relatively small data sets. In this paper we give (the first to the best of our knowledge) rigorous mathematical treatment of this problem using the LSH framework. LSH-based recommendation schemes are

used for individual recommendations but do not naturally support group recommendations. We extend LSH to support set-queries. We formalize this setting by introducing the notions of a set-query-to-point (s2p) similarity function, and of the novel *set-query LSH* (SLSH).

Our novel set-query LSH (SLSH) framework extends the LSH framework to similarities between a set of points and a point (s2p similarities). We define such a similarity between a set-query $Q = \{q_1, \dots, q_k\} \subset Z$ and a point $x \in Z$ by aggregating (e.g., averaging) point-to-point (p2p) similarities $(s(q_1, x), \dots, s(q_k, x))$ where $s : Z \times Z \rightarrow \mathbb{R}_{\geq 0}$ is a p2p similarity. Specifically, we consider the ℓ_p similarity $s_p(Q, x) = \frac{1}{k} \sum_{i=1}^k (s(q_i, x))^p$ for a constant $p \in \mathbb{N}$ (of which the *average similarity* $s_{avg}(Q, x) = s_1(Q, x)$ is a special case), the *geometric similarity* $s_{geo}(Q, x) = \prod_{i=1}^k s(q_i, x)$, and the *center similarity* $s_{cen}(Q, x) = \min_{q \in Q} s(q, x)$ of s .¹ Analogously, we can define s2p distance functions and SLSH framework for distances. We develop hash families for which the probability of collision between a set-query Q and a point x is higher when Q is similar to x than when Q is dissimilar to x .

Our contribution

We extend the LSH framework to a novel framework for handling set-queries (SLSH) for both distance and similarity functions, and study their set-query extensions. We develop various techniques for designing set-query LSH schemes, either by giving an SLSH family directly for the s2p similarity at hand, or by reducing the problem to a previously solved problem for a different distance or similarity.

Simple SLSH schemes via achievable p2p similarities. We say that a p2p similarity s is *achievable* if there exists a hash family such that the collision probability between x and y is exactly $s(x, y)$. The *angular*, *hamming* and *Jaccard* p2p similarities have this property. We show how to construct SLSH families for the ℓ_p and geometric s2p similarities that are obtained by aggregating a p2p similarity which is achievable.

Many of our SLSH families for s2p similarities can be extended to *weighted* s2p similarity functions, in which the contribution of each individual p2p similarity has a different weight. For example, define the weighted geometric s2p similarity (of a p2p similarity s) of a set-query Q and a data point x to be $s_{wgeo}(Q, x) = \prod_{i=1}^k (s(q_i, x))^{w_i}$. These weights are independent of the specific query and are given at preprocessing time. As an example, a solution for the SLSH problem for s_{wgeo} for any *achievable* p2p similarity s appears in Appendix A.2.

Additionally, we present an SLSH scheme for the average euclidean distance which is based upon the shrink-lift transformation (the “lift” refers to the lifting transformation from Bachrach et al. [6]) which approximately reduces euclidean distances to angular distances. We get an average angular distance problem which we then solve using the fact that the angular similarity is achievable and inversely related to the angular distance.²

¹ For ease of presenting our ideas, we define the s_p and center similarities to be the p 'th and k 'th power of their conventional definition in the literature. Note that the results follow for the conventional definitions since maximizing a similarity is equivalent to maximizing a constant power of it.

² We note that as the LSH approximation parameter c approaches 1, the required shrink approaches 0. This makes the angles between the lifted points small, which in turn deteriorates the performance of the angular similarity structure (in particular, one can show that the term in $\log_{1/p_2}(n)$ in the query time bound of the LSH structure approaches infinity). Therefore, we conclude that the shrink-lift transformation is useful for values of c which are not too close to 1. However, note that such a property holds for any LSH-based nearest neighbors algorithm, where for approximation ratios $c \rightarrow 1$, the performance becomes equivalent or worse than linear scan.

Ellipsoid ALSH. We define the novel *euclidean ellipsoid distance* which naturally extends the regular euclidean distance. We develop an LSH-based near neighbors structure for this distance by a reduction to an SLSH problem with respect to the geometric angular distance. Recall that in the euclidean approximate near neighbor problem, the query specifies the center of two concentric balls such that one is a scaled version of the other. Analogously, in our novel ellipsoid distance, the query specifies the center and orientation of two concentric ellipsoids such that one is a scaled version of the other. If there is a point in the small ellipsoid, we have to return a point in the large one. We reduce this problem to a novel angular ellipsoid distance counterpart via the shrink-lift transformation mentioned before. In this angular distance counterpart, the distance is a weighted sum of squared angles (rather than squared distances in the euclidean ellipsoid distance).

To solve the angular ellipsoid ALSH problem, we make a neat observation that the squared angle that a point creates in the direction of an angular ellipsoid axis, is inversely related to the collision probability of the point with the hyperplane perpendicular to the axis, in the ALSH family of Jain et al. [12]. This observation reduces the problem to a weighted geometric angular similarity SLSH problem, which we finally solve as indicated above using the fact that the angular similarity is achievable.

Center euclidean distance SLSH. The most challenging s2p distance is the center euclidean distance which wants to minimize the maximum distance from the points of the set-query. For this distance function, we obtain an SLSH scheme when the set-query is of size 2, via a reduction to the euclidean ellipsoid ALSH problem. This reduction is based on an observation that the points of center euclidean distance at most r to a set-query of size 2, approximately form an ellipsoid.

We focus on developing techniques to construct SLSH families, but we do not compute closed formulas for ρ as a function of S and c . These expressions can be easily derived for the simpler families but are more challenging to derive for the more complicated ones. We leave the optimization of ρ and testing the method on real recommendation data for future work.

Other related work

Since we study our novel SLSH framework, there is no direct previous work on this. That been said, there is related previous work on LSH, ALSH, and recommendation systems which are as follows. In the literature of recommendation systems, Koren and Volinsky [15] discuss matrix factorization models where user-item interactions are modeled as inner products, and Bachrach et al. [6] propose a transformation that reduces the inner product similarity to euclidean distances. Regarding group recommendation systems, Masthoff and Judith [17] show that humans care about fairness and avoiding individual misery when giving group recommendations, and Yahia et al. [1] formalize semantics that account for item relevance to a group, and disagreements among the group members. Regarding LSH and ALSH, Neyshabur and Srebro [20] study symmetric and asymmetric hashing schemes for the inner product similarity, and show a superior symmetric LSH to that of Shrivastava and Li [22], that uses the transformation of Bachrach et al. [6]. As stated before, we use the ALSH family of Jain et al. [12] to solve the angular ellipsoid ALSH problem. We show that this family can be interpreted as a private case of an SLSH family for an appropriate s2p similarity, however Jain et al. [12] did not need this property, and the connection is coincidental.

2 Preliminaries

We use the following standard definition of a *Locality Sensitive Hash Family (LSH)* with respect to a given point-to-point (p2p) similarity function $s : Z \times Z \rightarrow \mathbb{R}_{\geq 0}$.

► **Definition 1** (Locality Sensitive Hashing (LSH)). *Let $c < 1$, $S > 0$ and $p_1 > p_2$. A family H of functions $h : Z \rightarrow \Gamma$ is an (S, cS, p_1, p_2) -LSH for a p2p similarity function $s : Z \times Z \rightarrow \mathbb{R}_{\geq 0}$ if for any $x, y \in Z$,*

1. *If $s(x, y) \geq S$ then $\Pr_{h \in H}[h(x) = h(y)] \geq p_1$, and*
2. *If $s(x, y) \leq cS$ then $\Pr_{h \in H}[h(x) = h(y)] \leq p_2$.*

Note that in the definition above, and in all the following definitions, the hash family H is always sampled uniformly. Following Shrivastava and Li [22] we extend the LSH framework to asymmetric similarities $s : Z_1 \times Z_2 \rightarrow \mathbb{R}_{\geq 0}$ (where Z_1 is the domain of the data points and Z_2 is the domain of the queries). Here the (S, cS, p_1, p_2) -ALSH family H consists of pairs of functions $f : Z_1 \rightarrow \Gamma$ and $g : Z_2 \rightarrow \Gamma$, and the requirement is that $\Pr_{(f,g) \in H}[f(x) = g(y)] \geq p_1$ if $s(x, y) \geq S$, and $\Pr_{(f,g) \in H}[f(x) = g(y)] \leq p_2$ if $s(x, y) \leq cS$.

Set-Query LSH

A special kind of asymmetric similarities are similarities between a set of points and a point (s2p similarities). That is, similarities of the form $s^* : \mathcal{P}(Z, k) \times Z \rightarrow \mathbb{R}_{\geq 0}$, where $\mathcal{P}(Z, k)$ is the set of subsets of Z of size k . We focus on s2p similarity functions that are obtained by aggregating the vector of p2p similarities $(s(q_1, x), \dots, s(q_k, x))$ where $s : Z \times Z \rightarrow \mathbb{R}_{\geq 0}$ is a p2p similarity function, as we discussed in the introduction. We call an (S, cS, p_1, p_2) -ALSH for an s2p similarity s^* , an (S, cS, p_1, p_2) -SLSH for s^* . Our focus is on s2p similarities and SLSH families.

From similarities to distances

For distance functions we wish that close points collide with a higher probability than far points do. Specifically, we require that $\Pr_{h \in H}[h(x) = h(y)] \geq p_1$ if $d(x, y) \leq r$, that $\Pr_{h \in H}[h(x) = h(y)] \leq p_2$ if $d(x, y) \geq cr$, and that $c > 1$. We extend the LSH framework for distances to asymmetric distances and for s2p distances, and define ALSH and SLSH families as we did for similarities. As for similarity functions, we consider s2p distance functions that are defined based on the vector of p2p distances $(d(q_1, x), \dots, d(q_k, x))$. In particular, we consider the ℓ_p distance $d_p(Q, x) = \frac{1}{k} \sum_{q \in Q} (d(q, x))^p$ for a constant $p \in \mathbb{N}$ (of which the *average distance* $d_{avg}(Q, x) = d_1(Q, x)$ is a special case), the *geometric distance* $d_{geo}(Q, x) = \prod_{q \in Q} d(q, x)$, and the *center distance* $d_{cen}(Q, x) = \max_{q \in Q} d(q, x)$ of d , where $d : Z \times Z \rightarrow \mathbb{R}_{\geq 0}$ is a p2p distance function.

Additional definitions

We consider the following common p2p similarity functions $s : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$: 1) The *angular similarity* $\angle sim(x, y) = 1 - \frac{\angle(x, y)}{\pi}$, and 2) The *inner product similarity* $ip-sim(x, y) = x^T y$ [22]. We also consider the following common p2p distance functions $d : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$: 1) The angular distance $\angle(x, y)$, and 2) The euclidean distance $ed(x, y) = \|x - y\|_2$.

We say that a hash family is an (S, cS) -LSH for a p2p similarity function s if there exist $p_1 > p_2$ such that it is an (S, cS, p_1, p_2) -LSH. An (S, cS) -LSH family can be used (see [11, 10]) to solve the corresponding (S, cS) -LSH problem of finding an (S, cS) -LSH structure. An (S, cS) -LSH structure finds (with constant probability) a neighbor of similarity at least

cS to a query q if there is a neighbor of similarity at least S to q . We define these concepts analogously (and apply analogous versions of [11, 10]) for ALSH and SLSH hash families and for LSH for distances.

We denote the unit ball in \mathbb{R}^d by B_d and the unit sphere in \mathbb{R}^d by S_d . We also denote $[n] := \{1, \dots, n\}$, and occasionally use the abbreviations $(x_i)_{i=1}^m := (x_1, \dots, x_m)$ and $\{x_i\}_{i=1}^m := \{x_1, \dots, x_m\}$. All the missing proofs (from the body of the paper, and from the appendix) appear in the full version of the paper.³

3 Similarity schemes

We call a (symmetric or asymmetric) similarity function s *achievable* if there exists a hash family H such that for every query q and point x , $\Pr_{(f,g) \in H}[f(q) = g(x)] = s(q, x)$ (for symmetric p2p similarity functions $f = g$). Clearly, such an H is an (S, cS) -ALSH for s for any S and c . In this section, we show that the ℓ_p and geometric s2p similarity functions of an achievable p2p similarity, is by itself achievable and therefore has an (S, cS) -SLSH.

Note that many natural p2p similarity functions are achievable. For example, the random hyperplane hash family [2] achieves the angular similarity function $s(x, y) = 1 - \frac{\angle(x, y)}{\pi}$, the random bit hash family [9] achieves the hamming similarity $s((x_1, \dots, x_d), (y_1, \dots, y_d)) = \frac{|\{i | x_i = y_i\}|}{d}$, and MinHash [7] achieves the Jaccard similarity $s(S, T) = \frac{|S \cap T|}{|S \cup T|}$.

In the full version of the paper, we also give a very simple reduction from the average inner product SLSH problem to the regular inner product ALSH problem (which is not achievable).

ℓ_p similarity

In this section, we define *repeat-SLSH*, and prove that it is an SLSH for the ℓ_p s2p similarity s_p of any achievable p2p similarity function s for any constant $p \in \mathbb{N}$. The intuition behind repeat-SLSH is that given an LSH family that achieves a p2p similarity function s , a query point q collides with a data point x on p randomly and independently selected hash functions with probability $(s(Q, x))^p$. Thus, if we uniformly sample a point $q \in Q$ of the set-query,⁴ and then compute p consecutive hashes of q , the expected collision probability will be the ℓ_p similarity of Q and x . The formal definition is as follows.

► **Definition 2** (Repeat-SLSH). *Let s be an achievable p2p similarity function achieved by a hash family H_s , let k be the size of the set-query, and let $p \in \mathbb{N}$. We define the repeat-SLSH of H_s to be*

$$H = \{(Q \rightarrow (h_j(q_i))_{j=1}^p, x \rightarrow (h_j(x))_{j=1}^p) \mid i \in [k], (h_1, \dots, h_p) \in H_s^p\},$$

where q_i is the i 'th element of the set-query $Q = \{q_1, \dots, q_k\}$ in some consistent arbitrary order.⁵

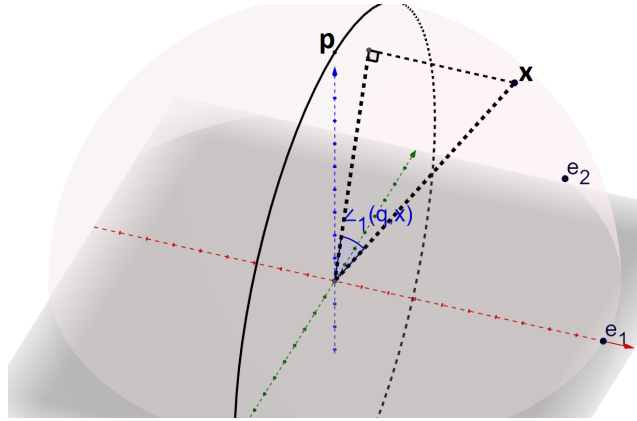
► **Theorem 3.** *Let s be an achievable p2p similarity function, and let H_s be a family that achieves s . Then for any $S > 0$ and $c < 1$, the repeat-SLSH of H_s is an (S, cS) -SLSH for s_p , the ℓ_p similarity of s .*

Proof. It is clear that $\Pr_{(f,g) \in H}[f(Q) = g(x)] = s_p(Q, x)$ for any set-query $Q = \{q_i\}_{i=1}^k$ and data point x , so it is an (S, cS) -SLSH for any $S > 0$ and $c < 1$. ◀

³ The link to the full paper appears in the front matter.

⁴ Therefore, for repeat-SLSH we do not need to know the set-query size k a-priori.

⁵ Let A be a set, and let $p \in \mathbb{N}$. We define $A^p := \{(x_i)_{i=1}^p \mid \forall i, x_i \in A\}$.



■ **Figure 2** An angular ellipsoid ALSH query $(p, \{e_i\}_{i=1}^d)$ and $\angle_1(q, x)$ for some $x \in S_{d+1}$.

The following lemma specifies the relation between the angle of the lifted points and the euclidean distance between the original points. The exact details of the reduction, including the presentation of an SLSH structure for the average euclidean distance, appear in Appendix B.2.

► **Lemma 4.** Let $x, y \in B_d$ and $\varepsilon \in (0, \frac{1}{2}]$, and define $m(\varepsilon) = \frac{\sqrt{1+2\varepsilon^2}}{\sqrt{1-2\varepsilon^2}}$. Then,

$$\varepsilon \|x - y\| \leq \angle(x^\uparrow, y^\uparrow) \leq m(\varepsilon) \cdot \varepsilon \|x - y\|.$$

5 Euclidean ellipsoid ALSH

In this section we present our most technically challenging result – an example that leverages SLSH to solve a geometric extension of the approximate near neighbor problem for the euclidean distance. Our structure is built for a specific “shape” of two concentric ellipsoids (specified by the weights of their axis), and their “sizes”, r and cr , respectively. Given a query which defines the common center and orientation of these ellipsoids, if there is a data point in the smaller r -ellipsoid, then the structure must return a point in the larger cr -ellipsoid. Specifically, we define the euclidean ellipsoid distance as follows.

Euclidean ellipsoid ALSH

Let $q = (p, \{e_i\}_{i=1}^d)$ be a “query” pair where $p \in B_d$ is a center of an ellipsoid and $\{e_i\}_{i=1}^d$ are orthogonal unit vectors specifying the directions of the ellipsoid axes, let $x \in B_d$ be a data point, and let $\{w_1, \dots, w_d\}$ be a fixed set of d rational non-negative weights.

We define the *euclidean ellipsoid distance* $d_o(q, x)$ between q and x with respect to the weights $\{w_1, \dots, w_d\}$ to be $\sum_{i=1}^d w_i (e_i^T(x - p))^2$.

In this section, we describe a structure for the euclidean ellipsoid distance (r, cr) -ALSH problem via a sequence of reductions. We reduce this problem to what we call an *angular ellipsoid ALSH* problem, which is then solved via another reduction to the *weighted geometric angular similarity SLSH* problem, which is solved in Appendix A.2.

We give a high level description of these reductions and defer the details to Appendix C. The first reduction is from the euclidean ellipsoid ALSH to what we call the *angular ellipsoid ALSH*. Recall that in Section 4, we have shown that for small values of ε , the shrink-lift transformation approximately reduces euclidean distances in B_d to angular distances on

for sets of size 2, using the following observation which we adapt to our setting. For any direction e and hyperplane h perpendicular to e through the origin, and any $x \in S_d$, it holds that $\angle_{\text{geo}}(\{e, -e\}, x) = (1 - \angle(x, e)/\pi)(1 - \angle(x, -e)/\pi) = \frac{1}{4} - \frac{\angle(x, h)^2}{\pi^2}$, where $\angle(x, h)$ is the angle between x and its projection on h , and the last step follows by the fact that $\min(\angle(x, e), \angle(x, -e)) = \frac{\pi}{2} - \angle(x, h)$ and $\max(\angle(x, e), \angle(x, -e)) = \frac{\pi}{2} + \angle(x, h)$. Recall that the angular ellipsoid distance between a query $q = (p, \{e_i\}_{i=1}^d)$ and a point x is a weighted sum of $(\angle_i(q, x))^2$. Therefore, if we hash the hyperplane orthogonal to e_i with H-hash, it will collide with higher probability with data points x with a smaller $(\angle_i(q, x))^2$. This suggests that we can answer an angular ellipsoid query $q = (p, \{e_i\}_{i=1}^d)$ by a weighted geometric angular similarity SLSH set-query where the set is the union of the sets $\{e_i, -e_i\}$ for all $i \in [d]$, using the angular ellipsoid weight w_i associated with the axis e_i for each $i \in [d]$. Specifically, the corresponding set-query is $Q = \{e_1, -e_1, e_2, -e_2, \dots, e_d, -e_d\}$, and the structure is built with the weights $\{w_1, w_1, w_2, w_2, \dots, w_d, w_d\}$. For the reduction's analysis to hold, we must require that any query $q = (p, \{e_i\}_{i=1}^d)$ and data point x satisfy $\angle(p, x) \leq \sqrt{\frac{c-1}{c}} \cdot \frac{\pi}{4}$. This can be easily guaranteed by taking a sufficiently small value of ε in the previous reduction from euclidean ellipsoids to angular ellipsoids, such that the set of transformed queries and data points has a sufficiently small angular diameter.

Finally, the weighted geometric angular similarity SLSH problem is solved in Appendix A.2.

6 Center euclidean distance for set-queries of size 2

In this section we present a data structure for the center euclidean (r, cr) -SLSH problem. This is among our most technically challenging results. Our data structure receives a set-query $Q = \{q_1, q_2\}$ and returns (with constant probability) a data point v such that $ed_{\text{cen}}(Q, v) = \max(\|v - q_1\|, \|v - q_2\|) \leq cr$, if there is a data point v such that $ed_{\text{cen}}(Q, v) = \max(\|v - q_1\|, \|v - q_2\|) \leq r$.

Our data structure requires that c is larger than c_{\min} where $c_{\min} = \frac{3}{2\sqrt{2}} \approx 1.06066$ is a constant slightly larger than 1. We also assume that the possible queries $Q = \{q_1, q_2\}$ are such that $\frac{1}{2}\|q_1 - q_2\| < (1 - \phi)r$, for a parameter $\phi < 1$ that is known to the structure.⁸

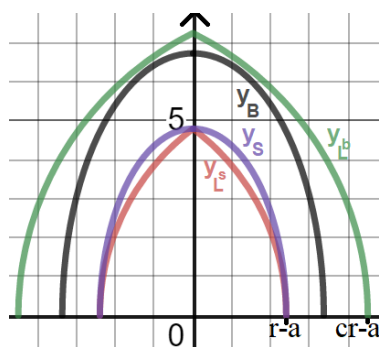
We construct our structure via a reduction to the euclidean ellipsoid ALSH from Section 5. Consider the query $Q = \{q_a, q_{-a}\}$ to the center euclidean SLSH structure where $q_a = (a, 0, \dots, 0)$ and $q_{-a} = (-a, 0, \dots, 0)$, for some $0 < a < (1 - \phi)r/2$. Let $L^s = \{v \mid \max(\|v - q_a\|, \|v - q_{-a}\|) \leq r\}$ be the set of point of center distance at most r from Q , and let $L^b = \{v \mid \max(\|v - q_a\|, \|v - q_{-a}\|) \leq cr\}$ be the set of point of center distance at most cr from Q . We also define the following two ellipsoids S and B centered at the origin with axes aligned with the standard axes x_1, \dots, x_d :

$$S = \left\{ (x_1, \dots, x_d) \mid \frac{r+a}{r-a}x_1^2 + \sum_{i=2}^d x_i^2 \leq r^2 - a^2 \right\},$$

$$B = \left\{ (x_1, \dots, x_d) \mid \frac{r+a}{r-a}x_1^2 + \sum_{i=2}^d x_i^2 \leq \left(\frac{cr}{c_{\min}}\right)^2 - a^2 \right\}.$$

Our reduction depends on the crucial observation stated in the following lemma.

⁸ For queries $Q = \{q_1, q_2\}$ such that $\frac{1}{2}\|q_1 - q_2\| > r$, no point v can satisfy $\max(\|v - q_1\|, \|v - q_2\|) \leq r$, and returning no points for such queries satisfies our structure requirements trivially.



■ **Figure 4** Plots of y_{L^s} , y_S , y_B , and y_{L^b} as functions of x_1 . $a = 3.6$, $r = 6$, $c = 1.35$.

► **Lemma 5.** *We have that $L^s \subseteq S \subseteq B \subseteq L^b$.*

To illustrate the relation between L^s , S , B , and L^b , we denote the distances of their boundaries from the axis x_1 by $y_{L^s}(x_1)$, $y_S(x_1)$, $y_B(x_1)$ and $y_{L^b}(x_1)$, respectively. These functions are plotted in Figure 4.

Intuitively, our reduction will replace L^s by S and L^b by B : If there is a point x in L_s then x is also in S and the euclidean ellipsoid structure will find a point in B which is in L^b . Specifically, we would like to query with $\{q_a, q_{-a}\}$ a euclidean ellipsoid $(r', c'r')$ -ALSH structure where $r' = r^2 - a^2$, c' is set such that $c'r' = \left(\frac{cr}{c_{\min}}\right)^2 - a^2$, and the weights are $\left\{\frac{r+a}{r-a}, 1, \dots, 1\right\}$.

The problem is that a depends on the query (it is half the distance between the query points) and obviously we cannot prepare a different euclidean ellipsoid $(r', c'r')$ -ALSH structure for each query. To overcome this we quantize the range of possible values of a and construct a data structure for each quantized value. The range of the possible values for a is $[0, (1 - \phi)r]$ and our quantization consists of the values $i \cdot \delta$ for $i = 0, \dots, \lceil \frac{(1-\phi)r}{\delta} \rceil$ where $\delta = \min\left(\frac{1}{2}, 1 - \sqrt{\frac{c_{\min}}{c}}\right) \phi r$.^{9,10}

The euclidean ellipsoid $(r', c'r')$ -ALSH structure corresponding to the value $i \cdot \delta$ has $r' = \frac{c}{c_{\min}} \cdot (r^2 - (i \cdot \delta)^2)$, $c' = \frac{c}{c_{\min}}$ and weights $\left\{\frac{r+i \cdot \delta}{r-i \cdot \delta}, 1, \dots, 1\right\}$. For correctness we will prove that the ellipsoids

$$S^+ = \left\{ (x_1, \dots, x_d) \mid \frac{r+a'}{r-a'} x_1^2 + \sum_{i=2}^d x_i^2 \leq \frac{c}{c_{\min}} \cdot (r^2 - (a')^2) \right\} \text{ and}$$

$$B^- = \left\{ (x_1, \dots, x_d) \mid \frac{r+a'}{r-a'} x_1^2 + \sum_{i=2}^d x_i^2 \leq \left(\frac{c}{c_{\min}}\right)^2 \cdot (r^2 - (a')^2) \right\}, \text{ where } a' = \lceil \frac{a}{\delta} \rceil \cdot \delta, \text{ are}$$

such that $S \subseteq S^+ \subseteq B^- \subseteq B$. One can easily show that $r \geq a' \geq 0$, so the coefficients of x_1^2 and the right hand side of the equations in S^+ and B^- are both non-negative and well-defined.

Query phase

Let $Q = \{q_1, q_2\} \subseteq B_d$ be a set-query where $\|q_1 - q_2\| = 2a$ for $a \in [0, (1 - \phi)r]$. Let $a' = \lceil \frac{a}{\delta} \rceil \delta$ as before. To get the answer, we query the euclidean ellipsoid $(r', c'r')$ -ALSH structure, where $r' = \frac{c}{c_{\min}} \cdot (r^2 - (a')^2)$, $c' = \frac{c}{c_{\min}}$ and the weights are $\left\{\frac{r+a'}{r-a'}, 1, \dots, 1\right\}$ with a query q defined as follows.

⁹ To ensure rationality of weights, if δ is irrational, we replace it by $\mathbb{Q}_{>0} \ni \delta' < \delta$.

¹⁰ Intuitively, when c is close to c_{\min} , and when ϕ is small, our quantization is finer.

Let R_{q_1, q_2} be a rigid transformation (rotation and translation) such that $R_{q_1, q_2}(q_1) = q_a$ and $R_{q_1, q_2}(q_2) = q_{-a}$ for $q_a = (a, 0 \dots, 0)$ and $q_{-a} = (-a, 0 \dots, 0)$. We set $q = (p, \{\bar{e}_i\}_{i=1}^d)$ where $p = R_{q_1, q_2}^{-1}((0, \dots, 0)) = \frac{q_1 + q_2}{2} \in B_d$ and $\forall i, \bar{e}_i = R_{q_1, q_2}^{-1}(e_i)$ where $\{e_i\}_{i=1}^d$ is the standard basis of \mathbb{R}^d . Our main result is,

► **Theorem 6.** *The structure described above is an (r, cr) -SLSH structure for the center euclidean distance and queries of size 2. (For any $c > c_{\min}$, and queries $Q = \{q_1, q_2\}$ such that $\frac{1}{2} \|q_1 - q_2\| < (1 - \phi)r$.)*

7 Conclusions and directions for future work

We present a novel extended LSH framework, motivated by group recommendation systems. We define several set-query extensions for distance and similarity functions, and show how to design SLSH families and data structures for them using different techniques. We use this framework to solve a geometric extension of the euclidean distance approximate near neighbor problem, which we call *euclidean ellipsoid ALSH*, via reduction to an SLSH problem. All the reductions we describe have some performance loss, which (for distance functions) is expressed by a smaller p_1 and p_2 , and a worse value of ρ . Estimating the exact performance loss (the value of ρ) and finding more efficient reductions is an interesting line of research. Finding a method for the center euclidean distance for set-queries larger than two is another intriguing open question.

References

- 1 Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *Proceedings of the VLDB Endowment*, 2(1):754–765, 2009. doi:10.14778/1687627.1687713.
- 2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468. IEEE, 2006. doi:10.1109/FOCS.2006.49.
- 3 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801. ACM, 2015. doi:10.1145/2746539.2746553.
- 4 Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. In *SOCG*, pages 1–11. ACM, 2016. doi:10.4230/LIPIcs.SocG.2016.9.
- 5 Liliana Ardissono, Anna Goy, Giovanna Petrone, Marino Segnan, and Pietro Torasso. Tailoring the recommendation of tourist information to heterogeneous user groups. In *Workshop on adaptive hypermedia*, pages 280–295. Springer, 2001. doi:10.1007/3-540-45844-1_26.
- 6 Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 257–264. ACM, 2014. doi:10.1145/2645710.2645741.
- 7 Andrei Z Broder. On the resemblance and containment of documents. In *Compression and complexity of sequences*, pages 21–29. IEEE, 1997. doi:10.1109/SEQUEN.1997.666900.
- 8 Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SOCG*, pages 253–262. ACM, 2004. doi:10.1145/997817.997857.
- 9 Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999. URL: <http://www.vldb.org/conf/1999/P49.pdf>.
- 10 Sarel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012. doi:10.4086/toc.2012.v008a014.

- 11 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613. ACM, 1998. doi:10.1145/276698.276876.
- 12 Prateek Jain, Sudheendra Vijayanarasimhan, and Kristen Grauman. Hashing hyperplane queries to near points with applications to large-scale active learning. *Transactions on Pattern Analysis and Machine Intelligence*, pages 276–288, 2014. doi:10.1109/TPAMI.2013.121.
- 13 Anthony Jameson. More than the sum of its members: challenges for group recommender systems. In *AVI*, pages 48–54. ACM, 2004. doi:10.1145/989863.989869.
- 14 Anthony Jameson and Barry Smyth. Recommendation to groups. In *The adaptive web*, pages 596–627. Springer, 2007. doi:10.1007/978-3-540-72079-9_20.
- 15 Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi:10.1109/MC.2009.263.
- 16 Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007. URL: <http://www.vldb.org/conf/2007/papers/research/p950-lv.pdf>.
- 17 Judith Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. In *Personalized digital television*, pages 93–141. Springer, 2004. doi:10.1023/B:USER.0000010138.79319.fd.
- 18 Kevin McCarthy, Lorraine McGinty, Barry Smyth, and Maria Salamó. The needs of the many: a case-based group recommender system. In *ECCBR*, pages 196–210. Springer, 2006. doi:10.1007/11805816_16.
- 19 Rajeev Motwani, Assaf Naor, and Rina Panigrahi. Lower bounds on locality sensitive hashing. In *SOCG*, pages 154–157. ACM, 2006. doi:10.1145/1137856.1137881.
- 20 Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *ICML*, pages 1926–1934, 2015. URL: <http://proceedings.mlr.press/v37/neyshabur15.html>.
- 21 Mark O’connor, Dan Cosley, Joseph A Konstan, and John Riedl. PolyLens: a recommender system for groups of users. In *ECSCW*, pages 199–218. Springer, 2001. doi:10.1007/0-306-48019-0_11.
- 22 Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, pages 2321–2329, 2014. URL: <http://arxiv.org/abs/1405.5869>.
- 23 Zhiwen Yu, Xingshe Zhou, Yanbin Hao, and Jianhua Gu. Tv program recommendation for multiple viewers based on user profile merging. *UMUAI*, 16(1):63–82, 2006. doi:10.1007/s11257-006-9005-6.

A Missing parts from Section 3

A.1 Geometric similarity

In this section, we define *exhaustive-SLSH*, and prove that it is an SLSH for the geometric similarity, s_{geo} , of any achievable p2p similarity function s .

Note that the geometric similarity is somewhat similar to the center similarity - both similarities are suitable when we want to enforce high similarity to all points of the set-query. Our scheme for center similarity given in Section 6 is technically challenging. Thus, exhaustive-SLSH could be a simple alternative that somewhat relaxes the requirement to be similar to all points of the query for simplicity.

The intuition behind exhaustive-SLSH is that given an LSH family H that achieves a p2p similarity function s , then for a set-query $Q = \{q_1, \dots, q_k\}$ and a point x , the expected collision probability of $(h_1(q_1), \dots, h_k(q_k))$ with $(h_1(x), \dots, h_k(x))$ when the $\{h_i\}$ ’s are sampled from H , is $s_{geo}(Q, x)$. The formal definition is as follows.

Exhaustive-SLSH

Let s be an achievable p2p similarity function achieved by a hash family H_s , and let k be the set-query size. We define the exhaustive-SLSH of H_s to be the following family of pairs $H = \{(Q \rightarrow (h_j(q_j))_{j=1}^k, x \rightarrow (h_j(x))_{j=1}^k) \mid (h_1, \dots, h_k) \in H_s^k\}$.

► **Theorem 7.** *Let s be an achievable p2p similarity, and H_s be a family that achieves s . Then the exhaustive-SLSH of H_s is an SLSH for the geometric similarity of s .*

Proof. It is clear that $\Pr_{(f,g) \in H}[f(Q) = g(x)] = s_{geo}(Q, x)$ for any set-query $Q = \{q_i\}_{i=1}^k$ and data point x , so it is an (S, cS) -SLSH for any $S > 0$ and $c < 1$. ◀

A.2 Weighted geometric similarity

In this section, we define *weighted exhaustive-SLSH*, and prove that it is an SLSH structure for the weighted geometric similarity s_{wgeo} of any achievable p2p similarity function s . So far, we have only considered equal-weighted query points, however, motivated by recommending movies to a set of people, a logical extension would be giving the individuals weights according to their importance, or the strength of their general preferences. To define the *weighted geometric similarity*, we use a sequence of non-negative **rational** weights $W = \{w_1, \dots, w_k\}$, where each w_i is defined by a pair (a_i, b_i) such that $a_i \in \mathbb{N} \cup \{0\}$, $b_i \in \mathbb{N}$, and $w_i = \frac{a_i}{b_i}$, and k is the set-query size. Given W and a p2p similarity function s , we define the weighted geometric similarity (of s) of a set-query $Q = \{q_1, \dots, q_k\}$ and a data point x to be $s_{wgeo}(Q, x) = \prod_{i=1}^k (s(q_i, x))^{w_i}$.¹¹ In case the underlying p2p similarity function s is achievable, we reduce the weighted geometric similarity (S, cS) -SLSH problem to the geometric similarity $(S', c'S')$ -SLSH problem.

Weighted exhaustive-SLSH

Given $S > 0$, $c < 1$, a p2p similarity function s , the set-query size k , and non-negative rational weights $\{w_i\}_{i=1}^k$ as defined above, we define $m = lcm(\{b_i\}_{i=1}^k) \in \mathbb{N}$.¹² The weighted exhaustive-SLSH structure works as follows. In the preprocessing phase, we store all the data points in an $(S^m, c^m S^m)$ -SLSH structure for the geometric similarity for a set-query of size $k' = m \cdot \sum_{i=1}^k w_i$.¹³ Given a set-query $Q = \{q_i\}_{i=1}^k$, we query the structure built in the preprocessing phase, with the set-query $T(Q) = \{q_1, \dots, q_1, \dots, q_k, \dots, q_k\}$,¹⁴ where each $q_i \in T(Q)$ is repeated $m \cdot w_i = a_i \cdot \frac{m}{b_i} \in \mathbb{N}$ times.

► **Theorem 8.** *Weighted exhaustive-SLSH is an (S, cS) -SLSH structure for the weighted geometric similarity s_{wgeo} of any achievable p2p similarity function s .*

Proof. Observe that for any set-query $Q = \{q_i\}_{i=1}^k$ of size k and any data point x , it holds that $s_{geo}(T(Q), x) = \prod_{i=1}^k (s(q_i, x))^{m \cdot w_i} = \left(\prod_{i=1}^k (s(q_i, x))^{w_i} \right)^m = (s_{wgeo}(Q, x))^m$. Thus, the claim follows since if there is a data point x such that $s_{wgeo}(Q, x) \geq S$, then $s_{geo}(T(Q), x) \geq S^m$, and the $(S^m, c^m S^m)$ -SLSH structure finds a data point x such that $s_{geo}(T(Q), x) \geq c^m S^m$, i.e., such that $s_{wgeo}(Q, x) \geq cS$. ◀

¹¹ For weighted similarities we assume that the set-query is ordered, and this order determines the correspondence between the weights and the points in the set-query.

¹² By *lcm* we denote the least common multiple.

¹³ We can derive such a structure from exhaustive-SLSH (which can be applied since s is achievable).

¹⁴ We allow set-queries that are in fact multi-sets. All our derivations apply to multi set-queries.

B Detailed results from Section 4

B.1 Average angular distance

We warm up with an easy result, and show that repeat-SLSH for the average angular **similarity** (Section 3) is an SLSH family for the average angular **distance** - a fact that follows since the average angular **similarity** is a decreasing function with respect to the average angular **distance**.

► **Theorem 9.** *Repeat-SLSH for the average angular **similarity** is an SLSH for the average angular **distance** \angle_{avg} .*

Proof. For any set-query Q of size k and data point x , $\angle sim_{avg}(Q, x) = \frac{1}{k} \sum_{q \in Q} \left(1 - \frac{\angle(q, x)}{\pi}\right) = 1 - \frac{\frac{1}{k} \sum_{q \in Q} \angle(q, x)}{\pi} = 1 - \frac{\angle_1(Q, x)}{\pi}$. Thus, the claim follows since for any $r > 0$ and $c > 1$, by Theorem 3, repeat-SLSH for the average angular similarity is an $(1 - \frac{r}{\pi}, 1 - \frac{cr}{\pi}, p_1, p_2)$ -SLSH for $\angle sim_1$ for some $p_1 > p_2$, and specifically is an (r, cr, p_1, p_2) -SLSH for \angle_{avg} . ◀

B.2 Average euclidean distance

We give a formal definition of Shrink-lift-SLSH, which reduces the average euclidean distance problem to the average angular distance problem. Shrink-lift-SLSH works as follows.

Preprocessing phase. Given the parameters $r > 0$, $c > 1$ and the set-query size k , define $\varepsilon = \frac{1}{2} \sqrt{1 - \frac{2}{1+c^2}} < \frac{1}{2}$. We transform each data point x to x^\uparrow , and store the transformed data points in an $(r', c'r')$ -SLSH structure for the average angular distance, for the parameters $r' = m(\varepsilon) \cdot \varepsilon r$, $c' = \frac{\varepsilon cr}{r'} = \frac{c}{m(\varepsilon)}$ and $k' = k$, where we define $m : [0, \frac{1}{2}] \rightarrow \mathbb{R}$ by $m(x) = \frac{\sqrt{1+2x^2}}{\sqrt{1-2x^2}}$.

Query phase. Let Q be a set-query of size k . We query the average angular distance $(r', c'r')$ -SLSH structure constructed in the preprocessing phase with the set-query $Q' = \{q^\uparrow \mid q \in Q\}$.

In order to prove that shrink-lift-SLSH is an (r, cr) -SLSH structure for the average euclidean distance, Lemma 10 bounds the angle between the lifted points in terms of their original euclidean distance, using the error function $e(\varepsilon, x, y) := \left(\sqrt{\frac{1}{\varepsilon^2} - \|x\|^2} - \sqrt{\frac{1}{\varepsilon^2} - \|y\|^2} \right)^2$.

► **Lemma 10.** *Let $x, y \in B_d$ and $\varepsilon \in (0, 1]$. Then*

$$2 \sin^{-1} \left(\frac{\varepsilon}{2} \cdot \|x - y\| \right) \leq \angle(x^\uparrow, y^\uparrow) = 2 \sin^{-1} \left(\frac{\varepsilon}{2} \sqrt{\|x - y\|^2 + e(\varepsilon, x, y)} \right).$$

The following lemma bounds the error term.

► **Lemma 11.** *For any $x, y \in B_d$ and $\varepsilon \in (0, \frac{1}{2}]$, $0 \leq e(\varepsilon, x, y) \leq \frac{4}{3} \|x - y\|^2 \varepsilon^2$.*

Next, we show the following property of $\sin^{-1}(\cdot)$, which is used in the proof of Lemma 13, and later in the proof of Lemma 15.

► **Lemma 12.** *$x \leq \sin^{-1}(x) \leq \frac{x}{\sqrt{1-x^2}}$ for any $x \in [0, 1)$.*

Then, we use Lemmas 10, 11 and 12 to derive the following important Lemma.

► **Lemma 13.** *Let $x, y \in B_d$ and $\varepsilon \in (0, \frac{1}{2}]$. Then, $\varepsilon \|x - y\| \leq \angle(x^\uparrow, y^\uparrow) \leq m(\varepsilon) \cdot \varepsilon \|x - y\|$.*

Finally, we use Lemma 13 to prove the following theorem, which is the main result of this section.

► **Theorem 14.** *Shrink-lift-SLSH is an (r, cr) -SLSH structure for the average euclidean distance ed_{avg} .*

C Euclidean ellipsoid ALSH detailed presentation

In this section, we give a detailed presentation of the two reductions we use to solve the euclidean ellipsoid ALSH problem from Section 5. Section C.1 gives a reduction from the euclidean ellipsoid ALSH to the angular ellipsoid problem. Section C.2 then reduces this problem to the weighted geometric angular similarity SLISH problem, which is solved in Appendix A.2. We note that this reduction requires that any query $q = (p, \{e_i\}_{i=1}^d)$ and data point x in the angular ellipsoid structure satisfy $\angle(p, x) \leq \sqrt{\frac{c-1}{c}} \cdot \frac{\pi}{4}$. As we will see, the inputs to the angular ellipsoid structure that we produce by the first reduction (i.e., from the euclidean ellipsoid problem) will satisfy this requirement.

It is worth mentioning that the solution in Appendix A.2 requires that the weights are rational, hence we also require rational weights in both the ellipsoid structures.

C.1 From euclidean ellipsoid ALSH to angular ellipsoid ALSH

In this section, we reduce the euclidean ellipsoid (r, cr) -ALSH problem to an angular ellipsoid $(r', c'r')$ -ALSH problem. To do this, we use the shrink-lift transformation $(\cdot)^\uparrow$ from Section 4 with an appropriately tuned shrinking parameter ε , to map our data points from B_d to S_{d+1} . For our proofs of Lemma 15 and Theorem 16 to hold, we need that $\varepsilon \leq \frac{1}{8}$. Additionally, to prove that the parameter c' that we use for the angular ellipsoid $(r', c'r')$ structure is larger than 1 (Theorem 16), we need that $\varepsilon \leq \frac{\sqrt[8]{c-1}}{\sqrt[8]{c+1}}$ and $\varepsilon \leq \sqrt{\frac{(c-\sqrt{c})r}{5(\sqrt{c+1}) \cdot \sum_{i=1}^d w_i}}$. Finally,

to ensure that $\angle(p, x) \leq \sqrt{\frac{c-1}{c}} \cdot \frac{\pi}{4}$ for any query $q = (p, \{e_i\}_{i=1}^d)$ and data point x in the angular ellipsoid structure (see the proof of Theorem 16 in the full paper), we need that $\varepsilon \leq \sqrt{1 - \frac{1}{\sqrt[4]{c}}} \cdot \frac{\pi}{8\sqrt{2}}$. We therefore set ε to be the minimum of all these upper bounds, that is

$$\varepsilon = \min \left(\frac{1}{8}, \frac{\sqrt[8]{c-1}}{\sqrt[8]{c+1}}, \sqrt{\frac{(c-\sqrt{c})r}{5(\sqrt{c+1}) \cdot \sum_{i=1}^d w_i}}, \sqrt{1 - \frac{1}{\sqrt[4]{c}}} \cdot \frac{\pi}{8\sqrt{2}} \right).$$

We store the images (by the shrink-lift transformation) of our data points in the angular ellipsoid $(r', c'r')$ -ALSH structure.¹⁵ We recall (Lemma 13) that for a sufficiently small ε the angular distance between x^\uparrow and y^\uparrow is approximately equal to ε times the euclidean distance between x and y . We set $r' = \varepsilon^2(1+\varepsilon)^2 \cdot (r + 5\beta(\varepsilon) \cdot \sum_{i=1}^d w_i)$, and $c' = \frac{\varepsilon^2(1-\varepsilon)^2 \cdot (cr - 5\beta(\varepsilon) \cdot \sum_{i=1}^d w_i)}{r'}$, where $\beta(\varepsilon) = \frac{1-\sqrt{1-\varepsilon^2}}{\sqrt{1-\varepsilon^2}} \approx \frac{\varepsilon^2}{2} \geq 0$. Our choice of ε guarantees that $\beta(\varepsilon) \cdot \sum_{i=1}^d w_i \ll r$ and thereby r' is approximately $\varepsilon^2 \cdot r$, as we expect since the angular ellipsoid distance is a sum of (weighted) squared angular distances each of which is smaller by a factor of ε from its corresponding euclidean distance. Notice also that for our choice of ε , c' is approximately equal to $\sqrt[4]{c}$.¹⁶ The angular ellipsoid structure uses the same weights as of the euclidean ellipsoid structure.

The query

Let $q_0 = (p, \{e_i\}_{i=1}^d)$ be a euclidean ellipsoid query, where $p \in B_d$ is a center of an ellipsoid and $\{e_i\}_{i=1}^d$ are the unit vectors of \mathbb{R}^d in the directions of the ellipsoid axes. We query the angular ellipsoid structure constructed in the preprocessing phase with the angular ellipsoid

¹⁵We do not want to set ε to be too small since this is likely to deteriorate the performance of the angular ellipsoid structure on these images.

¹⁶By using a smaller ε we can make c' closer to c .

query $q = (p^\uparrow, \{\bar{e}_i\}_{i=1}^d)$, where each \bar{e}_i is obtained by rotating $(e_i, 0)$ in the direction of $(0, \dots, 0, 1)$, until its angle with p^\uparrow becomes $\frac{\pi}{2}$ (this is illustrated in Figure 3). Formally, we define $\bar{e}_i := (a_i \cdot e_i; \sqrt{1 - a_i^2})$ where $a_i = -\text{sign}(p_i) \cdot \sqrt{\frac{1 - \|\varepsilon p\|^2}{\varepsilon^2 p_i^2 + 1 - \|\varepsilon p\|^2}} \in [-1, 1]$, and $\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$. To simplify the expression above, we define $z(p, \varepsilon) := \sqrt{\varepsilon^2 p_i^2 + 1 - \|\varepsilon p\|^2}$,

so we get that $a_i = -\text{sign}(p_i) \cdot \frac{\sqrt{1 - \|\varepsilon p\|^2}}{z(p, \varepsilon)}$, and $\sqrt{1 - a_i^2} = \frac{\varepsilon |p_i|}{z(p, \varepsilon)} = \frac{\varepsilon p_i \cdot \text{sign}(p_i)}{z(p, \varepsilon)}$.

Note that this definition of \bar{e}_i makes \bar{e}_i orthogonal to p^\uparrow . Indeed, $\bar{e}_i^T \cdot p^\uparrow = a_i \cdot \varepsilon p_i + \sqrt{1 - a_i^2} \cdot \sqrt{1 - \|\varepsilon p\|^2} = -\text{sign}(p_i) \cdot \frac{\sqrt{1 - \|\varepsilon p\|^2}}{z(p, \varepsilon)} \cdot \varepsilon p_i + \frac{\varepsilon p_i \cdot \text{sign}(p_i)}{z(p, \varepsilon)} \cdot \sqrt{1 - \|\varepsilon p\|^2} = 0$, where the first equality follows from the definition $x^\uparrow = (\varepsilon x_1, \dots, \varepsilon x_d, \sqrt{1 - \|\varepsilon x\|^2})$.

The following Lemma implies the correctness of our structure, stated in Theorem 16.

► **Lemma 15.** *Let $\varepsilon \in (0, \frac{1}{2})$, $x, p \in B_d$, and a euclidean ellipsoid query $q_0 = (p, \{e_i\}_{i=1}^d)$, where $\{e_i\}_{i=1}^d$ is the standard basis in \mathbb{R}^d . Then taking $q = (p^\uparrow, \{\bar{e}_i\}_{i=1}^d)$ as above, for every $i \in [d]$ we have that $\max(0, \varepsilon(1 - \varepsilon) \cdot (|x_i - p_i| - \beta(\varepsilon))) \leq \angle_i(q, x^\uparrow) \leq \varepsilon(1 + \varepsilon) \cdot (|x_i - p_i| + \beta(\varepsilon))$, where $\angle_i(q, x)$ is the angular distance between x and its projection on the hyperplane orthogonal to e_i (see Figure 2).*

In Section C.2, we show the existence of an angular ellipsoid $(r', c'r')$ -ALSH structure, so we conclude the following theorem.

► **Theorem 16.** *The structure above is an (r, cr) -ALSH structure for the euclidean ellipsoid distance d_\circ .*

Our reduction guarantees that any query $q = (p^\uparrow, \{\bar{e}_i\}_{i=1}^d)$ for the angular ellipsoid structure and any data point x^\uparrow stored in it, satisfy $\angle(p, x) \leq \sqrt{\frac{c'-1}{c'}} \cdot \frac{\pi}{4}$ as required.

C.2 From angular ellipsoid ALSH to weighted geometric angular similarity SLSH

In this section, we reduce the angular ellipsoid (r, cr) -ALSH problem that we have studied in Section C.1, to a weighted geometric angular similarity $(r', c'r')$ -SLSH problem.

C.2.1 H-hash - the LSH scheme of Jain et al.

Our data structure is based on the H-hash of Jain et al. [12]. The H-hash stores points which reside on S_{d+1} such that for a query hyperplane h through the origin, we can efficiently retrieve the data points that have a small angular distance with their projection on h .

H-hash in fact uses an SLSH family for the s2p geometric angular similarity for sets of size 2. That is, a hash function is defined by two random directions u and v . We hash a point x to the concatenation of $\text{sign}(x^T u)$ and $\text{sign}(x^T v)$ and we represent a query hyperplane h , perpendicular to e , by the set $\{e, -e\}$, which is hashed to the concatenation of $\text{sign}(e^T u)$ and $\text{sign}((-e)^T v)$.

The probability that a data point x collides with the hyperplane h perpendicular to e is equal to $\angle \text{sim}(x, e) \cdot \angle \text{sim}(x, -e) = (1 - \angle(x, e)/\pi)(1 - \angle(x, -e)/\pi)$. This collision probability increases with the angle between x and its projection on h , and attains its maximum when x is on h .

Recall that the angular ellipsoid distance between a query $q = (p, \{e_i\}_{i=1}^d)$ and a point x is a weighted sum of the terms $(\angle_i(q, x))^2$. Therefore, if we hash the hyperplane orthogonal to e_i with H-hash, it will collide with higher probability with data points x with a smaller

28:18 LSH for Set-Queries, Motivated by Group Recommendations

$\angle_i(q, x)$. This suggests that we can answer an angular ellipsoid query $q = (p, \{e_i\}_{i=1}^d)$ by a weighted geometric angular similarity SLSH set-query where the set is the union of the sets $\{e_i, -e_i\}$ for all $i \in [d]$, using an appropriate weight w_i for each $i \in [d]$. Specifically, given the parameters $r > 0$ and $c > 1$, we store the data points in an $(S', c'S')$ -SLSH structure for the weighted geometric angular s2p similarity for queries of size $k' = 2d$ and with the weights $\{w_1, w_1, w_2, w_2, \dots, w_d, w_d\}$.¹⁷ We define c' and S' as follows

$$S' = e^{\sum_{i=1}^d w_i \cdot \ln\left(\frac{1}{4}\right) - \frac{4r}{\pi^2 - 4\psi_c^2}}, \text{ and } c' = \frac{e^{\sum_{i=1}^d w_i \ln\left(\frac{1}{4}\right) - \frac{4cr}{\pi^2}}}{S'} = e^{-4r \left(\frac{c}{\pi^2} - \frac{1}{\pi^2 - 4\psi_c^2} \right)},$$

where we define $\psi_c = \sqrt{\frac{c-1}{c}} \cdot \frac{\pi}{4}$.¹⁸ To answer an angular ellipsoid query $q = (p, \{e_i\}_{i=1}^d)$, we query our structure with the set-query $Q = \{e_1, -e_1, e_2, -e_2, \dots, e_d, -e_d\}$. For the reduction to succeed, we require that any query $q = (p, \{e_i\}_{i=1}^d)$ and data point x satisfy $\angle(p, x) \leq \sqrt{\frac{c-1}{c}} \cdot \frac{\pi}{4}$.

Correctness of our structure follows from the following two theorems.

► **Theorem 17.** *Let $x \in S_{d+1}$ and $q = (p, \{e_i\}_{i=1}^d)$ be an angular ellipsoid query. Then, $\angle_{\text{sim}_{\text{geo}}}(\{e_i, -e_i\}, x) = \frac{1}{4} - \frac{\angle_i(q, x)^2}{\pi^2}$ for all $i \in [d]$.*

► **Theorem 18.** *The structure above is an (r, cr) -ALSH structure for the angular ellipsoid distance d_{\angle_\circ} .*

¹⁷ Such a structure is given in Appendix A.2.

¹⁸ In the full paper we prove that $c' < 1$.

Fast Multi-Subset Transform and Weighted Sums over Acyclic Digraphs

Mikko Koivisto

Department of Computer Science, University of Helsinki, Finland
mikko.koivisto@helsinki.fi

Antti Röyskö

Department of Computer Science, University of Helsinki, Finland
antti.roysko@helsinki.fi

Abstract

The zeta and Moebius transforms over the subset lattice of n elements and the so-called subset convolution are examples of unary and binary operations on set functions. While their direct computation requires $O(3^n)$ arithmetic operations, less naive algorithms only use $2^n \text{poly}(n)$ operations, nearly linear in the input size. Here, we investigate a related n -ary operation that takes n set functions as input and maps them to a new set function. This operation, we call multi-subset transform, is the core ingredient in the known inclusion–exclusion recurrence for weighted sums over acyclic digraphs, which extends Robinson’s recurrence for the number of labelled acyclic digraphs. Prior to this work, the best known complexity bound for computing the multi-subset transform was the direct $O(3^n)$. By reducing the task to rectangular matrix multiplication, we improve the complexity to $O(2.985^n)$.

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

Keywords and phrases Bayesian networks, Moebius transform, Rectangular matrix multiplication, Subset convolution, Weighted counting of acyclic digraphs, Zeta transform

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.29

Funding This work was partially supported by the Academy of Finland, Grant 316771.

Acknowledgements We thank Petteri Kaski for valuable discussions about the topic of the paper.

1 Introduction

In this paper, we consider the following problem. We are given a finite set U and, for each element $i \in U$, a function f_i from the subsets of U to some ring \mathcal{R} . The task is to compute the function g given by

$$g(T) = \sum_{S \subseteq T} \prod_{i \in T} f_i(S), \quad T \subseteq U. \quad (1)$$

We shall call g the *multi-subset transform* of $(f_i)_{i \in U}$. While the present study of this operation on set functions stems from a particular application to weighted counting of acyclic digraphs, which we will introduce later in this section, we believe the multi-subset transform could also have applications elsewhere.

A straightforward computation of the multi-subset transform requires $\Omega(3^n)$ arithmetic operations (i.e., additions and multiplications in the ring \mathcal{R}) when U has n elements. In the light of the input size $O(2^n n)$ and output size $O(2^n)$, one could hope for an algorithm that requires $2^n n^{O(1)}$ operations. Some support for optimism is provided by the close relation to two similar operations on set functions: the zeta transform of f and the subset convolution of f_1 and f_2 , given respectively by

$$(f\zeta)(T) = \sum_{S \subseteq T} f(S) \quad \text{and} \quad (f_1 * f_2)(T) = \sum_{S \subseteq T} f_1(S) f_2(T \setminus S), \quad T \subseteq U;$$



© Mikko Koivisto and Antti Röyskö;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 29; pp. 29:1–29:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

these unary and binary operations can be performed using $O(2^n n)$ [18, 12] and $O(2^n n^2)$ [2] arithmetic operations, thus significantly beating the naive $\Omega(3^n)$ -computation. Indeed, consider the seemingly innocent replacement of “ $i \in T$ ” by “ $i \in S$ ” or “ $i \in U$ ” in (1): either one would yield a variant that immediately (and efficiently) reduces to the zeta transform. Likewise, replacing the factor $\prod_{i \in T \setminus S} f_i(S)$ in the product by $\prod_{i \in T \setminus S} f_i(T \setminus S)$ would give us an instance of subset convolution. The present authors do not see how to fix these “broken reductions” – the multi-subset transform could be a substantially harder problem not admitting a nearly linear-time algorithm. One might even be tempted to hypothesize that one cannot reduce the base of the exponential complexity below the constant 3. We refute this hypothesis:

► **Theorem 1.** *The multi-subset transform can be computed using $O(2.985^n)$ arithmetic operations.*

We obtain our result by a reduction to *rectangular matrix multiplication (RMM)*. The basic idea is to split the ground set U into two halves U_1 and U_2 and divide the product over $i \in T$ into two smaller products accordingly. In this way we can view (1) as a matrix product of dimensions $2^{|U_1|} \times 2^{|U|} \times 2^{|U_2|}$. The two rectangular matrices are sparse, with at most $6^{n/2} = O(2.4495^n)$ non-zero elements out of the total $8^{n/2}$. The challenge is to exploit the sparsity. Known algorithms for general sparse matrix multiplication [19, 11] turn out to be insufficient for getting beyond the $O(3^n)$ bound (see Section 2.1 for details). Fortunately, in our case the sparsity occurs in a special, structured form that enables better control of zero-entries, and thereby a more efficient reduction to dense RMM. To get the best available constant base in the exponential bound, we call upon the recently improved fast RMM algorithms [7].

1.1 Application to weighted counting of acyclic digraphs

Let a_n be the number of labeled acyclic digraphs on n nodes. Robinson [14] and Harary and Palmer [10], independently discovered the following inclusion–exclusion recurrence:

$$a_n = \sum_{s=1}^n (-1)^{s-1} \binom{n}{s} 2^{s(n-s)} a_{n-s}.$$

To see why the formula holds, view s as the number of sinks (i.e., nodes with no out-neighbors), each of which can choose its in-neighbors freely from the remaining $n - s$ nodes.

Tian and He [16] generalized the recurrence to weighted counting of acyclic digraphs on a given set of n nodes V . Now every acyclic digraph D on V is assigned a *modular weight*, that is, a real-valued weight $w(D)$ that factorizes into node-wise weights $w_i(D_i)$, where $D_i \subseteq V \setminus \{i\}$ is the set of in-neighbors of node i in D . This counting problem has applications particularly in Bayesian learning of Bayesian networks from data; the weighted count is the partition function of a statistical model that associates each node of the graph with a random variable, and evaluating the partition function is the main computational bottleneck [6, 16, 15]. Letting a_V denote the weighted sum of acyclic digraphs on V , we have

$$a_V = \sum_D \prod_{i \in V} w_i(D_i) = \sum_{\emptyset \neq S \subseteq V} (-1)^{|S|-1} \left(\prod_{i \in S} \sum_{D_i \subseteq V \setminus S} w_i(D_i) \right) a_{V \setminus S}. \quad (2)$$

The recurrence enables computing a_V using $O(3^n n)$ arithmetic operations [16].

We will apply Theorem 1 to lower the base of the exponential bound:

► **Theorem 2.** *The sum over acyclic digraphs with modular weights can be computed using $O(2.985^n)$ arithmetic operations.*

1.2 Related work

There are numerous previous applications of fast matrix multiplication algorithms to decision, optimization, and counting problems. Here we only mention a few that are most related to the present work.

Williams [17] employs fast *square* matrix multiplication to count all variable assignments that satisfy a given number of constraints, each involving at most two variables. By a simple reduction, this yields the fastest known algorithm for the Max-2-CSP problem. The present work is based on the same idea of viewing the product of a group of low-arity functions as a large matrix; this general idea is also studied in the doctoral thesis of the first author [13, Sects. 3.3 and 3.6], including reductions to *RMM*, however, without concrete applications.

Björklund, Kaski, and Kowalik [3] apply fast RMM to show the following: Given a nonnegative integer q and three mappings f, g, h from the subsets of an n -element set to some ring, one can sum up the products $f(A)g(B)h(C)$ over all pairwise disjoint triplets of q -sets A, B, C using $O(n^{3q\tau+c})$ ring operations, where $\tau < \frac{1}{2}$ and $c \geq 0$ are constants independent of q and n . Consequently, one can count the occurrences of constant-size paths (or any other small-pathwidth patterns) faster than in the “meet-in-the-middle time” [3]. While the involvement of set functions and set relations bear a resemblance to those in multi-subset transform, the reduction of Björklund et al. is based on solving an appropriately constructed system of linear equations, and is thus very different from the combinatorial approach taken in the present work.

2 Fast multi-subset transform: proof of Theorem 1

We will develop an algorithm for multi-subset transform in several steps. In Section 2.1 we give the basic reduction to RMM and the idea of splitting the sum over into several smaller sums. Then, in Section 2.2 we present a simple implementation of the splitting idea, and get our first below-3 algorithm. This algorithm is improved upon in Section 2.3, yielding the claimed complexity bound. We end this section by presenting a more sophisticated splitting scheme in Section 2.4. We have not succeeded to give a satisfactory analysis of its complexity. Yet, our numerical calculations suggest the bound $O(2.930^n)$.

We will denote by $\omega(k)$, for $k \geq 0$, the smallest value such that the product of an $N \times \lceil N^k \rceil$ matrix by an $\lceil N^k \rceil \times N$ can be computed using $O(N^{\omega(k)+\epsilon})$ arithmetic operations for any constant $\epsilon > 0$; for a formal definition of $\omega(k)$, see Gall and Urrutia [7]. Thus, the exponent of square matrix multiplication is $\omega := \omega(1)$.

We will make repeated use of the following facts about binomial coefficients:

► **Fact 3.** For integers $k \geq 1$ and $n \geq 2k$ we have

$$(2n)^{-1/2} b\left(\frac{k}{n}\right)^n \leq \binom{n}{k} \leq \sum_{j=0}^k \binom{n}{j} \leq b\left(\frac{k}{n}\right)^n = 2^{nH(k/n)},$$

where

$$b(x) := x^{-x}(1-x)^{x-1} \quad \text{and} \quad H(x) := \log_2 b(x), \quad x \in [0, 1].$$

This can be proven using Stirling’s approximation to factorials.

► **Fact 4.** Let n be a positive integer. The function $k \mapsto \binom{n}{k} 2^k$ is increasing in $[0, \frac{2}{3}n)$ and strictly decreasing in $[\frac{2}{3}n, n)$.

This can be proven by observing that the ratio $\binom{n}{k+1} 2^{k+1} / \binom{n}{k} 2^k$ equals $2(n-k)/(k+1)$, and is thus decreasing in k , and is greater or equal to 1 exactly when $k \leq \frac{2}{3}n - \frac{1}{3}$.

2.1 Basic reduction to rectangular matrix multiplication

Assume without loss of generality that n is even. Let us arbitrarily partition U into two disjoint sets U_1 and U_2 , both of size $h := n/2$. If $T \subseteq U$, denote by T_1 and T_2 respectively the intersections $T \cap U_1$ and $T \cap U_2$. Furthermore, write $N := 2^h$ so that $2^n = N^2$.

Armed with this notation, we write the multi-subset transform of set functions $(f_i)_{i \in U}$ as

$$g(T) = G(T_1, T_2) := \sum_{S \subseteq U} F_1(T_1, S) F_2(T_2, S), \quad T \subseteq U, \quad (3)$$

where we define

$$F_p(T_p, S) := [S \cap U_p \subseteq T_p] \prod_{i \in T_p} f_i(S), \quad p = 1, 2.$$

Here the Iverson's bracket notation $[Q]$ evaluates to 1 if Q is true, and to 0 otherwise.

We can write the representation (3) in terms of a matrix product as

$$G = F_1 F_2^\top,$$

where G is an $N \times N$ matrix indexed in $2^{U_1} \times 2^{U_2}$ and F_p is an $N \times N^2$ matrix indexed in $2^{U_p} \times 2^U$. As above, we will write the index pair in parentheses (not as subscripts).

Applying fast RMM without any further tricks already yields a somewhat competitive asymptotic complexity bound. To see this, recall that $\omega(k)$ denotes the exponent of RMM of dimensions $N \times \lceil N^k \rceil \times N$. Since $\omega(2) < 3.252$ [7], we get that G , and thus g , can be computed using $O(N^{3.252}) = O(3.087^n)$ arithmetic operations. If the lower bound $\omega(2) \geq 3$ was tight, we would achieve the bound $O(2.829^n)$.

So far, we have ignored the sparsity of the matrices F_p . An entry $F_p(T_p, S)$ is zero whenever the intersection $S_p = S \cap U_p$ is not contained in T_p . Thus, out of the $8^{n/2}$ entries of F_p , at most $3^h 2^h = 6^{n/2}$ are nonzero. In general, one can compute a matrix product of dimensions $r \times r^k \times r$ using $O(mr^{(\omega-1)/2+\epsilon})$ operations, provided that the matrices have at most $m \geq r^{(\omega+1)/2}$ non-zero entries, irrespective of k [11]. This result applies to our case, but with the best known upper bound for ω [8], it only yields a bound $O(3.108^n)$. A direct reduction to multiple multiplications of sparse square matrices [19] yields an even worse bound, $O(3.142^n)$ (calculations omitted). Output-sensitive sparse matrix multiplication algorithms [1] will not work either, as our output matrix is dense in general.

Luckily, in our case, we can make more efficient use of the sparsity. We will decompose the matrix product into a sum of smaller matrix products, as formulated by the following representation (the proof is trivial and omitted):

► **Lemma 5.** *Let $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_M\}$ be a set partition of 2^U . Let F_{pq} be the submatrix of F_p obtained by removing all columns but those in \mathcal{S}_q , for $p = 1, 2$ and $q = 1, 2, \dots, M$. Then*

$$G = \sum_{q=1}^M G_q, \quad \text{where } G_q = F_{1q} F_{2q}^\top.$$

We will also apply this decomposition after removing some rows from the matrices F_{pq} . Then the index sets may be different for different G_q . To properly define the entry-wise addition in these cases, we simply make the convention that the missing entries equal zero.

To employ a fast RMM algorithm we will call a function $\text{FAST-RMM}(\mathcal{T}_1, \mathcal{S}, \mathcal{T}_2)$. The function returns the product $E_1 E_2^\top$, where each E_p is obtained from F_p by only keeping the rows \mathcal{T}_p and the columns \mathcal{S} . Note that we do not show the input matrices explicitly in the function call, as the submatrices will always be extracted from F_1 and F_2 .

■ **Algorithm 1** The COLUMNS algorithm for the multi-subset transform.

```

function COLUMNS-DIRECTLY( $\mathcal{S}$ )
1   $G[T] \leftarrow 0$  for all  $T \subseteq U$ 
2  for  $S \in \mathcal{S}$ 
3      for  $T \subseteq U$  s.t.  $S \subseteq T$ 
4           $G[T] \leftarrow G[T] + F_1(T_1, S)F_2(T_2, S)$ 
5  return  $G$ 

```

```

Algorithm COLUMNS( $(f_i)_{i \in U}$ )
1   $G[T] \leftarrow 0$  for all  $T \subseteq U$ 
2  select  $\sigma \in (\frac{1}{3}, \frac{1}{2})$ 
3   $\mathcal{S}_1 \leftarrow \{S \subseteq U : |S| \leq \sigma n\}$ 
4   $G \leftarrow G + \text{FAST-RMM}(2^{U_1}, \mathcal{S}_1, 2^{U_2})$ 
5   $G \leftarrow G + \text{COLUMNS-DIRECTLY}(2^U \setminus \mathcal{S}_1)$ 
6  return  $G$ 

```

2.2 A simple below-3 algorithm

We apply Lemma 5 with $M = 2$ and split the columns to those that are smaller than a threshold σn and to those that are at least as large:

$$\mathcal{S}_1 = \{S \subseteq U : |S| < \sigma n\} \quad \text{and} \quad \mathcal{S}_2 = \{S \subseteq U : |S| \geq \sigma n\}.$$

We assume σn is an integer and that $\frac{1}{3} < \sigma < \frac{1}{2}$. We will optimize the parameter σ later. The idea is to call fast RMM only for summing over the columns \mathcal{S}_1 and to handle the remaining columns in a brute-force manner. The algorithm COLUMNS is given in Algorithm 1.

Consider first the computation of the matrix G_1 . We compute G_1 using fast RMM. The computational complexity depends on the number of columns in the matrices F_{11} and F_{21} . Letting C be the number of columns, the required number of operations for the matrix multiplication of dimensions $N \times C \times N$ is $O(N^{\omega(k)})$, where $k = \log_N C$. We have

$$C = |\mathcal{S}_1| = \sum_{s=0}^{\sigma n} \binom{n}{s} \leq b(\sigma)^n, \quad (4)$$

where the inequality follows by Fact 3.

Consider then the computation of the matrix G_2 . To compute $G_2(T)$, for $T \subseteq U$, it suffices to compute the sum of the products $F_1(T_1, S)F_2(T_2, S)$ over all columns $S \subseteq T$ whose size is at least σn . Thus, the required number pairs (S, T) to be considered is at most

$$B := \sum_{s=\sigma n}^n \binom{n}{s} 2^{n-s} \leq n \binom{n}{\sigma n} 2^{n(1-\sigma)} \leq n(2^{1-\sigma} b(\sigma))^n \quad (5)$$

where the penultimate inequality follows by Fact 4 (since $1 - \sigma < \frac{2}{3}$) and the last by Fact 3.

Let us finally combine the bounds in (4) and (5).

► **Proposition 6.** *For any $\epsilon > 0$, the number of operations required by COLUMNS is*

$$O\left(2^{n(\omega(2H(\sigma))+\epsilon)/2} + n2^{n(1-\sigma+H(\sigma))}\right).$$

It remains to choose σ so as to optimize the bound. Clearly the first term is increasing and the second term is decreasing in σ . Thus, the bound is (asymptotically) minimized by choosing a σ that makes $\omega(2H(\sigma))$ equal to $2(1 - \sigma + H(\sigma))$. There are two obstacles to implement this idea: first, we only know upper bounds for $\omega(k)$, for various k ; second, no closed-form expression is known for the best upper bounds – upper bounds for $\omega(k)$ have been computed and reported only at some points k [7].

Due to these complications, we resort to the following facts:

► **Fact 7** ([7]). *The exponent of RMM satisfies $\omega(1.75) \leq 3.021591$.*

► **Fact 8**. *Let $k > 0$ and $r \geq 0$. The exponent of RMM satisfies $\omega(k + r) \leq \omega(k) + r$.*

(This follows by reducing the larger RMM instance trivially to multiple smaller instances.)

Combining these two facts yields an upper bound:

$$\omega(2H(\sigma)) \leq \omega(1.75) + 2H(\sigma) - 1.75 \leq 1.271591 + 2H(\sigma).$$

Now, solving $1.271591 + 2H(\sigma) = 2(1 - \sigma + H(\sigma))$ gives

$$\sigma = 1 - 1.271591/2 = 0.3642045.$$

With this choice of σ the complexity bound becomes $O(2.994^n)$.

2.3 A faster below-3 algorithm

Next we give a slightly faster algorithm to compute G_1 . This will allow us to choose a larger threshold σ , thus also rendering the computation of G_2 faster.

Instead of computing G_1 directly using fast RMM, we now compute some rows and columns of G_1 in a brute-force manner and only apply fast RMM to the remaining smaller matrix. Specifically, the algorithm only calls fast RMM to compute the entries $G_1(T_1, T_2)$ where the sizes of T_1 and T_2 exceed τh . We assume that τh is an integer and that $\tau \in (\frac{1}{2}, \frac{2}{3})$. We will optimize the parameter τ together with σ later. The algorithm ROWS&COLUMNS is given in Algorithm 2. The correctness of the algorithm being clear, we proceed to analysing the complexity in terms of the required number of arithmetic operations.

Consider first the computation of an entry $G_1(T_1, T_2)$ where $|T_1| \leq \tau h$. The number of pairs (S, T) satisfying $S \subseteq T \subseteq U$ and $|T_1| \leq \tau h$ is given by

$$B' := 3^h \sum_{t=0}^{\tau h} \binom{h}{t} 2^t \leq 3^h h \binom{h}{\tau h} 2^{\tau h} \leq h(3 \cdot 2^\tau b(\tau))^h; \quad (6)$$

the penultimate inequality follows by Fact 4 (since $\tau < \frac{2}{3}$) and the last inequality by Fact 3.

Similarly, computing the entries $G_1(T_1, T_2)$ for all $T_1 \subseteq U_1$ and $T_2 \subseteq U_2$ such that $|T_2| \leq \tau h$ requires at most B' additions and multiplications.

It remains to compute the entries $G_1(T_1, T_2)$ for $T_1 \subseteq U_1$ and $T_2 \subseteq U_2$ such that $|T_1|, |T_2| > \tau h$. This can be computed as a product of two matrices (submatrices of F_1 and F_2^\top) whose sizes are at most $R \times C$ and $C \times R$, where C is as before and

$$R := \sum_{j=\tau h+1}^h \binom{h}{j} \leq b(\tau)^h, \quad (7)$$

where the inequality follows by Fact 3 (since $\tau > \frac{1}{2}$).

Let us combine the bounds in (6) and (7):

■ **Algorithm 2** The ROWS&COLUMNS algorithm for the multi-subset transform.

```

function ROWS-TRIMMED( $\tau, \mathcal{S}$ )
1   $G[T] \leftarrow 0$  for all  $T \subseteq U$ 
2   $\mathcal{T}_p \leftarrow \{T_p \subseteq U_p : |T_p| > \tau h\}$  for  $p \leftarrow 1, 2$ 
3  for  $S \subseteq T \subseteq U$  s.t.  $S \in \mathcal{S}$  and  $(T_1 \notin \mathcal{T}_1$  or  $T_2 \notin \mathcal{T}_2)$ 
4       $G[T] \leftarrow G[T] + F_1(T_1, S)F_2(T_2, S)$ 
5   $G \leftarrow G + \text{FAST-RMM}(\mathcal{T}_1, \mathcal{S}, \mathcal{T}_2)$ 
6  return  $G$ 

```

```

Algorithm ROWS&COLUMNS( $(f_i)_{i \in U}$ )
1   $G[T] \leftarrow 0$  for all  $T \subseteq U$ 
2  select  $\sigma \in (\frac{1}{3}, \frac{1}{2})$  and  $\tau \in (\frac{1}{2}, \frac{2}{3})$ 
3   $\mathcal{S}_1 \leftarrow \{S \in U : |S| \leq \sigma n\}$ 
4   $G \leftarrow G + \text{ROWS-TRIMMED}(\tau, \mathcal{S}_1)$ 
5   $G \leftarrow G + \text{COLUMNS-DIRECTLY}(2^U \setminus \mathcal{S}_1)$ 
6  return  $G$ 

```

► **Proposition 9.** For any $\epsilon > 0$, the number of operations required by ROWS&COLUMNS is

$$O\left(n(3 \cdot 2^\tau b(\tau))^{n/2} + b(\tau)^{\omega(k) + \epsilon n/2} + n(2^{1-\sigma} b(\sigma))^n\right), \quad \text{where } k = 2 \log_{b(\tau)} b(\sigma).$$

To set the parameters σ and τ , we resort to the bound $\omega(k) \leq 1.271591 + k$ (Fact 7 and Fact 8). Balancing the latter two terms in the bound yields the equation

$$(1.271591 + k)H(\tau) = 2(1 - \sigma + H(\sigma)).$$

Equivalently, $1.271591 \cdot H(\tau) = 2(1 - \sigma)$. Solving for σ and equating the first and the third term in the bound leaves us the equation

$$\log_2 3 + \tau + H(\tau) = 1.271591 \cdot H(\tau) + 2H(1 - 0.6357955 \cdot H(\tau)).$$

By numerical calculations we find one solution in the valid range, $\tau \approx 0.59777$, and correspondingly $\sigma \approx 0.38185$. With these choices the complexity bound becomes $O(2.985^n)$. This completes the proof of Theorem 1.

2.4 A covering based algorithm

The previous algorithms were based on pruning some columns and rows of the matrices F_1 and F_2 , and applying fast RMM to the remaining multiplication of two reduced matrices. Now, we take a different approach and reduce the original problem instance into multiple, smaller RMM instances applying Lemma 5 with some $M > 2$. To this end, we cover – in the sense of a set cover – the columns by multiple groups such that the columns in one group contain a large block of zero entries (in the same set of rows) in the matrices F_1 and F_2 .

It will be convenient to consider sets of fixed sizes. For a set V and a nonnegative integer s , write $\binom{V}{s}$ for the set of all s -element subsets of V . Let $s_1, s_2 \in \{0, 1, \dots, h\}$ fix the sizes of the intersection of a column with the sets U_1 and U_2 . We wish to cover the set (of set pairs) $\binom{U_1}{s_1} \times \binom{U_2}{s_2}$ by a small number of sets of the form $\binom{K_1}{s_1} \times \binom{K_2}{s_2}$, where the sets K_1 and K_2 are of some fixed sizes $k_1 \geq s_1$ and $k_2 \geq s_2$. The following classic result [5] shows that this covering design problem has an efficient solution:

► **Theorem 10** ([5]). *Let $c(v, k, s)$ be the minimum number of subsets of $\{1, 2, \dots, v\}$ of size k such that every subset of size $s \leq k$ is contained by at least one of the sets. We have*

$$c(v, k, s) \binom{k}{s} \binom{v}{s}^{-1} \leq 1 + \ln \binom{k}{s}.$$

In particular, $c(v, k, s)$ is within the factor k of the obvious lower bound $\binom{v}{s} \binom{k}{s}^{-1}$.

► **Remark 11.** Although the work needed for constructing a covering does not contribute to the number of operations in the ring \mathcal{R} , a remark is in order if one is interested in the required number of other operations. The authors are not aware of any deterministic algorithm for constructing an optimal covering in time polynomial in $\binom{v}{k} + \binom{v}{s}$, while asymptotically optimal randomized polynomial-time algorithms are known [9].

Fortunately, for our purposes it suffices to run the well known greedy algorithm that iteratively picks a set that covers the largest number of yet uncovered elements. It finds a set cover whose size is within a logarithmic factor of the optimum, which is sufficient in our context. Furthermore, it can be implemented to run in time linear in the input size [4, Ex. 35.3–3], which is $\binom{v}{k} \binom{k}{s} \leq 3^v$ in our case (with $v = h = n/2$).

From now on, we assume that for $p = 1, 2$ we are given a set family $\mathcal{K}_p \subseteq \binom{U_p}{k_p}$ that has the desired coverage property, i.e., $\{\binom{K_p}{s_p} : K_p \in \mathcal{K}_p\}$ is a set cover of $\binom{U_p}{s_p}$, so that for every column $S \subseteq U$ satisfying $|S_1| = s_1, |S_2| = s_2$ there is a pair $(K_1, K_2) \in \mathcal{K}_1 \times \mathcal{K}_2$ such that $S_1 \subseteq K_1, S_2 \subseteq K_2$. In what follows, we will assume that some appropriate values of k_1, k_2 are chosen based on s_1, s_2 ; we will return back to the issue of finding good values at the end of this subsection.

For each pair (K_1, K_2) , we construct a submatrix E_1 of F_1 as follows: remove from F_1 all columns S not covered by (K_1, K_2) , and all rows T_1 whose intersection with K_1 contains less than s_1 elements (as otherwise we cannot have $S_1 \subseteq T_1$ and the entry $F_1(T_1, S)$ vanishes). We construct a matrix E_2 analogously by removing columns and rows from F_2 . The dimensions of the matrix product $E_1 E_2^\top$ are $R_1 \times C' \times R_2$, where

$$R_1 := \sum_{j=s_1}^{k_1} \binom{k_1}{j} 2^{h-k_1}, \quad C' := \binom{k_1}{s_1} \binom{k_2}{s_2}, \quad R_2 := \sum_{j=s_2}^{k_2} \binom{k_2}{j} 2^{h-k_2}.$$

Algorithm COVER-COLUMNS, given in Algorithm 3, organizes the reduction to multiple RMM instances like this using Lemma 5. Specifically, from the set cover of the columns it extracts a set partition by trivially keeping track of the already covered columns.

To analyze the complexity of the algorithm, let us first bound the dimensions R_1, C' , and R_2 for fixed s_1, s_2, k_1, k_2 . We aim at bounds of the form N^α for some $0 < \alpha < 2$, and therefore parameterize the set sizes as

$$s_p = \sigma_p h \quad \text{and} \quad k_p = \kappa_p h, \quad p = 1, 2.$$

Thus $0 \leq \sigma_p \leq \kappa_p \leq 1$. In what follows, we let σ_p/κ_p evaluate to 0 if $\sigma_p = \kappa_p = 0$.

► **Lemma 12.** *We have*

$$R_1 \leq N^{\beta_1}, \quad C' \leq N^{\alpha_1 + \alpha_2}, \quad R_2 \leq N^{\beta_2},$$

where

$$\alpha_p := \kappa_p H\left(\frac{\sigma_p}{\kappa_p}\right) \quad \text{and} \quad \beta_p := 1 - \kappa_p + \kappa_p H\left(\max\left\{\frac{\sigma_p}{\kappa_p}, \frac{1}{2}\right\}\right), \quad p = 1, 2. \quad (8)$$

■ **Algorithm 3** The COVER-COLUMNS algorithm for the multi-subset transform.

Algorithm COVER-COLUMNS($((f_i)_{i \in U})$)

```

1   $G[T] \leftarrow 0$  for all  $T \subseteq U$ 
2   $\mathcal{C} \leftarrow \emptyset$  // Already covered columns
3  for  $(s_1, s_2) \in \{0, 1, \dots, h\}^2$ 
4      select  $k_1$  and  $k_2$ 
5       $\mathcal{K}_p \leftarrow \text{COVERING-DESIGN}(s_p, k_p, U_p)$  for  $p \leftarrow 1, 2$ 
6      for  $(K_1, K_2) \in \mathcal{K}_1 \times \mathcal{K}_2$ 
7           $\mathcal{S} \leftarrow \{S_1 \cup S_2 : S_1 \in K_1 \text{ and } S_2 \in K_2\}$ 
8           $G \leftarrow G + \text{ROWS-TRIMMED}(0, \mathcal{S} \setminus \mathcal{C})$  // Trim only all-zero rows
9           $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{S}$ 
10 return  $G$ 

```

Proof. The bound for C' follows directly from the definitions of σ_p , κ_p , α_p and from Fact 3.

For the bound on R_1 (equivalently R_2), suppose first that $\kappa_1 \geq 2\sigma_1$. Then using the simple inequality $\sum_{j=s_1}^{k_1} \binom{k_1}{j} \leq 2^{k_1} = N^{\kappa_1 H(1/2)}$ gives the claimed bound. Otherwise, $\kappa_1 \leq 2\sigma_1$ and thus, by Fact 3, $\sum_{j=s_1}^{k_1} \binom{k_1}{j} \leq 2^{k_1 H(1-\sigma_1/\kappa_1)} = N^{\kappa_1 H(\sigma_1/\kappa_1)}$, implying the claimed bound. ◀

It remains to turn the bounds on the dimensions to a bound on the complexity of the corresponding RMM and sum up these bounds over the multiple matrix multiplication tasks.

► **Proposition 13.** *For any $\epsilon > 0$, the number of operations required by COVER-COLUMNS is $O(2^{(\gamma+\epsilon)n/2})$, where*

$$\gamma := \max_{\substack{0 \leq \sigma_1 \leq 1 \\ 0 \leq \sigma_2 \leq 1}} \min_{\substack{\sigma_1 \leq \kappa_1 \leq 1 \\ \sigma_2 \leq \kappa_2 \leq 1}} H(\sigma_1) + H(\sigma_2) - \alpha_1 - \alpha_2 + \beta_1 + \beta_2 + \beta_* \left(\omega \left(\frac{\alpha_1 + \alpha_2}{\beta_*} \right) - 2 \right), \quad (9)$$

with α_p and β_p as defined in (8), and $\beta_* := \min\{\beta_1, \beta_2\}$.

Proof. Let $\epsilon > 0$.

Consider first the complexity of a single matrix multiplication with fixed σ_p, κ_p , for $p = 1, 2$. By Lemma 12 we obtain an upper bound by taking $N^{\max\{\beta_1, \beta_2\} - \beta_*} = N^{\beta_1 + \beta_2 - 2\beta_*}$ matrix multiplications of dimensions $N^{\beta_*} \times N^{\alpha_1 + \alpha_2} \times N^{\beta_*}$. This gives us the upper bound $O(N^{\beta_1 + \beta_2 + \beta_* (\omega(k) - 2) + \epsilon/2})$, where $k = (\alpha_1 + \alpha_2)/\beta_*$. Note that we used only a half of ϵ – we will need the other half for tolerating a nonzero underestimation that is due to minimizing κ_p over reals. We will return to this issue at the end of the proof.

Consider then the number of matrix multiplications for fixed s_p, k_p , for $p = 1, 2$. By Theorem 10 and by the approximation ratio of the greedy algorithm, the number is at most

$$\begin{aligned} n^4 \binom{h}{s_1} \binom{h}{s_2} \binom{k_1}{s_1}^{-1} \binom{k_2}{s_2}^{-1} &\leq n^5 b(\sigma_1)^h b(\sigma_2)^h b(\sigma_1/\kappa_1)^{-\kappa_1 h} b(\sigma_2/\kappa_2)^{-\kappa_2 h} \\ &= n^5 N^{H(\sigma_1) + H(\sigma_2) - \alpha_1 - \alpha_2}. \end{aligned}$$

Here we used Fact 3 to bound the binomial coefficients, observing that $(2k_1)^{1/2} (2k_2)^{1/2} \leq n$.

Now, combine the above two bounds, recall that $N = 2^{n/2}$, and observe that replacing the sum over (s_1, s_2) by the maximum over (σ_1, σ_2) is compensated by adding a factor of n^2 to the bound. The algorithm can select optimal k_1 and k_2 by optimizing the upper bound,

29:10 Fast Multi-Subset Transform

which costs yet another factor of n^2 . Due to the constant ϵ in the exponent, we can ignore the $n^{O(1)}$ factor in the asymptotic complexity bound.

To complete the proof, we show that for any values of σ_p and κ_p (hence also for the optimal values) and for any large enough integer h , there are rational numbers $\kappa'_p \geq \sigma_p$ such that (i) $\kappa'_p h$ are integers and (ii) $\Gamma(\sigma_1, \sigma_2, \kappa'_1, \kappa'_2) \leq \Gamma(\sigma_1, \sigma_2, \kappa_1, \kappa_2) + \epsilon/2$, where

$$\Gamma(\sigma_1, \sigma_2, \kappa_1, \kappa_2) := H(\sigma_1) + H(\sigma_2) + \beta_1 + \beta_2 + \beta_* \left(\omega \left(\frac{\alpha_1 + \alpha_2}{\beta_*} \right) - \left(\frac{\alpha_1 + \alpha_2}{\beta_*} \right) - 2 \right). \quad (10)$$

Note that we rearranged some terms in (9), for a reason that will be revealed in a moment.

We will consider two cases: either σ_1 or σ_2 is near the boundary values 0 or 1, or both are in $[c, 1 - c]$, where $c > 0$ is a small constant. We choose $c < \frac{1}{2}$ such that if $0 \leq \sigma_1 < c$ or $1 - c < \sigma_1 \leq 1$, then regardless of σ_2 ,

$$\Gamma(\sigma_1, \sigma_2, 1, 1) \leq \omega(1) + \epsilon/2,$$

and symmetrically for σ_2 . To see that this is possible, observe first that at $\kappa_1 = \kappa_2 = 1$ we have $\alpha_1 = H(\sigma_1)$, $\alpha_2 = H(\sigma_2)$, and thus

$$\begin{aligned} \Gamma(\sigma_1, \sigma_2, 1, 1) &= \beta_1 + \beta_2 + \beta_* \left(\omega \left(\frac{\alpha_1 + \alpha_2}{\beta_*} \right) - 2 \right) \\ &\leq \beta_1 + \beta_2 + \beta_* \left(\omega \left(\frac{\alpha_*}{\beta_*} \right) + \frac{\alpha_1 + \alpha_2 - \alpha_*}{\beta_*} - 2 \right), \end{aligned}$$

where $\alpha_* := \alpha_p$ if $\beta_* = \beta_p$. Observe that $\alpha_* \leq \beta_*$. Since $\omega(1) - 2 \geq 0$ and $\alpha_1, \alpha_2, \beta_1, \beta_2 \leq 1$,

$$\Gamma(\sigma_1, \sigma_2, 1, 1) \leq \alpha_1 + \alpha_2 - \alpha_* + \beta_1 + \beta_2 + \omega(1) - 2 \leq \omega(1) + H(\sigma_1).$$

For the latter inequality we used the facts that $\alpha_* = \alpha_2$ if $\sigma_1 < c$ and that $\beta_1 = H(\sigma_1)$ if $\sigma_1 > 1 - c$. Finally, we observe that $H(\sigma_1)$ tends to 0 when σ_1 tends to 0 or 1.

On the other hand, we have the lower bound $\Gamma(\frac{1}{2}, \frac{1}{2}, \kappa_1, \kappa_2) \geq 2 + \beta_1 + \beta_2 - \beta_* \geq 2.5 > \omega(1)$, since $\omega(z) - z \geq 1$ and $\beta_p = 1 - \kappa_p + \kappa_p H(1/(2\kappa_p)) \geq \kappa_p \geq \frac{1}{2}$; here we used the fact that $H(x) \geq 2 - 2x$ for $x \in [\frac{1}{2}, 1]$.

We may thus restrict our attention to the domain

$$\Lambda_c := \{(\sigma_1, \sigma_2, \kappa_1, \kappa_2) : c \leq \sigma_1, \sigma_2 \leq 1 - c, \sigma_1 \leq \kappa_1 \leq 1, \sigma_2 \leq \kappa_2 \leq 1\}.$$

We now show that Γ is continuous on Λ_c . Observe first that the functions H , α_p , and β_p are continuous on Λ_c (as $\kappa_p > c$). We also have that β_* is continuous and strictly positive (as $\sigma_p \leq 1 - c$) and that $z \mapsto \omega(z)$ is continuous (as $|\omega(z + \delta) - \omega(z)| \leq \delta$ for all $\delta > 0$).

Since the domain Λ_c is compact, we have that Γ is uniformly continuous on Λ_c . This in turn implies that there is a $\delta_\epsilon > 0$ such that (ii) holds whenever $|\kappa'_p - \kappa_p| < \delta_\epsilon$, implying that we can make both (i) and (ii) hold for all $h > 1/\delta_\epsilon$ by putting $\kappa'_p := \lceil \kappa_p h \rceil / h$. \blacktriangleleft

Now we know that the complexity of the algorithm is $O(2^{(\gamma+\epsilon)n/2})$, but we do not know how large γ is. Unlike for the simpler algorithms given in the previous subsections, we cannot just select some values of the parameters σ_p and κ_p and bound γ from above by $\Gamma(\sigma_1, \sigma_2, \kappa_1, \kappa_2)$, as defined in (10), for we do not know the maximizing values of σ_p . Since Γ is uniformly continuous on the domain Λ_c , one could in principle prove any fixed strict upper bound on γ with a sufficiently large, finite computation. While at the present time the authors have not produced such a proof, evaluations of $\Gamma(\sigma_1, \sigma_2, \kappa_1, \kappa_2)$ at various values of the four parameters suggest the following:

► Conjecture 14. *The number of operations required by COVER-COLUMNS is $O(2.930^n)$.*

3 Fast weighted counting of acyclic digraphs: proof of Theorem 2

Let us write the inclusion–exclusion recurrence (2) as a multi-subset transform:

► **Lemma 15.** *Without loss of generality, suppose $0 \notin V$. Let $0 \in T \subseteq V \cup \{0\}$ and*

$$g(T) = \sum_{S \subseteq T} \prod_{i \in T} f_i(S),$$

where

$$f_i(S) = \begin{cases} 0 & \text{if } 0 \notin S \text{ or } |S| = |T|; \\ (-1)^{|S|-1} a_{S \setminus \{0\}} & \text{else if } i = 0; \\ \sum_{D_i \subseteq S \setminus \{0\}} w_i(D_i) & \text{else if } i \notin S; \\ 1 & \text{otherwise.} \end{cases}$$

Then $a_{T \setminus \{0\}} = (-1)^{|T|} g(T)$.

Proof. Because the summand vanishes unless $0 \in S \neq T$ and because $f_i(S) = 1$ unless $i \in \{0\} \cup (T \setminus S)$, we have

$$\begin{aligned} (-1)^{|T|} g(T) &= (-1)^{|T|} \sum_{0 \in S \subsetneq T} f_0(S) \prod_{i \in T \setminus S} f_i(S) \\ &= \sum_{0 \in S \subsetneq T} (-1)^{|T|+|S|-1} a_{S \setminus \{0\}} \prod_{i \in T \setminus S} \sum_{D_i \subseteq S \setminus \{0\}} w_i(D_i). \end{aligned}$$

Writing in terms of $T' := T \setminus \{0\}$ and $S' := T \setminus S$, and observing that $|S|$ and $-|S|$ have the same parity,

$$(-1)^{|T|} g(T) = \sum_{\emptyset \neq S' \subsetneq T'} (-1)^{|S'|-1} a_{T' \setminus S'} \prod_{i \in S'} \sum_{D_i \subseteq T' \setminus S'} w_i(D_i) = a_{T'}.$$

The last equality follows immediately from (2). ◀

It remains to organize the computations so that when computing a_T for some $T \subseteq V$, the values a_S have already been computed for all $S \subsetneq T$. To this end, we proceed in increasing order by $|T|$: for each $t = 1, 2, \dots, n$ in this order we simultaneously compute the values a_T for all $T \in \binom{V}{t}$ by calling the fast multi-subset transform, as detailed in algorithm SUM-ACYCLIC-DIGRAPHS given in Algorithm 4. As we only need n calls, the asymptotic complexity bound (with a rounded constant base of the exponential) remains valid.

References

- 1 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *12th International Conference on Database Theory, ICDT '09*, pages 121–126. ACM, 2009.
- 2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *39th ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007.
- 3 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. *ACM Trans. Algorithms*, 13(4):48:1–48:26, 2017.
- 4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- 5 Paul Erdős and Joel Spencer. *Probabilistic Methods in Combinatorics*. Akadémiai Kiadó, Budapest, 1974.

■ **Algorithm 4** The SUM-ACYCLIC-DIGRAPHS algorithm for the sum over acyclic digraphs with modular weights. FAST-MULTI-SUBSET-TRANSFORM($(f_i)_{i \in U}$) returns the multi-subset transform of $(f_i)_{i \in U}$.

Algorithm SUM-ACYCLIC-DIGRAPHS($(w_i)_{i \in V}$)

```

1   $a[\emptyset] \leftarrow 1$ ;  $a[S] \leftarrow 0$  for all  $\emptyset \neq S \subseteq V$ 
2  compute  $f_i[S \cup \{0\}] \leftarrow \sum_{X \subseteq S} w_i(X)$  for all  $i \in V$ ,  $S \subseteq V$  using fast zeta transform
3   $f_i[S] \leftarrow 0$  for all  $i \in V$ ,  $S \subseteq V$ .
4   $f_i[S \cup \{0\}] \leftarrow 1$  for all  $i \in S \subseteq V$ .
5  for  $t \leftarrow 1$  to  $n$ 
6      for  $S \in \binom{V}{t-1}$ 
7           $f_0[S \cup \{0\}] \leftarrow (-1)^{|S|-1} a[S]$ 
8       $g \leftarrow$  FAST-MULTI-SUBSET-TRANSFORM( $(f_i)_{i \in V \cup \{0\}}$ )
9      for  $T \in \binom{V}{t}$ 
10          $a[T] \leftarrow (-1)^{|T|+1} g[T \cup \{0\}]$ 
11 return  $a[V]$ 

```

- 6 Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Mach. Learn.*, 50(1-2):95–125, 2003.
- 7 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1029–1046. SIAM, 2018.
- 8 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14*, pages 296–303. ACM, 2014.
- 9 Daniel M. Gordon, Oren Patashnik, Greg Kuperberg, and Joel H. Spencer. Asymptotically optimal covering designs. *Journal of Combinatorial Theory, Series A*, 75(2):270–280, 1996.
- 10 Frank Harary and Edgar M. Palmer. *Graphical Enumeration*, pages 191–194. Academic Press, 1973. Section 8.8 “Acyclic Digraphs”.
- 11 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Twenty-Second Annual Symposium on Computational Geometry, SCG '06*, pages 52–60. ACM, 2006.
- 12 Robert Kennes. Computational aspects of the Möbius transformation of graphs. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):201–223, 1992.
- 13 Mikko Koivisto. *Sum-Product Algorithms for the Analysis of Genetic Risks*. PhD thesis, Department of Computer Science, University of Helsinki, January 2004.
- 14 Robert W. Robinson. Counting labeled acyclic digraphs. In *New Directions in the Theory of Graphs*, pages 239–273. Academic Press, New York, 1973.
- 15 Topi Talvitie, Aleksis Vuoksenmaa, and Mikko Koivisto. Exact sampling of directed acyclic graphs from modular distributions. In *Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019*, pages 345–352, 2019.
- 16 Jin Tian and Ru He. Computing posterior probabilities of structural features in Bayesian networks. In *25th Conference on Uncertainty in Artificial Intelligence*, pages 538–547. AUAI Press, 2009.
- 17 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.
- 18 Frank Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, 1937.
- 19 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.

Exact Exponential Algorithms for Two Poset Problems

László Kozma

Freie Universität Berlin, Institute of Computer Science, Germany

<http://www.lkozma.net/>

laszlo.kozma@fu-berlin.de

Abstract

Partially ordered sets (posets) are fundamental combinatorial objects with important applications in computer science. Perhaps the most natural algorithmic task, given a size- n poset, is to compute its number of *linear extensions*. In 1991 Brightwell and Winkler showed this problem to be #P-hard. In spite of extensive research, the fastest known algorithm is still the straightforward $O(n2^n)$ -time dynamic programming (an adaptation of the Bellman-Held-Karp algorithm for the TSP). Very recently, Dittmer and Pak showed that the problem remains #P-hard for *two-dimensional* posets, and no algorithm was known to break the 2^n -barrier even in this special case. The question of whether the two-dimensional problem is easier than the general case was raised decades ago by Möhring, Felsner and Wernisch, and others. In this paper we show that the number of linear extensions of a two-dimensional poset can be computed in time $O(1.8286^n)$.

The related *jump number problem* asks for a linear extension of a poset, minimizing the number of neighboring *incomparable* pairs. The problem has applications in scheduling, and has been widely studied. In 1981 Pulleyblank showed it to be NP-complete. We show that the jump number problem can be solved (in arbitrary posets) in time $O(1.824^n)$. This improves (slightly) the previous best bound of Kratsch and Kratsch.

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

Keywords and phrases poset, linear extension, jump number, exponential time

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.30

Funding *László Kozma*: Research supported by DFG grant KO 6140/1-1.

1 Introduction

A *partially ordered set* (poset) $\mathcal{P} = (X, \prec)$ consists of a ground set X and an irreflexive and transitive binary relation \prec on X . A *linear extension* of \mathcal{P} is a total order on X that contains \prec . The main problem considered in this paper is to determine, given a poset \mathcal{P} on a ground set of size n , the number of linear extensions $\text{LE}(\mathcal{P})$ of \mathcal{P} . We refer to this counting problem as #LE. A poset can alternatively be seen as a transitive directed acyclic graph (DAG), where #LE asks for the number of topological orderings of the graph.

Posets are fundamental objects in combinatorics (for a detailed treatment we refer to the monographs [44, 37], [40, §3], [19, §8]) with several applications in computer science. For instance, every comparison-based algorithm (e.g. for sorting) implicitly defines a sequence of posets on the input elements, where each poset captures the pairwise comparisons known to the algorithm at a given time. An efficient sorter must find comparisons whose outcomes split the number of linear extensions in a balanced way. A central and long-standing open question in this area is whether a comparison with ratio (at worst) $1/3 : 2/3$ exists in every poset [5]; slightly weaker constant ratios are known to be achievable [23, 4].

Counting linear extensions (exactly or approximately) is a bottleneck in experimental work, e.g. when testing combinatorial conjectures. In computer science the #LE problem is relevant, besides the mentioned task of optimal comparison-based sorting, for learning



© László Kozma;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 30; pp. 30:1–30:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graphical models [45, 34], probabilistic ranking [46, 18, 30], reconstruction of partial orders from sequential data [31], convex rank tests [32], multimedia delivery in networks [1], and others.

The complexity of #LE has been thoroughly studied (see Linial [28] for an early reference). Lovász [29, §2.4] mentions the problem as a special case of polytope volume computation; Stanley [39] gives a broad overview of the polytope-formulation of #LE. Brightwell and Winkler [6] show that #LE is #P-hard, and thus unlikely to admit a polynomial-time solution. In fact, despite the significant attention the problem has received (e.g. the mentioned papers and references therein and thereof), the best upper bound on the running time remains $O(n2^n)$. This bound can be achieved via dynamic programming over the subsets of the ground set [26, 11], an approach¹ that closely resembles the Bellman-Held-Karp algorithm for the *traveling salesman problem (TSP)* [3, 20].

A bound of 2^n appears to be a natural barrier² for the running time of #LE, similarly to some of the most prominent combinatorial optimization problems (e.g. *set cover/hitting set*, *CNF-SAT*, *graph coloring*, *TSP*).³ We show that #LE can be solved faster when the input poset is *two-dimensional*. Dimension is perhaps the most natural complexity-measure of posets, and can be seen informally as a measure of the *nonlinearity* of a poset (see e.g. Trotter [44]). As one-dimensional posets are simply total orders, the first nontrivial case is dimension two. The structure of two-dimensional posets is, however, far from trivial. Posets in this class capture the *point-domination order* in the plane.

The question of the complexity of #LE in two-dimensional posets was raised in the 1980s by Möhring [33] and later by Felsner and Wernisch [17]. An even earlier mention of the problem is by Atkinson, Habib, and Urrutia, in a discussion of open problems concerning posets, cf. Rival [37, p. 481].

Efficient algorithms for #LE are known for various restricted classes of posets, e.g. *series-parallel* [33], *low treewidth* [25, 24, 15], *small width* [11], avoiding certain *substructures* [16], and others; see Möhring [33] for an early survey of tractable special cases. However, as the techniques used in these works rely on certain kinds of *sparsity* in the input, they are not applicable for the case of two-dimensional posets. It is easy to see that the latter may be arbitrarily dense, containing, for example, a complete bipartite graph of linear size. In fact, Dittmer and Pak [12] recently showed that #LE is #P-hard already for this class of inputs. Our first result is stated in the following theorem.

► **Theorem 1.** *The number of linear extensions of a two-dimensional poset of size n can be computed in time $O(1.8286^n)$.*

Our second result is an algorithm for the *jump number problem*. In this (optimization) problem a linear extension of \mathcal{P} is sought, such as to minimize the number of adjacent pairs of elements that are incomparable in \mathcal{P} (such pairs are called *jumps*). The problem is known to be NP-hard [36], and has been well-studied due to its applications in scheduling.

Similarly to #LE, the jump number problem can be solved by dynamic programming in time $2^n n^{O(1)}$. An improved algorithm with running time $O(1.8638^n)$ was given by Kratsch and Kratsch [27]. We also refer to their paper for further background and motivation for the problem. Improving the bound of Kratsch and Kratsch, we obtain the following result.

¹ A finer bound on the running time is $O(w \cdot |I|)$, where w is the *width* of the poset, and I is its set of *ideals* (i.e. downsets); in the worst case, however, this expression does not improve the given bound.

² We only study exact algorithms in this paper; for *approximating* $\text{LE}(\mathcal{P})$, fully polynomial-time randomized schemes are known [14, 7].

³ The *strong exponential time hypothesis* [21] states that a running time $O(c^n)$ with $c < 2$ is not achievable for CNF-SAT, and a similar barrier has been conjectured for set cover [10].

► **Theorem 2.** *The jump number problem can be solved in time $O(1.824^n)$.*

Note that in this case no assumption is made on the dimension of the input poset. Whether *jump number* remains NP-hard in two-dimensional posets is a long-standing open question [33, 8, 42].

Poset dimension. Formally, the dimension $\dim(\mathcal{P})$ of a poset $\mathcal{P} = (X, \prec)$ is the smallest number d of total orders, whose intersection is \mathcal{P} . In other words, if $\dim(\mathcal{P}) = d$, then there exists a collection of orders \prec_1, \dots, \prec_d (called *realizers* of \mathcal{P}), such that for all $x, y \in X$, we have $x \prec y$ if and only if $x \prec_k y$ for all $1 \leq k \leq d$.

Poset dimension was introduced by Dushnik and Miller in 1941 [13], and the concept has since been extensively studied; we refer to the monograph of Trotter dedicated to poset dimension theory [44]. Various kinds of *sparsity* of \mathcal{P} are known to imply upper bounds on $\dim(\mathcal{P})$ (see e.g. [22, 38] for recent results in a long line of such works). The converse is, in general, not true, as two-dimensional posets may already be arbitrarily dense, and are known not to have a characterisation in terms of finitely many forbidden substructures [2, 33].

The term *dimension* is motivated by the following natural geometric interpretation. Suppose \mathcal{P} is a d -dimensional, size- n poset with realizers \prec_1, \dots, \prec_d . The ground set can then be viewed as a set of n points in d -dimensional Euclidean space, with no two points aligned on any coordinate, such that the ordering of the points according to the k -th coordinate coincides with the order \prec_k , for all $1 \leq k \leq d$. The partial order \prec is then exactly the *point-domination order*, i.e. $x \prec y$ if and only if all d coordinates of y are larger than the corresponding coordinates of x . In this geometric view, a linear extension of a low-dimensional poset can be seen as a tour that visits all points, never moving behind the *Pareto front* of the already visited points.

Two-dimensional posets are particularly natural, as they are in bijection with permutations (the ranks of points by \prec_1 and \prec_2 can be seen respectively as the *index* and *value* of a permutation-entry). Swapping the two coordinates yields a *dual* poset, turning chains into antichains and vice versa. It follows that the complement of the *comparability graph* is itself a comparability graph, which is yet another exact characterization of two-dimensional posets.⁴ It is not hard to see that two-dimensional posets are exactly the *inclusion-posets* of intervals on a line.

Yet another interpretation of two-dimensional posets relates them to the weak Bruhat order on permutations. In this setting the number of linear extensions of a two-dimensional poset equals the number of permutations that are *reachable* from a given permutation π by a sequence of swaps between mis-sorted adjacent elements; a question of independent interest [17, 12].

2 Counting linear extensions in two-dimensional posets

Denote $[k] = \{1, \dots, k\}$. For a set Y with partial order \prec , let $\max(Y)$ denote the *set of maxima* of Y , i.e. the set of elements $x \in Y$ with the property that $x \prec y$ implies $y \notin Y$.

Let $\mathcal{P} = (X, \prec)$ be a size- n poset. To introduce the main elements of our #LE algorithm, we review first the classical $O(n2^n)$ time algorithm.

⁴ Given a poset $\mathcal{P} = (X, \prec)$, its *comparability graph* is $C(\mathcal{P}) = (X, E)$, where $\{x, y\} \in E$ if $x \prec y$ or $y \prec x$. The *width* of \mathcal{P} is the size of the largest *antichain* in \mathcal{P} , i.e. independent set in $C(\mathcal{P})$, and the *height* of \mathcal{P} is one less than the size of the largest *chain* in \mathcal{P} , i.e. clique in $C(\mathcal{P})$.

30:4 Exact Exponential Algorithms for Two Poset Problems

For all $Y \subseteq X$, let $\text{LE}(Y)$ denote the number of linear extensions of the subposet of \mathcal{P} induced by Y , and let $\text{LE}(\emptyset) = 1$. We recursively express $\text{LE}(Y)$ for all nonempty Y , by removing in turn all elements that can appear at the end of a total order on Y :

$$\text{LE}(Y) = \sum_{x \in \max(Y)} \text{LE}(Y \setminus \{x\}). \quad (1)$$

To compute $\text{LE}(\mathcal{P}) = \text{LE}(X)$, we evaluate recurrence (1), saving all intermediate entries $\text{LE}(Y)$ for $Y \subseteq X$. There are at most 2^n such entries, and computing each takes $O(n)$ time, once the results of the recursive calls are available. (With simple bookkeeping, $\max(Y)$ is available for all calls without additional overhead.)

2.1 A first improvement

A well-known observation is that when computing $\text{LE}(X)$ by (1) only those subproblems $Y \subseteq X$ arise where $y \in Y$ and $x \prec y$ imply $x \in Y$, i.e. the *downsets* of \mathcal{P} . In general, the number of downsets can be as high as 2^n , when \mathcal{P} consists of a single antichain. Nonetheless, we can give better bounds on the number of downsets, if necessary, by modifying the input poset \mathcal{P} .

Large matching case. An observation already made in previous works (e.g. [27]) is the following. Consider a size- m *matching* M in the comparability graph $\mathbf{C}(\mathcal{P})$, with matched edges $\{x_i, y_i\}$, where $x_i \prec y_i$, for all $i \in [m]$. Let W denote the set of vertices matched by M and let $A = X \setminus W$.

Then, the sets $Y \subseteq X$ where $Y \cap \{x_i, y_i\} = \{y_i\}$ for some $i \in [m]$ are not downsets and cannot be reached by recursive calls. The remaining sets can be partitioned as $T_0 \cup T_1 \cup \dots \cup T_m$, where $T_0 \subseteq A$ is an arbitrary subset of the unmatched vertices, and $T_i \in \{\emptyset, \{x_i\}, \{x_i, y_i\}\}$ for $i \in [m]$.

The number of sets of this form is $2^{n-2m} \cdot 3^m$. If $m = \alpha n$, this quantity equals $(2 \cdot (\frac{3}{4})^\alpha)^n$. When $\alpha \geq 1/3$, the number of subproblems is thus less than 1.8172^n , and the running time is within the required bounds.

Small matching case. Let us assume from now on that M is a *maximum* matching of size $m = \alpha n$ for $\alpha < 1/3$. The maximality of M implies that the unmatched vertices A form an independent set in $\mathbf{C}(\mathcal{P})$, i.e. an antichain of \mathcal{P} , of size $|A| = (1 - 2\alpha)n$. We assume $\alpha > 0$, as otherwise \mathcal{P} is a single antichain and the problem is trivial.

For $x \in A$, let $N(x)$ denote the *open neighborhood* of x in $\mathbf{C}(\mathcal{P})$, i.e. the set of elements in X that are comparable with x . Observe that $N(x) \cap A = \emptyset$ for all $x \in A$.

If $N(x) \cap A = \emptyset$ for an element $x \in W$, then we say that x is *incomparable* with A . Otherwise, if $x \prec y$ for some $y \in N(x) \cap A$, we say that x is *below* A , and if $y \prec x$, for some $y \in N(x) \cap A$, we say that x is *above* A . Observe that x cannot be both below and above A , as that would make two elements of A comparable, contradicting the fact that A is an antichain.

The sets $N(x)$ define a *partition* of A , where $x, y \in A$ are in the same class if and only if $N(x) = N(y)$. In general posets there can be as many as $\min\{2^{n-|A|}, |A|\}$ classes. The following lemma states that in two-dimensional posets the number of classes is much smaller.

► **Lemma 3.** *Let $\mathcal{P} = (X, \prec)$ be a size- n poset, with $\dim(\mathcal{P}) \leq 2$, and let $A \subseteq X$ be an antichain. Then, $N(\cdot)$ partitions A into at most $2(n - |A|)$ classes.*

Before proving Lemma 3, we show that it can be used to compute $\text{LE}(\mathcal{P})$ more efficiently. Let A_1, \dots, A_ℓ be the partition of A defined by $N(\cdot)$, and for each $i \in [\ell]$, denote $a_i = |A_i|$. Let x_i^k , where $i \in [\ell]$ and $k \in [a_i]$, be a *virtual element*, and let Q denote the set of all such virtual elements.

Construct a new poset $\mathcal{P}' = (X', \prec')$ as follows. Let $X' = W \cup Q$. In words, the ground set X' contains all vertices matched by M , and instead of the elements of the antichain A , it contains the virtual elements of Q . Observe that $|Q| = |A|$ and therefore $|X'| = |X|$.

The relation \prec' is defined as follows, covering all cases:

- if $x, y \in W$, then $x \prec' y \iff x \prec y$,
- if $x = x_i^p$ and $y = x_j^q$, then $x \prec' y \iff i = j$ and $p < q$,
- if $x \in W$ and $y = x_i^p$, then $x \prec' y \iff x \prec z$, for some $z \in A_i$,
- if $x = x_i^p$ and $y \in W$, then $x \prec' y \iff z \prec y$, for some $z \in A_i$.

In words, \prec' preserves the relation \prec between elements of W . Virtual elements with the same index i form a chain $x_i^1 \prec' \dots \prec' x_i^{a_i}$, for all $i \in [\ell]$. Virtual elements with different indices are incomparable. The relation between a virtual element x_i^k and an element $y \in W$ preserves the relation \prec between an arbitrary element $z \in A_i$ and y . The choice of z is indeed arbitrary, as the elements in A_i are by definition indistinguishable.

Intuitively, x_i^k is a placeholder for the element of A_i that appears as the k -th among all elements of A_i in some linear extension of \mathcal{P} . The sequence $x_i^1, \dots, x_i^{a_i}$ corresponds to an arbitrary permutation of the elements of A_i . This intuition is captured by the following statement.

► **Lemma 4.** *With the above definitions:*

$$\text{LE}(\mathcal{P}) = \prod_{i \in [\ell]} (a_i!) \cdot \text{LE}(\mathcal{P}').$$

Let us postpone proving Lemma 4 as well, and state our first algorithm, #LE-2D, as Algorithm 1. The algorithm constructs the poset \mathcal{P}' and computes its number of linear extensions using recurrence (1), then computes the correct count for \mathcal{P} via Lemma 4.

■ **Algorithm 1** Algorithm #LE-2D.

Input: Poset $\mathcal{P} = (X, \prec)$, where $|X| = n$.

Output: The number of linear extensions $\text{LE}(\mathcal{P})$ of \mathcal{P} .

- 1: Find a maximum matching M of $\mathcal{C}(\mathcal{P})$ with vertex set W .
 - 2: Let $A = X \setminus W$.
 - 3: Let A_1, \dots, A_ℓ be the partition of A by the neighborhoods in $\mathcal{C}(\mathcal{P})$.
 - 4: Let $a_i = |A_i|$ for $i \in [\ell]$.
 - 5: Construct $\mathcal{P}' = (X', \prec')$, as described.
 - 6: Compute $N = \text{LE}(\mathcal{P}')$ using (1).
 - 7: **return** $\prod_{i \in [\ell]} (a_i!) \cdot N$.
-

Analysis of the running time. Step 1 amounts to running a standard maximum matching algorithm (see e.g. [43]). Computing the partition in Step 3 takes linear time with careful data structuring. Steps 2,4,5,7 clearly take linear time overall.

The polynomial-time overhead of steps other than Step 6, as well as the polynomial factor in the analysis of (1) are absorbed in the exponential running time of Step 6, where we round the base of the exponential upwards. To derive a worst-case upper bound on the running time of Step 6, it only remains to bound the number of downsets of \mathcal{P}' .

Observe that the ground set X' can, by construction, be partitioned into chains. The matched vertices of W are partitioned into m chains $x_i \prec' y_i$, for $i \in [m]$, as before. The virtual elements of Q are partitioned into ℓ chains of lengths a_1, \dots, a_ℓ , where the i -th chain is $x_i^1 \prec' \dots \prec' x_i^{a_i}$. All downsets of \mathcal{P}' are then of the form $(T_1 \cup \dots \cup T_m) \cup (Q_1 \cup \dots \cup Q_\ell)$, where $T_i \in \{\emptyset, \{x_i\}, \{x_i, y_i\}\}$ for $i \in [m]$, and $Q_i = \{x_i^j : j \leq t_i\}$, for some threshold $0 \leq t_i \leq a_i$, for $i \in [\ell]$.

The number of such sets is $3^m \cdot \prod_{i \in [\ell]} (a_i + 1)$. Recall that $m = \alpha n$ and $|A| = (1 - 2\alpha)n$. The quantity $\prod_{i=1}^\ell (a_i + 1)$ is maximized when the values $a_i + 1$ are all equal, and thus equal to $(|A| + \ell)/\ell$, yielding the overall upper bound $\left(3^\alpha \left(\frac{(1-2\alpha)n}{\ell} + 1\right)^\ell\right)^n$. (Observe that $\ell \geq 1$ always holds.)

Since the quantity is increasing in ℓ , and $\ell \leq 2(n - |A|) = 4\alpha n$ by Lemma 3, we obtain the upper bound $\left(3^\alpha \left(\frac{1+2\alpha}{4\alpha}\right)^{4\alpha}\right)^n$. In the range of interest $0 < \alpha < 1/3$ the base achieves its maximum for $\alpha \approx 0.258$ at a value below 1.975, resulting in the bound $O(1.975^n)$ on the running time.

To reach the bound given in Theorem 1, we need further ideas. Let us first prove the two lemmas from which the correctness of the current algorithm and its analysis follow.

Proof of Lemma 4. Let $q = \prod_{i=1}^\ell (a_i!)$. We describe an explicit mapping from linear extensions of \mathcal{P} to linear extensions of \mathcal{P}' .

Consider a linear extension $<$ of \mathcal{P} viewed as a sequence $z = (z_1, \dots, z_n)$, where $z_1 < \dots < z_n$. The sequence z contains ℓ disjoint subsequences of lengths a_1, \dots, a_ℓ formed respectively by the elements of A_1, \dots, A_ℓ . Let $z' = (z'_1, \dots, z'_n)$ be the sequence obtained from z by replacing, for all $i \in [\ell]$, the elements of A_i in the sequence z , in the order of their appearance, by the virtual elements $x_i^1, \dots, x_i^{a_i}$.

We proceed via two claims about the mapping $z \rightarrow z'$ from which the statement follows: (1) z' is a linear extension of \mathcal{P}' , and (2) for every linear extension z' of \mathcal{P}' there are q different linear extensions of \mathcal{P} that map to z' .

For (1), let i_1, i_2 be two arbitrary indices $1 \leq i_1 < i_2 \leq n$. We need to show that $z'_{i_2} \not\prec' z'_{i_1}$. The four cases to consider are: (1a) $z'_{i_1}, z'_{i_2} \in W$, (1b) $z'_{i_1} = x_i^p$ and $z'_{i_2} = x_j^q$, (1c) $z'_{i_1} \in W$ and $z'_{i_2} = x_i^p$, and (1d) $z'_{i_1} = x_i^p$, and $z'_{i_2} \in W$. These correspond to the four cases in the definition of \prec' and the claim easily follows in each case by the construction of z' .

For (2), consider a linear extension (sequence) z' of \mathcal{P}' , and for all $i \in [\ell]$ replace the elements $\{x_i^1, \dots, x_i^{a_i}\}$ in z' by an arbitrary permutation of the elements of A_i . In this way we obtain q different linear extensions of \mathcal{P} , and when applying the above mapping to these linear extensions, they all yield the same z' . ◀

Proof of Lemma 3. Let $t = |A|$, and let us label the elements of A as z_1, \dots, z_t . Let $<_1$ and $<_2$ be the realizers of the two-dimensional poset \mathcal{P} . Then, as A is an antichain, its elements can be labeled such that $z_1 <_1 \dots <_1 z_t$, and $z_t <_2 \dots <_2 z_1$. The crucial observation is that the neighborhood of an arbitrary $y \in X \setminus A$ in A is defined by an *interval* of indices.

Formally, for $y \in X \setminus A$ that is above or below A , let z_i, z_j be the elements of $N(y) \cap A$ with smallest, resp. largest index (it may happen that $i = j$). Define $b(y) = i - 0.5$ and $b'(y) = j + 0.5$ the *boundaries* of the neighborhood of y . If y is incomparable with A , set the boundaries to dummy values $b(y) = 0, b'(y) = t + 1$.

If y is above A , then for all k such that $b(y) < k < b'(y)$, we have $z_k \prec y$. To see this, observe that $z_k <_1 z_j <_1 y$, and $z_k <_2 z_i <_2 y$.

Symmetrically, if y is below A , then for all k such that $b(y) < k < b'(y)$, we have $y \prec z_k$. To see this, observe that $y <_1 z_i <_1 z_k$, and $y <_2 z_j <_2 z_k$.

Let $b_1, \dots, b_{2(n-t)}$ be the multiset of neighborhood boundaries sorted in increasing order. Their number is $2(n-t)$ as each of the $n-t$ elements of $X \setminus A$ contribute exactly two boundaries. Let us add the two dummy boundaries $b_0 = 0$ and $b_{2(n-t)+1} = t+1$ (in case they never occurred during the process).

The classes of A defined by the partition $N(\cdot)$ are then of the form $\{z_j : b_i < j < b_{i+1}\}$ where $0 \leq i \leq 2(n-t)$. There are at most $2(n-t) + 1$ such classes (not all boundaries are necessarily distinct, and we can now remove empty classes due to duplicate boundaries). Moreover, the two classes delimited by b_0 to the left, respectively by $b_{2(n-t)+1}$ to the right are identical, corresponding to elements of A incomparable to all $y \in X \setminus A$. The claimed bound on the maximum number of classes follows. ◀

2.2 A faster algorithm

We now describe the improvements to Algorithm #LE-2D and its analysis that lead to the running time claimed in Theorem 1.

Canonical matchings. Observe that set A in Lemma 3 denotes an arbitrary antichain. When A is assumed to be the complement of a maximum matching with a certain property, a stronger statement can be shown.

Let M be a maximum matching of $\mathcal{C}(\mathcal{P})$, let W be its vertex set, and let $A = X \setminus W$. We call an edge $\{x_i, y_i\}$ of M *separated*, if there exist $x_1, x_2 \in A$ such that $x_i \prec x_1$ and $x_2 \prec y_i$. (In other words, x_i is below A , and y_i is above A .) Observe that, in this case, x_1 and x_2 must be the same, as otherwise M could be made larger by replacing edge $\{x_i, x_j\}$ by the two edges $\{x_i, x_1\}, \{x_2, x_j\}$. A matching is *canonical* if it contains no separated edges.

We argue that in an arbitrary poset a canonical matching of the same size as the maximum matching can be found in polynomial time. Indeed, start with an arbitrary maximum matching M . If M contains no separated edges, we are done. Otherwise, let $\{x_i, y_i\}$ be an edge of M with $x \in A$ such that $x_i \prec x \prec y_i$. (Such a triplet can easily be found in polynomial time.) Replace the edge $\{x_i, y_i\}$ in M by the edge $\{x, y_i\}$. As x was previously not matched, the resulting set of edges is still a maximum matching. We claim that with $O(n^2)$ such swaps we obtain a canonical matching (i.e. one without separated edges). To see this, consider as potential function the sum of ranks of all vertices in the current matching, according to an arbitrary fixed linear extension of \mathcal{P} . Each swap increases the potential by at least one (since x must come after x_i in every linear extension). Since the sum of ranks is an integer in $O(n^2)$, the number of swaps until we are done is also in $O(n^2)$. In the following, we can therefore assume that M is a canonical maximum matching. We can now state the stronger structural lemma.

► **Lemma 5.** *Let $\mathcal{P} = (X, \prec)$ be a size- n poset, with $\dim(\mathcal{P}) \leq 2$. Let M be a canonical maximum matching in $\mathcal{C}(\mathcal{P})$ with vertex set W , and let $A = X \setminus W$. Then, $N(\cdot)$ partitions A into at most $|W|$ classes.*

Proof. Since M is canonical, for every edge $\{x_i, y_i\}$, one of the following must hold:

- (i) x_i and y_i are both above A ,
- (ii) x_i and y_i are both below A ,
- (iii) x_i is incomparable with A and y_i is above A ,
- (iv) y_i is incomparable with A and x_i is below A .

Recall that in the proof of Lemma 3, we considered, for all $y \in X \setminus A$, the two boundaries of the interval $N(y) \cap A$. Now, in cases (iii) and (iv), only one of x_i and y_i need to be considered, as the neighborhood of the other is disjoint from A .

In cases (i) and (ii), it is also sufficient to consider only one of x_i and y_i as we have $N(x_i) \cap A = N(y_i) \cap A$. Furthermore, in this case $|N(x_i) \cap A| = 1$. To see this, suppose that there are $z, z' \in A$ such that $z \in N(x_i)$ and $z' \in N(y_i)$. Then M could be extended by replacing the edge $\{x_i, y_i\}$ with the edges $\{z, x_i\}$ and $\{z', y_i\}$, contradicting the maximality of M .

It follows that in the argument of Lemma 3 we only need to consider the intervals created by $|M|$ elements of $X \setminus A$, yielding the bound $2|M| = |W|$ on the number of classes. It is easy to construct examples where the bound is tight. ◀

It follows that, if we require the matching M in Step 1 of Algorithm #LE-2D to be canonical, then by Lemma 4, the bound on the number of downsets improves to $\left(3^\alpha \left(\frac{1}{2\alpha}\right)^{2\alpha}\right)^n$. In the range of interest $0 < \alpha < 1/3$ this quantity is easily upper bounded by 1.8912^n with maximum at $\alpha \approx 0.319$.

Packing triplets and quartets. The final improvement in running time comes from the attempt to find, instead of a matching (i.e. a packing of edges), a packing of larger connected structures. Beyond the concrete improvement, the technique may be of more general applicability and interest, which we illustrate in §3 for the jump number problem.

Assume, as before, that M is a canonical maximum matching of $\mathcal{C}(\mathcal{P})$ of size αn with vertex set W and that A denotes the antichain $X \setminus W$. Let us form an auxiliary bipartite graph B with vertex sets L and R , where $L = A$, and $R = M$, i.e. R consists of the *edges* of M . A vertex $x \in L$ is connected to a vertex $\{x_i, y_i\} \in R$ exactly if x is comparable to one or both of x_i and y_i . Let M_B be a maximum matching of B , of size βn . Clearly, $\beta \leq \alpha$.

Edges of M_B connect vertices in A to matched edges of M , forming *triplets* of vertices of X that induce connected subgraphs in $\mathcal{C}(\mathcal{P})$. Let T denote the set of all triplets created by edges of M_B .

Let us form now another auxiliary bipartite graph B' with vertex sets L' and R' , where L' consist of the vertices of A *unmatched* in M_B , and let $R' = T$, i.e. the triplets found in the previous round. A vertex $x \in L'$ is connected to a vertex $z \in R'$ exactly if x is comparable to at least one of the vertices forming the triplet z . Let $M_{B'}$ be a maximum matching of B' , and denote its size by γn . Clearly, $\gamma \leq \beta$.

Edges of $M_{B'}$ connect vertices in A to triplets of T , forming *quartets* of vertices of X that induce connected subgraphs in $\mathcal{C}(\mathcal{P})$. Let Q denote the set of all quartets created by edges of $M_{B'}$.

Let A' denote the vertices of A that were not matched in either of the two matching rounds. Observe that $|A'| = n(1 - 2\alpha - \beta - \gamma)$. We make the following observations.

(1) The endpoints of edges of M that were *unmatched* in M_B are not comparable to any vertex in A' (assuming that A' is nonempty), as otherwise M_B would not have been maximal. There are $n(\alpha - \beta)$ such unmatched edges. These contribute a factor of $3^{n(\alpha - \beta)}$ to the number of downsets.

(2) The vertices in triplets of T that were *unmatched* in $M_{B'}$ are not comparable to any vertex in A' (assuming that A' is nonempty), as otherwise $M_{B'}$ would not have been maximal. There are $n(\beta - \gamma)$ such triplets. A simple case-analysis shows that the number of downsets of a size-3 poset with connected comparability graph is at most 5. It follows that these triplets contribute a factor of at most $5^{n(\beta - \gamma)}$ to the number of downsets.

(3) There are γn quartets in Q . A case-analysis⁵ shows that the number of downsets of a size-4 poset with connected comparability graph is at most 9. It follows that these quartets contribute a factor of at most $9^{n\gamma}$ to the number of downsets.

(4) All vertices in $X \setminus A'$ are accounted for. As for the vertices in A' , we partition them into classes A_1, \dots, A_ℓ by $N(\cdot)$, and apply the same transformation as previously, creating a new poset \mathcal{P}' . By the previous discussion, only the vertices from the quartets in Q may be comparable to vertices in A' . Furthermore, in each quartet, only the vertices coming from the original matching M may be comparable to a vertex in A' (other vertices come from the antichain $A \supseteq A'$). Thus, by Lemma 5, the number of classes created on A' is $\ell \leq 2\gamma n$.

Putting everything together, assuming $\gamma > 0$ (the case $\gamma = 0$ is discussed later), we obtain the upper bound τ^n on the number of downsets of \mathcal{P}' , where $\tau = \tau(\alpha, \beta, \gamma) = 3^{(\alpha-\beta)} \cdot 5^{(\beta-\gamma)} \cdot 9^\gamma \cdot \left(\frac{1-2\alpha-\beta+\gamma}{2\gamma}\right)^{2\gamma}$. Under the constraint $0 < \gamma \leq \beta \leq \alpha < 1/3$, the bound $\tau < 1.8286$ holds, with the maximum attained for $\alpha = \beta = \gamma (\approx 0.1882)$. Observe that the least favorable case occurs when all edges of M are matched into triplets, and all triplets are matched into quartets.

When $\gamma = 0$, no quartets are created, and A' forms a single class, transformed in \mathcal{P}' into a single chain, contributing a linear factor to the overall bound. Thus, the upper bound $n \cdot \tau^n$ holds, with $\tau = \tau(\alpha, \beta) = 3^{(\alpha-\beta)} \cdot 5^\beta < 1.71$, maximum attained for $\alpha = \beta (\approx 1/3)$.

The resulting algorithm #LE-2D* is listed as Algorithm 2. The correctness and running time bounds (Theorem 1) follow from the previous discussion. We defer some remarks about the algorithm and its analysis to §4.

Open questions. The following questions about counting linear extensions are suggested in increasing order of difficulty. (1) Can #LE be solved in two-dimensional posets faster than the algorithm of Theorem 1? (2) Can #LE be solved in time $O(c^n)$ for $c < 2$ in d -dimensional posets, for $d \geq 3$? (3) Can #LE be solved in time $O(c^n)$ for $c < 2$ in arbitrary posets?

3 The jump number problem

In this section we present our improvement for the jump number problem. We start with a formal definition of the problem, and the straightforward dynamic programming. We then review the algorithm of Kratsch and Kratsch, followed by our extension. The result is intended as an illustration of the matching technique of §2, which is not specific to two-dimensional posets.

Given a linear extension $x_1 < \dots < x_n$ of a poset $\mathcal{P} = (X, \prec)$, a pair of neighbors (x_i, x_{i+1}) is a *jump* if $x_i \not\prec x_{i+1}$, and is a *bump* if $x_i \prec x_{i+1}$. The number of jumps, resp. bumps of the linear extension $<$ of \mathcal{P} is denoted as $\text{jump}(<)$, resp. $\text{bump}(<)$. The jump number problem asks to compute the minimum possible value $\text{jump}(<)$ for a linear extension $<$ of \mathcal{P} . Additionally, a linear extension realizing this value should be constructed. In the algorithms we describe, obtaining a linear extension that realizes the minimum jump number is a mere technicality, we thus focus only on computing the minimum jump number.

An easy observation is that the relation $\text{jump}(<) + \text{bump}(<) = n - 1$ holds for all linear extensions $<$ of \mathcal{P} . Minimizing the number of jumps is thus equivalent to maximizing the number of bumps, allowing us to focus on the latter problem.

⁵ An easy induction shows more generally that the maximum number of downsets of a size- n poset with connected comparability graph is $2^{n-1} + 1$.

30:10 Exact Exponential Algorithms for Two Poset Problems

Let $\text{bump}(\mathcal{P})$ denote the maximum bump number of a linear extension of \mathcal{P} . For all $Y \subseteq X$, and $x \in \max(Y)$, let $\text{bump}(Y, x)$ denote the maximum bump number of a linear extension of the subposet of \mathcal{P} induced by Y that ends with element x . Let us define $\text{bump}(\{x\}, x) = 0$, for all $x \in X$. We recursively express $\text{bump}(Y, x)$ by removing x from the end and trying all remaining elements in turn as the new last element:

$$\text{bump}(Y, x) = \max_{y \in \max(Y \setminus \{x\})} \left(\text{bump}(Y \setminus \{x\}, y) + [y \prec x] \right). \quad (2)$$

The term $[y \prec x]$ denotes the value 1 if $y \prec x$, i.e. if the last pair forms a bump, and 0 otherwise. Executing recurrence (2) naively leads to an algorithm that computes $\text{bump}(\mathcal{P})$ in time $O(2^n \cdot n^2)$.

We now describe the improvement of Kratsch and Kratsch [27]. Observe that jumps partition a linear extension of \mathcal{P} uniquely into a sequence of chains of \mathcal{P} , such that the last element of each chain is incomparable with the first element of the next chain, and all other neighboring pairs are comparable.

Consider a linear extension with minimum jump number and let C_1, \dots, C_k denote the non-trivial chains of its decomposition (i.e. all chains of length at least 2). Let C denote the set of vertices of chains C_1, \dots, C_k . Then, as all bumps occur between elements of C , the bump number of \mathcal{P} equals the bump number of the subposet induced by C . In other words, to compute the maximum bump number, it is sufficient to consider in recurrence (2) the subsets of the ground set X that are candidate sets C in the optimum.

Kratsch and Kratsch consider a maximum matching M of $\mathcal{C}(\mathcal{P})$ with vertex set W and observe that the vertices of the antichain $A = X \setminus W$ that participate in nontrivial chains (i.e. that are in C) form a matching with vertices of W . (This is because a vertex $v \in A$ can only form a bump together with a vertex from $X \setminus A$, and two vertices $v, v' \in A$ cannot form bumps with the same vertex, as that would contradict their incomparability.) Moreover, $v, v' \in A$ cannot be the neighbors of the two endpoints of a matched edge of M , as that would contradict the maximality of M .

Thus, it suffices to compute (2) over subsets of X that consist of $W \cup A'$, where $A' \subseteq A$ and $|A'| \leq |M|$. Furthermore, only *downsets* of \mathcal{P} need to be considered, leading to a further saving due to the fact that W forms a matching. Denoting $|M| = \alpha n$, the overall number of subsets of X that need to be considered is $\binom{(1-2\alpha)n}{\leq \alpha n} \cdot 3^{\alpha n}$.

Packing triplets. We now describe our improvement. Again, let M be a canonical maximum matching of $\mathcal{C}(\mathcal{P})$ of size αn with vertex set W and let A denote the antichain $X \setminus W$. Form an auxiliary bipartite graph B with vertex sets L and R , where $L = A$, and $R = M$. A vertex $x \in L$ is connected to a vertex $\{x_i, y_i\} \in R$ (i.e. an edge of M) exactly if $x \prec y_i$ or $x_i \prec x$. Let M_B be a maximum matching of B , and denote its size by βn . Clearly, $\beta \leq \alpha$.

Edges of M_B connect vertices in A to matched edges of M , forming *triplets* of vertices of X that induce connected subgraphs in $\mathcal{C}(\mathcal{P})$. Let T denote the set of all such triplets. (To keep the argument simple we forgo in this case further rounds of matching and the forming of quartets. The result is thus not optimized to the fullest extent.)

Let A' denote the vertices of A that were not matched in M_B . Observe that $|A'| = n(1 - 2\alpha - \beta)$. We make the following observations.

(1) The endpoints of edges of M that were *unmatched* in M_B are not comparable to any vertex in A' (assuming that A' is nonempty), as otherwise M_B would not have been maximal. There are $n(\alpha - \beta)$ such edges. These edges contribute a factor of $3^{n(\alpha - \beta)}$ to the number of downsets.

(2) There are βn triplets in T . These contribute a factor of at most $5^{n\beta}$ to the number of downsets.

(3) All vertices in $X \setminus A'$ are accounted for. Vertices of A' that participate in non-trivial chains of the optimal linear extension can be matched to vertices in the triplets of T . A vertex $v \in A'$ can only be connected to those vertices of a triplet in T that are endpoints of an edge in M (all other vertices come from A , are thus incomparable with v). Furthermore, $v, v' \in A'$ may not connect to different endpoints of the same edge in M , as that would contradict the maximality of M . It follows that each vertex in A' that participates in C must be matched to a unique triplet in T . Thus, at most βn vertices of A' need to be considered.

The resulting Algorithm JN is listed as Algorithm 3. Its correctness follows from the previous discussion.

Running time. In the large matching ($\alpha \geq 1/3$) case, the bound given in §2 on the number of downsets holds, and the running time is within the bound of Theorem 2. We assume therefore that $\alpha < 1/3$.

In the special case $\beta = 0$ only vertices in M need to be considered, with an overall upper bound $3^{\alpha n}$ on the number of downsets. For $\alpha < 1/3$, this quantity is below 1.443^n .

Assuming $\beta > 0$, we have an upper bound τ^n on the number of downsets of \mathcal{P} , where $\tau^n = 3^{(\alpha-\beta)n} \cdot 5^{\beta n} \cdot \binom{n(1-2\alpha-\beta)}{\leq \beta n}$. To obtain a simpler expression, we use a standard upper bound [9, p. 406] on the sum of binomial coefficients. Assuming $0 \leq 2b \leq a \leq 1$, we have $\binom{na}{\leq nb} = \sum_{k=0}^{nb} \binom{na}{k} \leq n^{O(1)} \cdot \left(\frac{a^a}{b^b \cdot (a-b)^{(a-b)}} \right)^n$.

Plugging in $a = 1 - 2\alpha - \beta$ and $b = \beta$, and assuming $2b \leq a$, we have $2\alpha + 3\beta \leq 1$. Omitting the polynomial factor, we obtain $\tau \leq 3^{(\alpha-\beta)} \cdot 5^\beta \cdot \frac{(1-2\alpha-\beta)^{(1-2\alpha-\beta)}}{\beta^\beta \cdot (1-2\alpha-2\beta)^{(1-2\alpha-2\beta)}}$. In the critical region $0 < \beta \leq \alpha < 1/3$ we obtain the bound $\tau < 1.824$, with the maximum attained for $\alpha = \beta (\approx 0.1918)$.

When $2\alpha + 3\beta \geq 1$, we use the easier upper bound on the sum of binomial coefficients $\binom{na}{\leq nb} \leq 2^{na}$, obtaining $\tau \leq 3^{(\alpha-\beta)} \cdot 5^\beta \cdot 2^{(1-2\alpha-\beta)n}$. In the allowed range $0 < \beta \leq \alpha < 1/3$ and additionally requiring $2\alpha + 3\beta \geq 1$, the quantity is maximized for $\alpha = \beta = 0.2$, yielding $\tau < 1.8206$, within the required bounds.

4 Discussion

We start with some remarks about algorithm #LE-2D*. It is straightforward to extend this algorithm beyond pairs, triplets, and quartets, to also form k -tuples for $k > 4$ via further matching rounds. A similar analysis, however, indicates no further improvements in the upper bound. When forming triplets and quartets, other strategies are also possible. For instance, we may try to combine connected pairs of edges from M into quartets. The quartets formed in this way are, in fact, preferable to those obtained by augmenting triplets, as their number of downsets is strictly less than the value 9 given before. (The value 9 is attained when 3 of the 4 vertices form an antichain, which is not possible if the quartet consists of two matched edges.)

We observe that in some instances, the largest antichain may be significantly larger than the antichain A obtained as the complement of the maximum matching. One can find the largest antichain in time $O(n^{5/2})$ via a reduction to bipartite matching (see e.g. [43]). In these cases, using the partition of the antichain A_i, \dots, A_ℓ (without arguing about matchings) may lead to a better running time. In two-dimensional posets with a realization \langle_1, \langle_2 , the largest antichain can be found in time $O(n \log n)$ by reduction to the *largest decreasing subsequence* problem. In our analysis, we assumed the classes A_i to be of equal size. The running time can, of course, be significantly lower when the distribution of class sizes is far from uniform.

30:12 Exact Exponential Algorithms for Two Poset Problems

We further remark that the actual time and space requirement of our algorithms is dominated by the number of downsets of a given poset \mathcal{P} (or rather, of the transformed poset \mathcal{P}'). The number of downsets (order ideals) is known to equal the number of antichains [40, §3]. Counting antichains is, in general, $\#P$ -hard [35], but solvable in two-dimensional posets in polynomial time [41, 33]. Thus, assuming that the transformed poset \mathcal{P}' is also two-dimensional, we can efficiently compute a precise, *instance-specific* estimate of the time and space requirements of our algorithms.

To see that indeed, $\dim(\mathcal{P}') \leq 2$, recall that \mathcal{P}' is obtained from \mathcal{P} by replacing antichains A_i by chains of equal size, such that the comparability of the involved elements to elements in $X \setminus A$ is preserved. We can obtain a two-dimensional embedding of \mathcal{P}' by starting with a two-dimensional embedding of \mathcal{P} , with points having integer coordinates, and no two points aligned on either coordinate. For an arbitrary $x \in A_i$, form a 0.5×0.5 box around the point x , and place the chain replacing A_i on the main diagonal of this box. Then the comparability of points in the chain with elements in $X \setminus A_i$ is the same as for the point x .

Optimization. Our first two bounds in §2 depend only on the fraction α of matched vertices, and their maxima are found using standard calculus. The final bounds given in Theorem 1 and Theorem 2 however, require us to optimize over unwieldy multivariate quantities with constrained variables. The given numerical bounds were obtained using Wolfram Mathematica software. We have, however, independently certified the bounds, by the method illustrated next.

Suppose we want to show that $\tau < 1.8286$, where $\tau = \tau(\alpha, \beta, \gamma) = 3^{(\alpha-\beta)} \cdot 5^{(\beta-\gamma)} \cdot 9^\gamma \cdot \left(\frac{1-2\alpha-\beta+\gamma}{2\gamma}\right)^{2\gamma}$, and $A \leq \gamma \leq \beta \leq \alpha \leq B$, for $A, B \in (0, 1/3)$.

Consider a box $\mathcal{B} = [\alpha_1, \alpha_2] \times [\beta_1, \beta_2] \times [\gamma_1, \gamma_2] \subseteq [A, B]^3$. Then, at an *arbitrary* point $(\alpha, \beta, \gamma) \in \mathcal{B}$, the following (rather weak) upper bound holds.

$$\tau(\alpha, \beta, \gamma) \leq 3^{\alpha_2} \cdot \left(\frac{5}{3}\right)^{\beta_2} \cdot \left(\frac{9}{5}\right)^{\gamma_2} \cdot \left(\frac{1-2\alpha_1-\beta_1+\gamma_2}{2\gamma_1}\right)^{2\gamma_2}.$$

To show $\tau < 1.8286$, it is sufficient to exhibit a collection of boxes, such that (1) for all boxes, the stated upper bound evaluates to a value smaller than 1.8286, and (2) the union of the boxes covers the entire domain of the variables. We can find such a collection of boxes if we start with a single box that contains the entire domain of the variables, and recursively split boxes into two equal parts (along the longest side) whenever the upper bound evaluates to a value larger than the required value.

Higher dimensions. A straightforward extension of Algorithms $\#LE-2D$ and $\#LE-2D^*$ to higher dimensional posets does not yield improvements over the naïve dynamic programming. The crux of the argument in two dimensions is that a large antichain in \mathcal{P} is split, according to the neighborhoods in $\mathcal{C}(\mathcal{P})$ into a small number of classes. In dimensions three and above it is easy to construct posets with an antichain containing *almost all* elements, e.g. such that $|X \setminus A| = O(\sqrt{n})$, with the property that all elements of A have unique neighborhoods. In this case, the number of classes is $|A|$ and the described techniques yield no significant savings.

Algorithm 2 Algorithm #LE-2D*.

Input: Poset $\mathcal{P} = (X, \prec)$, where $|X| = n$.

Output: The number of linear extensions $\text{LE}(\mathcal{P})$ of \mathcal{P} .

- 1: Find a maximum matching M of $\mathcal{C}(\mathcal{P})$ with vertex set W .
 - 2: Let $A = X \setminus W$.
 - 3: Find T and Q as described, and let A' be the unmatched part of A .
 - 4: Let A_1, \dots, A_ℓ be the partition of A' by the neighborhoods in $\mathcal{C}(\mathcal{P})$.
 - 5: Let $a_i = |A_i|$ for $i \in [\ell]$.
 - 6: Construct $\mathcal{P}' = (X', \prec')$.
 - 7: Compute $N = \text{LE}(\mathcal{P}')$ using (1).
 - 8: **return** $\prod_{i \in [\ell]} (a_i!) \cdot N$.
-

Algorithm 3 Algorithm JN.

Input: Poset $\mathcal{P} = (X, \prec)$, where $|X| = n$.

Output: The minimum jump number of a linear extension of \mathcal{P} .

- 1: Find a maximum matching M of $\mathcal{C}(\mathcal{P})$ with vertex set W .
 - 2: Let $A = X \setminus W$.
 - 3: Find T as described, and let A' be the unmatched part of A .
 - 4: Let $\beta = |T|$.
 - 5: Compute $B = \text{bump}(\mathcal{P})$ by (2), using downsets of \mathcal{P} with at most βn vertices from A' .
 - 6: **return** $n - 1 - B$.
-

References

- 1 P. D. Amer, C. Chassot, T. J. Connolly, M. Diaz, and P. Conrad. Partial-order transport service for multimedia and other applications. *IEEE/ACM Transactions on Networking*, 2(5):440–456, October 1994. doi:10.1109/90.336326.
- 2 KA Baker, Peter C Fishburn, and Fred S Roberts. Partial orders of dimension 2, interval orders, and interval graphs, 1970.
- 3 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach.*, 9:61–63, 1962.
- 4 G. R. Brightwell, S. Felsner, and W. T. Trotter. Balancing pairs and the cross product conjecture. *Order*, 12(4):327–349, December 1995. doi:10.1007/BF01110378.
- 5 Graham Brightwell. Balanced pairs in partial orders. *Discrete Mathematics*, 201(1):25–52, 1999. doi:10.1016/S0012-365X(98)00311-2.
- 6 Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3):225–242, September 1991. doi:10.1007/BF00383444.
- 7 Russ Bubley and Martin Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201(1):81–88, 1999. doi:10.1016/S0012-365X(98)00333-1.
- 8 Stéphan Ceroi. A weighted version of the jump number problem on two-dimensional orders is np-complete. *Order*, 20(1):1–11, 2003.
- 9 T.M. Cover and J.A. Thomas. *Elements of Information Theory*. A Wiley-Interscience publication. Wiley, 2006.
- 10 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as cnf-sat. *ACM Trans. Algorithms*, 12(3):41:1–41:24, May 2016. doi:10.1145/2925416.
- 11 Karel De Loof, Hans De Meyer, and Bernard De Baets. Exploiting the lattice of ideals representation of a poset. *Fundam. Inf.*, 71(2,3):309–321, February 2006.

- 12 Samuel Dittmer and Igor Pak. Counting linear extensions of restricted posets, 2018. [arXiv:1802.06312](https://arxiv.org/abs/1802.06312).
- 13 Ben Dushnik and E. W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610, 1941.
- 14 Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, January 1991. doi:10.1145/102782.102783.
- 15 E. Eiben, R. Ganian, K. Kangas, and S. Ordyniak. Counting linear extensions: Parameterizations by treewidth. *Algorithmica*, 81(4):1657–1683, April 2019. doi:10.1007/s00453-018-0496-4.
- 16 Stefan Felsner and Thibault Manneville. Linear extensions of n-free orders. *Order*, 32(2):147–155, 2015. doi:10.1007/s11083-014-9321-0.
- 17 Stefan Felsner and Lorenz Wernisch. Markov chains for linear extensions, the two-dimensional case. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA.*, pages 239–247, 1997. URL: <http://dl.acm.org/citation.cfm?id=314161.314262>.
- 18 Peter C. Fishburn and William V. Gehrlein. A comparative analysis of methods for constructing weak orders from partial orders. *The Journal of Mathematical Sociology*, 4(1):93–102, 1975. doi:10.1080/0022250X.1975.9989846.
- 19 R. L. Graham, M. Grötschel, and L. Lovász, editors. *Handbook of Combinatorics (Vol. 1)*. MIT Press, Cambridge, MA, USA, 1995.
- 20 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10:196–210, 1962.
- 21 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, December 2001. doi:10.1006/jcss.2001.1774.
- 22 Gwenaël Joret, Piotr Micek, and Veit Wiechert. Sparsity and dimension. *Combinatorica*, 38(5):1129–1148, October 2018. doi:10.1007/s00493-017-3638-4.
- 23 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, June 1984. doi:10.1007/BF00565647.
- 24 Kustaa Kangas, Teemu Hankala, Teppo Mikael Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets. In *IJCAI 2016*, pages 603–609, 2016. URL: <http://www.ijcai.org/Abstract/16/092>.
- 25 Kustaa Kangas, Mikko Koivisto, and Sami Salonen. A faster tree-decomposition based algorithm for counting linear extensions. In *IPEC 2018*, pages 5:1–5:13, 2018. doi:10.4230/LIPIcs.IPEC.2018.5.
- 26 Donald E. Knuth and Jayme Luiz Szwarcfiter. A structured program to generate all topological sorting arrangements. *Inf. Process. Lett.*, 2(6):153–157, 1974. doi:10.1016/0020-0190(74)90001-5.
- 27 Dieter Kratsch and Stefan Kratsch. The jump number problem: Exact and parameterized. In *IPEC 2013*, pages 230–242, 2013. doi:10.1007/978-3-319-03898-8_20.
- 28 Nathan Linial. Hard enumeration problems in geometry and combinatorics. *SIAM J. Algebraic Discrete Methods*, 7(2):331–335, April 1986. doi:10.1137/0607036.
- 29 L. Lovász. *An Algorithmic Theory of Numbers, Graphs, and Convexity*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1986.
- 30 Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Probabilistic preference logic networks. In *ECAI 2014*, pages 561–566, 2014. doi:10.3233/978-1-61499-419-0-561.
- 31 Heikki Mannila and Christopher Meek. Global partial orders from sequential data. In *ACM SIGKDD 2000*, pages 161–168, 2000. doi:10.1145/347090.347122.
- 32 Jason Morton, Lior Pachter, Anne Shiu, Bernd Sturmfels, and Oliver Wienand. Convex rank tests and semigraphoids. *SIAM J. Discrete Math.*, 23(3):1117–1134, 2009. doi:10.1137/080715822.

- 33 Rolf Möhring. *Computationally Tractable Classes of Ordered Sets*, pages 105–193. Springer, January 1989. doi:10.1007/978-94-009-2639-4_4.
- 34 Teppo Mikael Niinimäki and Mikko Koivisto. Annealed importance sampling for structure learning in bayesian networks. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1579–1585, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6885>.
- 35 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983. doi:10.1137/0212053.
- 36 William R Pulleyblank. *On minimizing setups in precedence constrained scheduling*. Universität Bonn. Institut für Ökonometrie und Operations Research, 1981.
- 37 I. Rival. *Ordered Sets: Proceedings of the NATO Advanced Study Institute held at Banff, Canada, August 28 to September 12, 1981*. Nato Science Series C: Springer Netherlands, 2012.
- 38 Alex Scott and David R. Wood. Better bounds for poset dimension and boxicity. *CoRR*, abs/1804.03271, 2018. arXiv:1804.03271.
- 39 Richard P Stanley. Two poset polytopes. *Discrete & Computational Geometry*, 1(1):9–23, 1986.
- 40 R.P. Stanley. *Enumerative Combinatorics:.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1997.
- 41 George Steiner. On estimating the number of order ideals in partial orders, with some applications. *Journal of statistical planning and inference*, 34(2):281–290, 1993.
- 42 George Steiner and Lorna K Stewart. A linear time algorithm to find the jump number of 2-dimensional bipartite partial orders. *Order*, 3(4):359–367, 1987.
- 43 R.E. Tarjan. *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1983.
- 44 W.T. Trotter. *Combinatorics and partially ordered sets: dimension theory*. Johns Hopkins Series in the Mathematical Sciences. J. Hopkins University Press, 1992.
- 45 Chris S. Wallace, Kevin B. Korb, and Honghua Dai. Causal discovery via mml. In *ICML 1996*, pages 516–524, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
- 46 P. Winkler. Average height in a partially ordered set. *Discrete Mathematics*, 39(3):337–341, 1982. doi:10.1016/0012-365X(82)90157-1.

Space-Efficient Data Structures for Lattices

J. Ian Munro 

Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
imunro@uwaterloo.ca

Bryce Sandlund

Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
bcsandlund@uwaterloo.ca

Corwin Sinnamon 

Department of Computer Science, Princeton University, Princeton, NJ 08540, USA
sinncore@gmail.com

Abstract

A lattice is a partially-ordered set in which every pair of elements has a unique meet (greatest lower bound) and join (least upper bound). We present new data structures for lattices that are simple, efficient, and nearly optimal in terms of space complexity.

Our first data structure can answer partial order queries in constant time and find the meet or join of two elements in $O(n^{3/4})$ time, where n is the number of elements in the lattice. It occupies $O(n^{3/2} \log n)$ bits of space, which is only a $\Theta(\log n)$ factor from the $\Theta(n^{3/2})$ -bit lower bound for storing lattices. The preprocessing time is $O(n^2)$. This structure admits a simple space-time tradeoff so that, for any $c \in [\frac{1}{2}, 1]$, the data structure supports meet and join queries in $O(n^{1-c/2})$ time, occupies $O(n^{1+c} \log n)$ bits of space, and can be constructed in $O(n^2 + n^{1+3c/2})$ time.

Our second data structure uses $O(n^{3/2} \log n)$ bits of space and supports meet and join in $O(d \frac{\log n}{\log d})$ time, where d is the maximum degree of any element in the transitive reduction graph of the lattice. This structure is much faster for lattices with low-degree elements.

This paper also identifies an error in a long-standing solution to the problem of representing lattices. We discuss the issue with this previous work.

2012 ACM Subject Classification Theory of computation; Mathematics of computing → Graph algorithms

Keywords and phrases Lattice, Partially-ordered set, Space-efficient data structure, Succinct data structure

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.31

Funding This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs Program.

1 Introduction

A lattice is a partially-ordered set with the property that for any pair of elements x and y , the set of all elements greater than or equal to both x and y must contain a unique minimal element less than all others in the set. This element is called the *join* (or *least upper bound*) of x and y . A similar condition holds for the set of all elements less than both x and y : It must contain a maximum element called the *meet* (or *greatest lower bound*) of x and y .

We consider lattices from the perspective of succinct data structures. This area of study is concerned with representing a combinatorial object in essentially the minimum number of bits while supporting the “natural” operations in constant time. The minimum number of bits required is the logarithm (base 2) of the number of such objects of size n , e.g. about $2n$ bits for a binary tree on n nodes. Succinct data structures have been very successful in dealing with trees, planar graphs, and arbitrary graphs. Our goal in this paper is to broaden the horizon for succinct and space-efficient data structures and to move to more



© J. Ian Munro, Bryce Sandlund, and Corwin Sinnamon;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 31; pp. 31:1–31:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algebraic structures. There has indeed been progress in this direction with abelian groups [7] and distributive lattices [17]. We take another step here in studying space-efficient data structures for arbitrary finite lattices.

Lattices have a long and rich history spanning many disciplines. Existing at the intersection of order theory and abstract algebra, lattices arise naturally in virtually every area of mathematics [9]. The area of formal concept analysis is based on the notion of a *concept lattice*. These lattices have been studied since the 1980s [24] and have applications in linguistics, data mining, and knowledge management, among many others [8]. Lattices have also found numerous applications in the social sciences [16].

Within computer science, lattices are also important, particularly for programming languages. Lattice theory is the basis for many techniques in static analysis of programs, and thus has applications to compiler design. Dataflow analysis and abstract interpretation, two major areas of static analysis, rely on fixed-point computations on lattices to draw conclusions about the behaviour of a program [18].

Lattice operations appear in the problem of hierarchical encoding, which is relevant to implementing type inclusion for programming languages with multiple inheritance (among other applications) [1, 4, 5, 15]. Here the problem is to represent a partially-ordered set by assigning a short binary string to each element so that lattice-like operations can be implemented using bitwise operations on these strings. The goal is to minimize the length of the strings for the sake of time and space efficiency.

In short, lattices are pervasive and worthy of study. From a data structures perspective, the natural question follows: How do we represent a lattice so that not too much space is required and basic operations like partial order testing, meet, and join can be performed quickly?

It was proven by Klotz and Lucht [14] that the number of different lattices on n elements is at least $2^{\Omega(n^{3/2})}$, and an upper bound of $2^{O(n^{3/2})}$ was shown by Kleitman and Winston [13]. Thus, any representation for lattices must use $\Omega(n^{3/2})$ bits in the worst case, and this lower bound is tight within a constant factor. We should then expect a data structure for lattices to use comparably little space.

Two naive solutions suggest themselves immediately. First, we could simply build a table containing the meet and join of every pair of elements in the given lattice. Any simple lattice operation could be performed in constant time. However, the space usage would be quadratic – a good deal larger than the lower bound. Alternatively, we could store only the transitive reduction graph of the lattice. This method turns out to be quite space-efficient: Since the transitive reduction graph of a lattice can only have $O(n^{3/2})$ edges [14, 25], the graph can be stored in $O(n^{3/2} \log n)$ bits of space; thus, the space complexity lies within a $\Theta(\log n)$ factor of the lower bound. However, the lattice operations become extremely slow as they require exhaustively searching through the graph. Indeed, it is not easy to come up with a data structure for lattices that uses less than quadratic space while answering meet, join, and partial order queries in less than linear time in the worst case.

The construction of a lattice data structure with good worst-case behaviour also has attractive connections to the more general problem of reachability in directed acyclic graphs (DAGs). Through its transitive reduction graph, a lattice can be viewed as a special type of DAG. Among other things, this paper shows that we can support reachability queries in constant time for this class of graphs while using subquadratic space. Most classes of DAGs for which this has been achieved, such as planar DAGs [23], permit a strong bound on the order dimension of the DAGs within that class. This is a property not shared by lattices, which may have order dimension linear in the size of the lattice. A long-standing difficult

problem in this line of research is to show a similar nontrivial result for the case of arbitrary sparse DAGs [19].

There has been significant progress in representation of *distributive* lattices, an especially common and important class of lattices. Space-efficient data structures for distributive lattices have been established since the 1990s [10, 11] and have been studied most recently by Munro and Sinnamon [17]. Munro and Sinnamon show that it is possible to represent a distributive lattice on n elements using $O(n \log n)$ bits of space while supporting meet and join operations (and thus partial order testing) in $O(\log n)$ time. This comes within a $\Theta(\log n)$ factor of the space lower bound by enumeration: As the number of distributive lattices on n elements is $2^{\Theta(n)}$ [6], at least $\Theta(n)$ bits of space are required for any representation.

The problem of developing a space-efficient data structure for arbitrary lattices was first studied by Talamo and Vocca in 1994, 1997, and 1999 [20, 21, 22]. They claimed to have an $O(n^{3/2} \log n)$ -bit data structure that supports partial order queries in constant time and meet and join operations in $O(\sqrt{n})$ time. However, there is a nontrivial error in the details of their structure. Although much of the data structure is correct, we believe that this mistake is a critical flaw that is not easily repaired.

To our knowledge, no other data structures have been proposed that can perform lattice operations efficiently while using less than quadratic space. Our primary motivation is to fill this gap.

2 Contributions

Drawing on ideas from [22], we present new data structures for lattices that are simple, efficient for the natural lattice operations, and nearly optimal in space complexity. Our data structures support three queries:

- **Test Order:** Given two elements x and y , determine whether $x \leq y$ in the lattice order.
- **Find Meet:** Find the meet of two elements.
- **Find Join:** Find the join of two elements.

Our first data structure (Theorem 9) is based on a two-level decomposition of a lattice into many smaller lattices. It tests the order between any two elements in $O(1)$ time and answers meet and join queries in $O(n^{3/4})$ time in the worst case. It uses $O(n^{3/2})$ words of space¹, which is a $\Theta(\log n)$ factor from the known lower bound of $\Omega(n^{3/2})$ bits. The preprocessing time is $O(n^2)$.

We generalize this structure (Corollary 10) to allow for a tradeoff between the time and space requirements. For any $c \in [\frac{1}{2}, 1]$, we give a data structure that supports meet and join operations in $O(n^{1-c/2})$ time, occupies $O(n^{1+c})$ space, and can be constructed in $O(n^2 + n^{1+3c/2})$ time. At $c = 1/2$, it coincides with the first data structure.

Taking a different approach to computing meets and joins, we present another data structure (Theorem 12) based on a recursive decomposition of the lattice. Here the operational complexity is parameterized by the maximum degree d of any element in the lattice, where the degree is defined in reference to the transitive reduction graph of the lattice. This structure answers meet and join queries in $O(d \frac{\log n}{\log d})$ time, which improves significantly on the first data structure when applied to lattices with low degree elements (as is the case for distributive lattices, for example). It uses $O(n^{3/2})$ space.

¹ We assume a word RAM model with $\Theta(\log n)$ -bit words. Henceforth, unless bits are specified, “ $f(n)$ space” means $f(n)$ words of size $\Theta(\log n)$.

This paper is organized as follows. In Section 3, we give the necessary definitions and notation used throughout the paper. In Section 4, we give the main tool we use to decompose a lattice, which we call a block decomposition. Section 5 describes the order-testing data structure and Section 6 extends this data structure to compute meets and joins. Some details of the preprocessing are left to Appendix A. Section 7 contains our recursive degree-bounded data structure. In Appendix B, we discuss the error in the papers [21, 22] and give some evidence of why it may be irreparable.

3 Preliminaries

Given a partially-ordered set (poset) (P, \leq) , we define the *downset* of an element $x \in P$ by $\downarrow x = \{z \in P \mid z \leq x\}$ and the *upset* of x by $\uparrow x = \{z \in P \mid z \geq x\}$.

► **Definition 1.** A lattice is a partially-ordered set (L, \leq) in which every pair of elements has a meet and a join.

The meet of x and y , denoted $x \wedge y$, is the unique maximal element of $\downarrow x \cap \downarrow y$ with respect to \leq . Similarly, the join of x and y , denoted $x \vee y$, is the unique minimal element of $\uparrow x \cap \uparrow y$.

Meet (\wedge) and join (\vee) are also called greatest lower bound (GLB) and least upper bound (LUB), respectively. Lattices have the following elementary properties. Let $x, y, z \in L$.

■ The meet and join operations are idempotent, associative, and commutative:

$$\begin{array}{lll} x \vee x = x & x \vee (y \vee z) = (x \vee y) \vee z & x \vee y = y \vee x \\ x \wedge x = x & x \wedge (y \wedge z) = (x \wedge y) \wedge z & x \wedge y = y \wedge x \end{array}$$

- If $x \leq y$, then $x \wedge y = x$ and $x \vee y = y$.
- If $z \leq x$ and $z \leq y$, then $z \leq x \wedge y$. If $z \geq x$ and $z \geq y$, then $z \geq x \vee y$.
- A lattice must have a unique *top* element above all others and unique *bottom* element below all others in the lattice order.

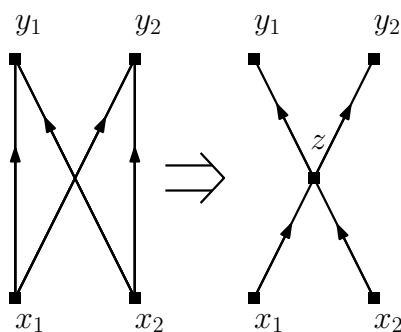
Moreover, meet and join are dual operations. If the lattice is flipped upside-down, then meet become join and vice versa.

In this paper, we prefer to work with *partial lattices*. A partial lattice is the same as a lattice except that it does not necessarily have top or bottom elements. Thus, the meet or join of two elements may not exist in a partial lattice; we use the symbol `null` to indicate this. We write $x \wedge y = \text{null}$ if $\downarrow x \cap \downarrow y = \emptyset$ and $x \vee y = \text{null}$ if $\uparrow x \cap \uparrow y = \emptyset$. Note that in a partial lattice the meet or join of x and y may not exist, but when they do exist they must be unique.

Equivalently, a partial lattice is a partially-ordered set satisfying the *lattice property*: If there are four elements x_1, x_2, y_1 , and y_2 such that $x_1, x_2 < y_1, y_2$, then there must exist an intermediate element z with $x_1, x_2 \leq z \leq y_1, y_2$. See Figure 1. This statement trivially follows from the definition of a lattice; it only says that there cannot be multiple maximal elements in $\downarrow y_1 \cap \downarrow y_2$ or multiple minimal elements in $\uparrow x_1 \cap \uparrow x_2$.

Henceforth, we use the term “lattice” to mean “partial lattice”. The difference is trivial in a practical sense, and our results are easier to express when we only consider partial lattices.

We assume that any lattice we wish to represent is given initially its *transitive reduction graph* (TRG). This is a directed acyclic graph (DAG) having a node for each lattice element and an edge (u, v) whenever $u < v$ and there is no intermediate node w such that $u < w < v$. The edge relation of this graph is called the *covering* relation: Whenever (u, v) is an edge of the TRG we say that v *covers* u .



■ **Figure 1** The configuration on the left cannot exist in a lattice for any nodes $x_1, x_2, y_1,$ and y_2 . There must be a node z between them as shown. We refer to this as the *lattice property*.

4 Block Decompositions

The main tool used in our data structure is called a block decomposition of a lattice. It is closely based on techniques used by Talamo and Vocca in [21, 22].

Let L be a lattice with n elements. A block decomposition of L is a partition of the elements of L into subsets called *blocks*. The blocks are chosen algorithmically using the following method. We first specify a positive integer k to be the *block size* of the decomposition (our application will use the block size \sqrt{n}). Then we label the elements of L as “fat” or “thin” according to the sizes of their downsets. A fat node is “minimal” if all elements in its downset, except itself, are thin. Formally:

▶ **Definition 2.** A node $x \in L$ is called fat if $|\downarrow x| \geq k$, and x is called thin if $|\downarrow x| < k$. We say x is a minimal fat node if x is fat and every other node in $\downarrow x$ is thin.

Minimal fat nodes are the basis for choosing blocks, which is done as follows. While there exists a minimal fat node h in the lattice, create a new *principal* block B containing the elements of $\downarrow h$, and then delete those nodes from the lattice. The node h is called the *block header* of B .

Deleting the elements of B may cause some fat nodes to become thin by removing elements from their downsets; this should be accounted for before choosing the next block. When there are no fat nodes in the lattice, put the remaining elements into a single block B_{res} called the *residual block*.

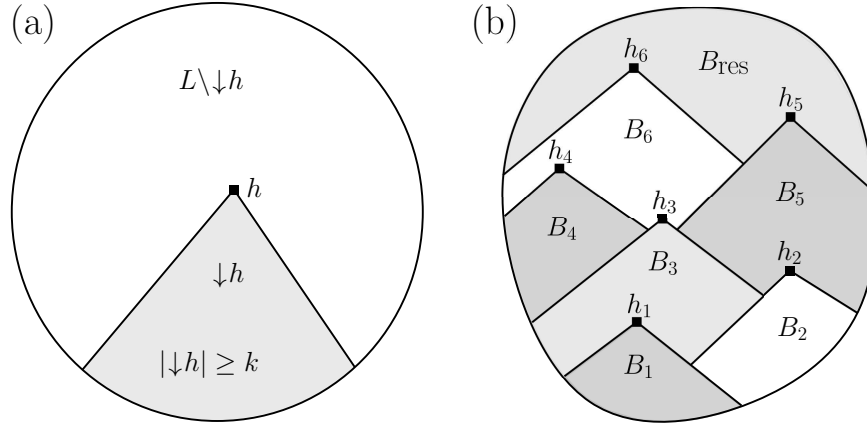
This method creates a set of principal blocks $\{B_1, B_2, \dots, B_m\}$ and a residual block B_{res} . Each principal block B_i has a block header h_i , which was the minimal fat node used to create B_i . A block header is always the top element within its block. The residual block may or may not have a top element, but it is not considered to have a block header regardless. Figure 2 shows a full block decomposition.

The block decomposition algorithm is summarized in Algorithm 1; it will be shown later that this algorithm can be implemented to run in $O(n^{7/4})$ time, where n is the number of elements in the lattice.

4.1 Properties of Block Decompositions

Let us note some elementary properties of block decompositions. Let L be a lattice with n elements.

- Every element of the lattice lies in exactly one block.



■ **Figure 2** (a) A minimal fat node h is used as a block header during the decomposition. The downset of h is removed and the process repeats on $L \setminus \downarrow h$. (b) A block decomposition yields a set of disjoint principal blocks, each having a block header. The residual block consists of the lattice elements that are not below any block header.

■ **Algorithm 1** Block Decomposition (Intuitive Version).

INPUT: A partial lattice L on n elements and a positive integer k .

OUTPUT: A block decomposition of L with block size k .

```

1:  $i = 1$ 
2: while there exists a minimal fat node  $h$  do
3:    $B_i = \downarrow h \cap L$ 
4:    $L = L \setminus B_i$ 
5:    $i = i + 1$ 
6:  $B_{\text{res}} = L$ 
    
```

- There can be at most n/k principal blocks as each one has size between k and n . Consequently, there are at most n/k block headers.
- Since the block headers are chosen to be *minimal* fat nodes, every other element is thin relative to the block it lies in. That is, if x lies in a block B and x is not the block header of B , then $|\downarrow x \cap B| < k$.

The last fact motivates the following term, which we will use frequently.

▶ **Definition 3.** *The local downset of an element x is the set $\downarrow x \cap B$, where B is the block containing x .*

Restated, the last property listed above says that the local downset of any element that is not a block header has size less than k . We also note that if h is the block header of a principal block B , then the local downset of h is B .

Somewhat less obvious is the following lemma.

▶ **Lemma 4.** *Every block is a partial lattice.²*

² Here the partial order on a block is inherited from the order on L .

Proof. The lemma follows from two facts.

1. The downset of any element in a partial lattice is also a partial lattice.
2. If the downset of an element is removed from a partial lattice, then the remaining elements still form a partial lattice.

We prove the first fact. Let h be an element of a partial lattice L . We prove that the poset $\downarrow h$ satisfies the lattice property (see Figure 1). Suppose there are four elements $x_1, x_2, y_1, y_2 \in \downarrow h$ such that $x_1, x_2 < y_1, y_2$. These elements also lie in L , and since L is a lattice there must be an element $z \in L$ such that $x_1, x_2 \leq z \leq y_1, y_2$. As $z \leq y_1 \leq h$, z must lie in $\downarrow h$. Thus $\downarrow h$ is a partial lattice because it satisfies the lattice property.

The second fact is similar. Suppose $\downarrow h$ is removed from a partial lattice L . If there are four elements $x_1, x_2, y_1, y_2 \in L \setminus \downarrow h$ with $x_1, x_2 < y_1, y_2$, then there must be an element $z \in L$ with $x_1, x_2 \leq z \leq y_1, y_2$. This element z cannot lie in $\downarrow h$ because $x_1 \leq z$ and $x_1 \notin \downarrow h$. Therefore $z \in L \setminus \downarrow h$. ◀

► **Remark 5.** *To avoid confusion in our notation, all lattice relations and operators are assumed to be with respect to L . In particular, \wedge , \vee , \uparrow , and \downarrow always reference the full lattice and are not restricted to a single block.*

4.2 Intuition for Block Decompositions

We can now explain intuitively why a block decomposition is a good idea and how it leads to an effective data structure. Lemma 4 means that the blocks can be treated as independent partial lattices. Moreover, the elements within each block are all thin, with the noteworthy exception of the block headers. For any single block, this thinness condition makes it possible to create a fast, simple, space-efficient data structure that facilitates computations within that block. However, such a data structure only contains local information about its block; it cannot handle operations that span multiple blocks.

For those operations, we rely upon the block headers to bridge the gaps. The block headers are significant because they induce a *unique representative property* on the blocks: If h is the block header of some principal block B and x is some element of the lattice, then we think of $x \wedge h$ as the representative of x in block B . For all of the operations that we care about, the representative of x in B faithfully serves the role of x during computations within B .

Combining the power of the unique representative property with our ability to quickly perform block-local operations gives us an effective data structure for lattices, which we are now prepared to describe.

5 A Data Structure for Order Testing

First, we describe a simple data structure that performs order-testing queries (answers “Is $x \leq y$?”) in constant time. We later extend it to handle meet and join queries as well.

Given a partial lattice L with n elements, we perform a block decomposition on L using the block size $k = \sqrt{n}$. Let B_1, B_2, \dots, B_m , and B_{res} be the blocks of this decomposition and h_1, \dots, h_m be the block headers. Note that $m \leq \sqrt{n}$.

5.1 Information Stored

We represent each element of L by a node with two fields.³ One field contains a unique *identifier* for the lattice element, a number between 0 and $n - 1$, for indexing purposes. The other field indicates the block that the element belongs to.

Our data structure consists of (\mathcal{A}) , a collection of arrays, and (\mathcal{B}) , a collection of dictionaries.

- (\mathcal{A}) For each block header h_i , we store an array containing a pointer to the node $h_i \wedge x$ for each $x \in L$. The meet of any node with any block header can be found with one access to the appropriate array.
- (\mathcal{B}) For each $x \in L$ we store a dictionary $\text{DOWN}(x)$ containing the identifiers of all the nodes in the local downset of x . By using a space-efficient static dictionary (e.g. [3]), membership queries can be performed in constant time. With this, we can test the order between any two nodes in the same block in constant time.

5.2 Testing Whether $x \leq y$

Given nodes x and y in L , we can test whether $x \leq y$ in three cases.

Case 1: If x is in a principal block B_i , then find $y_i = h_i \wedge y$ using (\mathcal{A}) . If $y_i \in B_i$, then $x \leq y$ if and only if x is a member of $\text{DOWN}(y_i)$; this can be tested using (\mathcal{B}) . If $y_i \notin B_i$, then $x \not\leq y$.

Case 2: If $x \in B_{\text{res}}$ and $y \in B_{\text{res}}$, then $x \leq y$ if and only if x is a member of $\text{DOWN}(y)$.

Case 3: If $x \in B_{\text{res}}$ and $y \notin B_{\text{res}}$, then $x \not\leq y$.

The three cases can be tested in constant time using (\mathcal{A}) and (\mathcal{B}) .

► **Proposition 6.** *The above method correctly answers order queries.*

Proof. Clearly the three cases cover all possibilities for x and y .

In Case 1, $y_i = h_i \wedge y$ has the property that $x \leq y$ if and only if $x \leq y_i$. This property holds because $x \leq h_i$ by assumption, and by the definition of meet,

$$x \leq h_i \wedge y \text{ if and only if } x \leq h_i \text{ and } x \leq y.$$

If $y_i \in B_i$, then the order can be tested directly using $\text{DOWN}(y_i)$. If $y_i \notin B_i$, then y_i cannot be above x in the lattice because $y_i \leq h_i$ and every element between x and h_i must lie in B_i .

Case 2 is checked directly using (\mathcal{B}) .

Case 3 is correct because B_{res} consists of all elements that are not below any block header. As y is in some principal block, it must lie below some block header. Hence, x cannot be below y . ◀

5.3 Space Complexity

Storing the n nodes of the lattice requires $\Theta(n)$ space. Each array of (\mathcal{A}) requires $\Theta(n)$ space and there are at most \sqrt{n} block headers, yielding $O(n^{3/2})$ space in total.

Assuming (\mathcal{B}) uses a succinct static dictionary (see [3]), the space usage for (\mathcal{B}) will be proportional to the sum of $|\downarrow x \cap B_x|$ over all $x \in L$, where B_x is the block containing x . If x is not a block header, then $|\downarrow x \cap B_x| < \sqrt{n}$ because the local downsets must be smaller

³ We often use the term “node” to refer to the element of L that the node represents.

than the block size of the decomposition. There are $n - m$ such elements, as m denotes the number of principal blocks. If x is a block header, then $\downarrow x \cap B_x = B_x$. Thus

$$\sum_{x \in L} |\downarrow x \cap B_x| \leq (n - m)\sqrt{n} + \sum_{i=1}^m |B_i| \leq (n - m)\sqrt{n} + n \leq 2n^{3/2}.$$

The total space for the data structure is therefore $O(n^{3/2})$.

6 Finding Meets and Joins

We now extend the order-testing data structure of the last section to answer meet queries: Given two elements x and y in L , we wish to find $x \wedge y$. Our data structure can answer these queries in $O(n^{3/4})$ time.

6.1 Subblock Decompositions

Let B_i be a principal block with block header h_i . A *subblock decomposition* of B_i is simply a block decomposition of $B_i \setminus \{h_i\}$.

To state it explicitly, the subblock decomposition is a partition of $B_i \setminus h_i$ into a set of principal subblocks $\{S_{i,1}, S_{i,2}, \dots, S_{i,\ell_i}\}$, each having a subblock header $g_{i,j}$, and one residual subblock $S_{i,\text{res}}$. The decomposition strategy is identical to that of a block decomposition, and it still depends on a *subblock size* r that we specify.

We exclude h_i from the subblock decomposition as a convenience. We want to use the property that the local downsets of the elements in B_i have size less than \sqrt{n} , and this holds for every element of B_i except for h_i .

Obviously, the subblocks have the same properties as blocks.

- Each principal subblock $S_{i,j}$ is a subset of B_i with $|S_{i,j}| \geq r$. Hence, $\ell_i \leq \frac{|B_i|}{r}$.
- If $x \in S_{i,j} \setminus \{g_{i,j}\}$ then $|\downarrow x \cap S_{i,j}| < r$.
- If $x \in S_{i,\text{res}}$ then $|\downarrow x \cap S_{i,\text{res}}| < r$.
- Each subblock is a partial lattice.

6.2 Extending the Data Structure

As before, let B_1, B_2, \dots, B_m , and B_{res} be the blocks of the decomposition of L , each having size at least \sqrt{n} . Within each principal block B_i , we perform a subblock decomposition with subblock size $r = \sqrt{|B_i|}$, yielding subblocks $S_{i,1}, S_{i,2}, \dots, S_{i,\ell_i}$, and $S_{i,\text{res}}$. We have $\ell_i \leq \sqrt{|B_i|}$ for $1 \leq i \leq m$. There is a subblock header $g_{i,j}$ for each principal subblock $S_{i,j}$, $1 \leq i \leq m$ and $1 \leq j \leq \ell_i$.

6.3 Information Stored

We add a new field to each node that indicates which subblock contains it. We store (\mathcal{A}) and (\mathcal{B}) as in the order-testing structure, and additionally:

- (\mathcal{C}) For each subblock header $g_{i,j}$, we store an array containing a pointer to $g_{i,j} \wedge x$ for all $x \in B_i$. These arrays allow us to determine the meet of any subblock header and any node in the same block with a single access.
- (\mathcal{D}) For each principal subblock $S_{i,j}$, we store a table that contains the meet of each pair of elements from $S_{i,j}$, unless the meet lies outside $S_{i,j}$. That is, the table has $|S_{i,j}|^2$ entries indexed by pairs of elements in $S_{i,j}$. The entry for (x, y) contains a pointer to $x \wedge y$ if it lies in $S_{i,j}$, or **null** otherwise. We can compute meets within any principal subblock in constant time using these tables.

- (\mathcal{E}) For every element x in a residual subblock $S_{i,\text{res}}$, we store $\downarrow x \cap S_{i,\text{res}}$ as a linked list of pointers. This allows us to iterate through the local downset of each element in the residual subblock.

6.4 Finding the Meet

This data structure allows us to find the meet of two elements $x, y \in L$ in $O(n^{3/4})$ time. The meet-finding operation works by finding representative elements for x and y in each principal block and computing the meet of each pair of representatives. We call these *candidate meets* for x and y . Once the set of candidate meets is compiled, the algorithm finds the largest element among them (with respect to the lattice order) and returns it.

We refer to the algorithm as MEET. This algorithm uses a subroutine called MEET-IN-BLOCK that finds the meet of two elements from the same principal block, or else determines that the meet does not lie within that block. The subroutine is similar to the main procedure except that it works on the subblock level instead of the block level.

■ Algorithm 2 Meet.

MEET: Given $x, y \in L$, find $x \wedge y$.

- (a) Initialize an empty set Z to store candidate meets for x and y .
 - (b) *Check principal blocks:* For each principal block B_i , find the representative elements $x_i = x \wedge h_i$ and $y_i = y \wedge h_i$ using (\mathcal{A}). If $x_i \in B_i$ and $y_i \in B_i$, then use the subroutine MEET-IN-BLOCK to either find $x_i \wedge y_i$ or determine that B_i does not contain it. If $x_i \wedge y_i$ is found, then add it to Z .
 - (c) *Check residual block:* If x and y are both in the residual block B_{res} , then use $\text{DOWN}(x)$ to iterate through every element $z \in \downarrow x \cap B_{\text{res}}$. Add z to Z whenever $z \leq y$.
 - (d) Using the order-testing operation, determine the maximum element in Z and return it. If Z is empty, then conclude that the meet of x and y does not exist and return **null**.
-

■ Algorithm 3 Meet-In-Block.

MEET-IN-BLOCK: Given $x_i, y_i \in B_i$, either find $x_i \wedge y_i \in B_i$ or determine that $x_i \wedge y_i \notin B_i$.

- (a) If $x_i = h_i$ or $y_i = h_i$, then return the smaller of x_i and y_i . Otherwise, initialize an empty set Z_i to store candidate meets for x_i and y_i in B_i .
- (b) *Check principal subblocks:* For each principal subblock $S_{i,j}$, find the representative elements $x_{i,j} = x_i \wedge g_{i,j}$ and $y_{i,j} = y_i \wedge g_{i,j}$ using (\mathcal{C}). If $x_{i,j}$ and $y_{i,j}$ are both in $S_{i,j}$, then look up

$$z_{i,j} = \begin{cases} x_{i,j} \wedge y_{i,j} & \text{if } x_{i,j} \wedge y_{i,j} \in S_{i,j} \\ \text{null} & \text{otherwise} \end{cases}$$

using the appropriate table in (\mathcal{D}). If $z_{i,j} \neq \text{null}$ then add it to Z_i .

- (c) *Check residual subblock:* If x_i and y_i are both in the residual subblock $S_{i,\text{res}}$, then use (\mathcal{E}) to iterate through every element $z \in \downarrow x_i \cap S_{i,\text{res}}$. Add z to Z_i whenever $z \leq y_i$.
 - (d) Using the order-testing operation, determine the largest node in Z_i and return it. If Z_i is empty, then conclude that $x_i \wedge y_i \notin B_i$ and return **null**.
-

6.5 Correctness

We now prove that this algorithm is correct, beginning with the correctness of MEET-IN-BLOCK.

► **Lemma 7.** *MEET-IN-BLOCK returns $x_i \wedge y_i$ if it lies in B_i and **null** otherwise.*

Proof. If $x_i = h_i$ or $y_i = h_i$, then $x_i \wedge y_i$ is returned in step (i). Otherwise, the correctness of the algorithm relies on two facts.

Fact 1. Every element $z \in Z_i$ satisfies $z \leq x_i \wedge y_i$.

Fact 2. If $x_i \wedge y_i$ exists and lies in B_i , then it is added to Z .

Assuming these hold, step (iv) must correctly answer the query: In the case that $x_i \wedge y_i \in B_i$, the meet must be added to Z_i and it must be the maximum element among all elements in Z_i . If $x_i \wedge y_i \notin B_i$, then Z_i will be empty by the first fact.

Fact 1 is straightforward. Every candidate meet z added to Z_i in step (ii) is $x_{i,j} \wedge y_{i,j}$ for some $j \in \{1, \dots, \ell_i\}$, as reported by (\mathcal{D}) . Since $x_{i,j} \leq x_i$ and $y_{i,j} \leq y_i$ we have $z \leq x_i \wedge y_i$. When a candidate meet z is added to Z in step (iii) it is because $z \in \downarrow x_i \cap B_{\text{res}}$ and $z \leq y_i$; hence $z \leq x_i \wedge y_i$.

To prove Fact 2, first suppose that $x_i \wedge y_i$ lies in a principal subblock $S_{i,j}$. Then $x_i \wedge y_i \leq g_{i,j}$. By the elementary properties of the meet operation,

$$x_i \wedge y_i = x_i \wedge y_i \wedge g_{i,j} = (x_i \wedge g_{i,j}) \wedge (y_i \wedge g_{i,j}) = x_{i,j} \wedge y_{i,j}.$$

Thus, $x_i \wedge y_i$ is added to Z during step (ii) when the subblock $S_{i,j}$ is considered.

Now suppose that $x_i \wedge y_i$ lies in the residual subblock $S_{i,\text{res}}$. In this case, x_i and y_i must themselves lie in $S_{i,\text{res}}$, for if either one is below any subblock header of B_i then their meet would also be below that same block header. Thus, $x_i \wedge y_i$ will be added to Z_i in step (3) during which every element of $\downarrow x_i \cap \downarrow y_i \cap S_{i,\text{res}}$ is added to Z_i . This proves Fact 2. ◀

► **Lemma 8.** *MEET finds $x \wedge y$ or correctly concludes that it does not exist.*

Proof. This proof is similar to that of Lemma 7. It relies on the same two facts.

Fact 1. Every element $z \in Z$ satisfies $z \leq x \wedge y$.

Fact 2. If $x \wedge y$ exists, then it is added to Z .

Assuming these hold, step (4) must correctly answer the query. The only significant difference between MEET and MEET-IN-BLOCK is the method of finding candidate meets in step (2). MEET calls MEET-IN-BLOCK to find $x_i \wedge y_i$ if it lies in B_i whereas MEET-IN-BLOCK uses (\mathcal{D}) to find $x_{i,j} \wedge y_{i,j}$ if it lies in $S_{i,j}$. By Lemma 7, MEET-IN-BLOCK accurately returns $x_i \wedge y_i$ if $x_i \wedge y_i \in B_i$ and **null** otherwise. Now Facts 1 and 2 may be proved by the same arguments. ◀

6.6 Time Analysis

The meet procedure takes $O(n^{3/4})$ time in the worst case. We first analyze the time for MEET-IN-BLOCK applied to a principal block B_i . Step (i) takes constant time. Step (ii) takes constant time per principal subblock of B_i using (\mathcal{C}) and (\mathcal{D}) . Since each principal subblock has size at least $\sqrt{|B_i|}$, there are at most $|B_i|/\sqrt{|B_i|} = \sqrt{|B_i|}$ principal subblocks; hence the time for step (ii) is $O(\sqrt{|B_i|})$. Step (iii) performs constant-time order testing on all the elements below x_i in the residual subblock. By the subblock decomposition method, there are at most $\sqrt{|B_i|}$ such elements.

When step (iv) is reached, Z_i has been populated with at most one element per principal subblock ($\sqrt{|B_i|}$ in total) and at most $\sqrt{|B_i|}$ elements from the residual subblock. The maximum element in Z_i is found in linear time during this step. Thus, MEET-IN-BLOCK runs in $O(\sqrt{|B_i|})$ time when applied to block B_i .

31:12 Space-Efficient Data Structures for Lattices

Now the main procedure can be analyzed in a similar fashion. Step (1) takes constant time. Step (2) calls MEET-IN-BLOCK on every principal block, and hence the total time for step (2) is proportional to $\sum_{i=1}^m \sqrt{|B_i|}$. By Jensen's inequality, $\sum_{i=1}^m \sqrt{|B_i|}$ is maximized when all the blocks have size \sqrt{n} , since each principal block has size at least \sqrt{n} and $\sum_{i=1}^m |B_i| \leq n$. Thus

$$\sum_{i=1}^m \sqrt{|B_i|} \leq \sum_{i=1}^{\sqrt{n}} n^{1/4} \leq n^{3/4}.$$

As in the analysis of steps (iii) and (iv), steps (3) and (4) take $O(\sqrt{n})$ time. Therefore the time complexity of MEET is $O(n^{3/4})$.

6.7 Space Complexity

The space required to store the nodes, (\mathcal{A}) , and (\mathcal{B}) is $O(n^{3/2})$ as in Section 5.

Fix $i \in \{1, \dots, m\}$. We show that the parts of (\mathcal{C}) , (\mathcal{D}) , and (\mathcal{E}) relating to B_i occupy $O(|B_i|\sqrt{n})$ space. Since $\sum_{i=1}^m |B_i| \leq n$, it follows that the entire data structure takes $O(n^{3/2})$ space.

Each array in (\mathcal{C}) requires $O(|B_i|)$ space. There are at most $\sqrt{|B_i|}$ subblock headers for a total of $O(|B_i|^{3/2})$ space.

The lookup table in (\mathcal{D}) for subblock $S_{i,j}$ takes $O(|S_{i,j}|^2)$ space. Since $\sqrt{|B_i|} \leq |S_{i,j}| \leq \sqrt{n}$, we have $\sum_{j=1}^{\ell_i} |S_{i,j}|^2 \leq \sqrt{n} \sum_{j=1}^{\ell_i} |S_{i,j}|$. Notice $\sum_{j=1}^{\ell_i} |S_{i,j}| \leq |B_i|$ as the subblocks are disjoint subsets of B_i . Therefore the total space occupied by (\mathcal{D}) is $O(|B_i|\sqrt{n})$.

The lists stored by (\mathcal{E}) occupy $O(\sqrt{|B_i|})$ space each for a total of $O(|B_i|^{3/2})$ space. The space charged to block B_i is therefore $O(|B_i|^{3/2} + |B_i|\sqrt{n} + |B_i|^{3/2}) = O(|B_i|\sqrt{n})$.

6.8 Preprocessing

It remains to discuss how to efficiently decompose the lattice and initialize the structures $(\mathcal{A}) - (\mathcal{E})$. Recall that we the lattice is presented initially by its transitive reduction graph. It is known that the number of edges in the TRG of a lattice is $O(n^{3/2})$ [14, 25]. We assume that the TRG is stored as a set of n nodes, each with a list of its out-neighbours (nodes that cover it) and a list of in-neighbours (nodes that it covers). The total space needed for this representation is $O(n^{3/2})$. The preprocessing takes $O(n^2)$ time and the space usage never exceeds $O(n^{3/2})$.

The first step in preprocessing is to determine the block decomposition. The same technique will apply to subblock decompositions. We begin by computing a *linear extension* of the lattice. A linear extension of a partially-ordered set is an order of the elements x_1, x_2, \dots, x_n such that if $i \leq j$ then $x_j \not\prec x_i$. A linear extension may be found by performing a topological sort on the TRG, which can be done in $O(n^{3/2})$ time [12].

We now visit each element of L in the order of this linear extension and determine the size of its downset. The size of the downset can be computed by a depth-first search beginning with the element and following edges descending the lattice. This search takes time proportional to the number of edges between elements in the downset. As soon as this process discovers a fat node h (a node with at least \sqrt{n} elements in its downset), it can be used as a block header. Then h and every element of its downset can be deleted from L . The process of computing the sizes of the downsets can continue from the node following h in the linear extension, and the only difference is that the graph searches used to compute the size of each downset must now be restricted to $L \setminus \downarrow h$. There is no need to recompute the downset size of any node before h in the linear extension because the size of its downset

was less than \sqrt{n} previously and deleting $\downarrow h$ can only reduce this value. The fat nodes encountered in this way form the block headers of the decomposition. After every node has been visited, the remaining elements can be put into the residual block.

The time needed for the decomposition depends on the number of edges in each downset. By Lemma 4, every downset is a partial lattice, and thus a downset with k nodes can have only $O(k^{3/2})$ edges. For every thin node encountered, the number of edges in the downset is at most $O((\sqrt{n})^{3/2}) = O(n^{3/4})$ because it contains less than \sqrt{n} elements. Thus, the time needed to visit all the thin nodes is $O(n^{7/4})$.

Whenever a fat node is discovered its downset is removed immediately, and so the edges visited during the DFS are never visited again. Hence, the time needed to examine all of the block headers is proportional to the number of edges in the whole TRG. Therefore a block decomposition can be computed in $O(n^{7/4})$ time.

By the same procedure, the subblocks can be computed in $O(\sum_{i=1}^m |B_i|^{7/4})$ time. Since $\sum_{i=1}^m |B_i| \leq n$, this is at most $O(n^{7/4})$.

With the block and subblock decompositions in hand, data structures $(\mathcal{A}) - (\mathcal{E})$ can be initialized. See Appendix A for details.

We have now proven the main theorem of this paper.

► **Theorem 9.** *There is a data structure for lattices that requires $O(n^{3/2})$ space, answers order-testing queries in $O(1)$ time, and computes the meet or join of two elements in $O(n^{3/4})$ time. The preprocessing time starting from the transitive reduction graph of the lattice is $O(n^2)$.*

A straightforward generalization of the data structure allows for a space-time tradeoff.

► **Corollary 10.** *For any $c \in [\frac{1}{2}, 1]$, there is a data structure for lattices that requires $O(n^{1+c})$ space and computes the meet or join of two elements in $O(n^{1-c/2})$ time. The preprocessing time, starting from the transitive reduction graph of the lattice, is $O(n^2 + n^{1+3c/2})$.*

Proof. The modification is obtained by adjusting the block size of the initial decomposition from \sqrt{n} to n^c . Otherwise, the data structure and methods are identical. The time, space, and preprocessing analyses are similar. ◀

Note that for $c = \frac{1}{2}$, this data structure is precisely that of Theorem 9.

7 Degree-Bounded Extensions

Recall that we assume that the lattice is initially represented by its transitive reduction graph (TRG). Let the *degree* of a lattice node be the number of in-neighbours in the TRG, or equivalently, the number of nodes it covers. Interestingly, developing methods that handle high-degree nodes efficiently has been the primary obstacle to improving on our data structure. Indeed, the “dummy node” technique of Talamo and Vocca, explained in Appendix B, is effectively used to get around high-degree lattice elements. We have found that meets and joins can be computed more efficiently as long as the maximum degree of any node in the lattice is not too large. This is the case for distributive lattices, for example, as $\log_2 n$ is the maximum degree of a node in a distributive lattice⁴. In this section, we explore new data structures for meet and join operations that perform well under this assumption.

⁴ We leave this as an exercise using Birkhoff’s Representation Theorem [2].

Let d be the maximum degree of any node in a partial lattice L . As a convenience, we assume in this section that L has a top element. The purpose of this assumption is to avoid a lattice with more than d maximal elements; otherwise we would need to define d as the larger of the maximum degree and the number of maximal elements in the lattice.

This assumption has the effect that the residual block in any block decomposition of L has a top element (unless it is empty). The only practical difference between the residual block and a principal block is that the residual block may be smaller than the block size of the decomposition. The results of this section are easier to relate if we assume henceforth that all blocks are principal blocks and each has a block header. Thus, a block decomposition with block size k creates m blocks B_1, \dots, B_m with block headers h_1, \dots, h_m , where $|B_i| \geq k$ for $1 \leq i \leq m - 1$. The number of blocks is at most $\frac{n}{k} + 1$.

We begin with a simple data structure that computes joins between elements using a new strategy. It is more efficient than our earlier method when $d \leq n^{3/4}$. We then generalize the idea to create a more sophisticated recursive data structure. It improves on the simple structure for all values of d and works especially well when $d \leq \sqrt{n}$. The space usage is $O(n^{3/2})$ for both data structures. Either one can be used to compute meets as well by inverting the lattice order and rebuilding the data structure, although the value of d may be different in the flipped lattice.

► **Theorem 11.** *There is a data structure for lattices that requires $O(n^{3/2})$ space and computes the join of two elements in $O(\sqrt{n} + d)$ time.*

Proof. This data structure uses a block decomposition with block size $k = \sqrt{n}$ and stores (\mathcal{A}) and (\mathcal{B}) just as in Section 5. This is everything we need to perform order-testing in constant time. However, we now use this information to compute *joins* instead of meets.

Let B_1, \dots, B_m be the blocks of the decomposition with block headers h_1, \dots, h_m . Further assume that the order B_1, B_2, \dots, B_m reflects the order that the blocks were extracted from L during the decomposition.

Given $x, y \in L$, $x \vee y$ may be found as follows.

- (1) Use order-testing to compare x and y to every block header. Let $i^* \in \{1, \dots, m\}$ be the smallest value for which $x \leq h_{i^*}$ and $y \leq h_{i^*}$.
- (2) It must be that $x \vee y$ lies in B_{i^*} . Let $c_1, c_2, \dots, c_t \in B_{i^*}$ be the elements covered by h_{i^*} in B_{i^*} ⁵. Compare x and y to each of these elements using order-testing queries. If $x, y \leq c_j$ for some $j \in \{1, \dots, t\}$, then proceed to step (3). Otherwise, conclude that $x \vee y = h_{i^*}$.
- (3) The join of x and y must lie in the local downset of c_j . Find $x \vee y$ by comparing x and y to every element in $\downarrow c_j \cap B_{i^*}$ and choosing the smallest node z with $x, y \leq z$.

This procedure always finds $x \vee y$. The purpose of step (1) is to identify the block containing $x \vee y$. With i^* defined as in the algorithm, observe that $x \vee y$ must have been added to B_{i^*} during the decomposition because $x \vee y \in \downarrow h_{i^*}$ and $x \vee y \notin \downarrow h_i$ for any $i < i^*$. This step takes $O(\sqrt{n})$ time as $m \leq \sqrt{n} + 1$.

Once B_{i^*} has been identified, the difficulty lies in finding the join. The algorithm checks all of the children c_1, \dots, c_t of h_{i^*} to find an element c_j above $x \vee y$. This step requires $O(d)$ time as $t \leq d$. If the algorithm succeeds in finding c_j then it compares x and y with all of the elements in the local downset of c_j to determine the join. By the thinness property, this step takes only $O(\sqrt{n})$ time. If no such c_j exists, then h_{i^*} must be the only element in B_{i^*} above both x and y . Thus, this data structure finds $x \vee y$ in $O(\sqrt{n} + d)$ time. ◀

⁵ It is possible that h_{i^*} covers other elements belonging to earlier blocks. These are not included.

► **Theorem 12.** *There is a data structure for lattices that requires $O(n^{3/2})$ space and computes the join of two elements in $O(d \frac{\log n}{\log d})$ time.*

Proof. We extend the ideas of Theorem 11 using a recursive decomposition of a lattice.

The recursive decomposition works in two stages. First, we perform a block decomposition of L using the block size n/d . This produces up to $d + 1$ blocks B_1, \dots, B_m .

We decompose each B_i further using a *cover decomposition*. If B_i has a block header h_i and $c_1, c_2, \dots, c_t \in B_i$ are the elements covered by h_i , then a cover decomposition of B_i is a partition of B_i into the sets

$$C_{i,j} = (\downarrow c_j \cap B_i) \setminus \left(\bigcup_{\ell=1}^{j-1} \downarrow c_\ell \right) \text{ for } 1 \leq j \leq t.$$

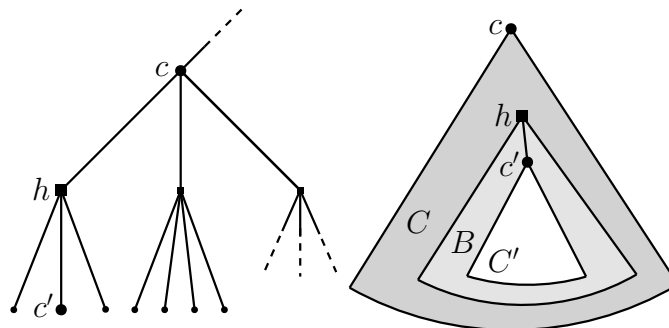
We call these sets *chunks* to avoid overloading “block” and we call c_j the *chunk header* of $C_{i,j}$. Unlike a block decomposition, a cover decomposition does not depend on a block size. It is unique up to the ordering of c_1, \dots, c_t .

So far, our decomposition produces blocks $\{B_1, \dots, B_m\}$ and chunks $\{C_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq \text{deg}(h_i)\}$. We recursively decompose every chunk $C_{i,j}$ in the same two stages, first by a block decomposition with block size $\frac{|C_{i,j}|}{d}$ and then by a cover decomposition of each of the resulting blocks. The recursive decomposition continues in this fashion on any chunk with size at least $2d$.

The recursion induces a tree structure on the set of block headers and chunk headers in the lattice. The children of each block header are the chunk headers chosen during its decomposition and vice versa. The order of the children of a node corresponds to the order that the blocks or chunks are taken during the decomposition. Finally, we create one special node to act as the root of the tree. The children of the root are the block headers of the initial decomposition. We call this the *decomposition tree*.

It is easy to see that every lattice element occurs at most once in the tree and that the maximum degree of any tree node is at most $d + 1$. Less obvious is the fact that the depth of the tree is $O(\frac{\log n}{\log d})$.

To see this, let c be a chunk header, let h be one of its children in the tree, and let c' be a child of h . Assume c is the header of a chunk C , h is the header of a block B contained in C , and c' is the header of a chunk C' contained in B ; see Figure 3. Block B was formed during a block decomposition of C with size $|C|/d$. Since h covers c' in B , c' must have been a thin node during that decomposition. The chunk C' was then created from the local downset of c' in B . Thus $|C'| \leq |\downarrow c' \cap B| \leq |C|/d$.



■ **Figure 3** Three nodes in the decomposition tree and the corresponding chunks and block of the recursive decomposition. The size of C' can be no larger than $|C|/d$.

31:16 Space-Efficient Data Structures for Lattices

This implies that the size of chunks decreases by a factor of d between every chunk header and its grandchildren in the decomposition tree. After $2^{\lceil \frac{\log n}{\log d} \rceil}$ generations in the decomposition tree, every chunk must have size less than $2d$. This proves the claim.

The data structure is now simple to describe. We store the decomposition tree and, for each leaf, we store a list of the elements in the chunk of that chunk header. Since the chunks represented by leaves are pairwise disjoint, only $O(n)$ space is needed for this structure. Additionally, we create and store the order-testing structure of Section 5, bringing the total space to $O(n^{3/2})$.

The join of two elements can be found using a recursive version of the algorithm from Theorem 11. Suppose we are given $x, y \in L$ and must determine $x \vee y$. Through a variable u that represents the node being considered, we recursively traverse the decomposition tree. Initially set u equal to the root and proceed as follows.

7.1 Base Case

If u is a leaf, then consider the stored list of elements for u . Find $x \vee y$ by comparing x and y to every element in the list and returning the smallest node z with $x, y \leq z$.

7.2 Recursive Case

If u is not a leaf, let v_1, v_2, \dots, v_k be the children of u in the decomposition tree, listed in order. Use order-testing to compare x and y to each v_i . If there is no v_i such that $x \leq v_i$ and $y \leq v_i$, then conclude that $x \vee y = u$. Otherwise, let $i^* \in \{1, \dots, k\}$ be the smallest value for which $x \leq v_{i^*}$ and $y \leq v_{i^*}$. Recurse on v_{i^*} .

This procedure spends $O(d)$ time on each node. In the base case, the list stored for u has length $O(d)$ and the join can be found in this list in linear time. The recursive case takes $O(d)$ time as well since the maximum degree of the decomposition tree is at most $d + 1$. As the depth of the tree is $O(\frac{\log n}{\log d})$, the total time of this procedure is $O(d \frac{\log n}{\log d})$.

Correctness is a consequence of the fact that $x \vee y$ lies in the first block of each block decomposition whose header is above both x and y . The same fact holds for the chunks in a cover decomposition. Thus, each time i^* is chosen in the recursive case, it must be that $x \vee y$ lies in the block or chunk for v_{i^*} . ◀

8 Conclusions

We have presented a data structure to represent lattices in $O(n^{3/2})$ words of space, which is within a $\Theta(\log n)$ factor of optimal. It answers order queries in constant time and meet or join queries in $O(n^{3/4})$ time. This work is intended to replace the earlier solution to this problem which was incorrect; see Appendix B for a discussion of the error. Our degree-bounded data structure uses $O(n^{3/2})$ space and answers meet or join queries in $O(d \frac{\log n}{\log d})$ time. For some low-degree lattices, this structure improves dramatically on our subblock-based approach. Ours are the only data structures known to us that uses less than the trivial $O(n^2)$ space.

We wonder what can be done to improve on our results. The time to answer meet and join queries may yet be reduced, perhaps to the $O(\sqrt{n})$ bound claimed by [22]. Another natural question is whether the space of the representation can be reduced to the theoretical minimum of $\Theta(n^{3/2})$ bits.

References

- 1 Hassan Ait-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1):115–146, 1989.
- 2 Garrett Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
- 3 Andrej Brodnik and J Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.
- 4 Yves Caseau. Efficient handling of multiple inheritance hierarchies. *ACM SIGPLAN Notices*, 28(10):271–287, 1993.
- 5 Yves Caseau, Michel Habib, Lhouari Nourine, and Olivier Raynaud. Encoding of multiple inheritance hierarchies and partial orders. *Computational Intelligence*, 15(1):50–62, 1999.
- 6 Marcel Ern e, Jobst Heitzig, and J urgen Reinhold. On the number of distributive lattices. *Electron. J. Combin.*, 9(1):23, 2002.
- 7 Arash Farzan and J Ian Munro. Succinct representation of finite abelian groups. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 87–92. ACM, 2006.
- 8 Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal concept analysis: foundations and applications*, volume 3626. Springer, 2005.
- 9 George Gr atzer and Friedrich Wehrung. *Lattice theory: special topics and applications*. Springer, 2016.
- 10 Michel Habib, Raoul Medina, Lhouari Nourine, and George Steiner. Efficient algorithms on distributive lattices. *Discrete Applied Mathematics*, 110(2):169–187, 2001.
- 11 Michel Habib and Lhouari Nourine. Tree structure for distributive lattices and its applications. *Theoretical Computer Science*, 165(2):391–405, 1996.
- 12 Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.
- 13 DJ Kleitman and KJ Winston. The asymptotic number of lattices. *Annals of Discrete Mathematics*, 6:243–249, 1980.
- 14 Walter Klotz and Lutz Lucht. Endliche verb ande. *Journal f ur die Reine und Angewandte Mathematik*, 247:58–68, 1971.
- 15 Andreas Krall, Jan Vitek, and R Nigel Horspool. Near optimal hierarchical encoding of types. In *European Conference on Object-Oriented Programming*, pages 128–145. Springer, 1997.
- 16 Bernard Monjardet. The presence of lattice theory in discrete problems of mathematical social sciences. Why? *Mathematical Social Sciences*, 46(2):103–144, 2003.
- 17 J Ian Munro and Corwin Sinnamon. Time and space efficient representations of distributive lattices. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 550–567. Society for Industrial and Applied Mathematics, 2018.
- 18 Flemming Nielson, Hanne R Nielson, and Chris Hankin. *Principles of program analysis*. Springer, 2015.
- 19 Mihai P atraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011.
- 20 Maurizio Talamo and Paola Vocca. Fast lattice browsing on sparse representation. In *Orders, Algorithms, and Applications*, pages 186–204. Springer, 1994.
- 21 Maurizio Talamo and Paola Vocca. A data structure for lattice representation. *Theoretical Computer Science*, 175(2):373–392, 1997.
- 22 Maurizio Talamo and Paola Vocca. An efficient data structure for lattice operations. *SIAM journal on computing*, 28(5):1783–1805, 1999.
- 23 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, 2004.
- 24 Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Springer, 1982 .
- 25 Kenneth James Winston. *Asymptotic analysis of lattices and tournament score vectors*. PhD thesis, Massachusetts Institute of Technology, 1979.

A

 Initializing the Data Structure

We now show how (\mathcal{A}) , (\mathcal{B}) , (\mathcal{C}) , (\mathcal{D}) , and (\mathcal{E}) can be constructed in $O(n^2)$ time. We assume that we have access to the TRG of the lattice and that the block and subblock decompositions have already been computed.

(A) Some care is required to construct (\mathcal{A}) efficiently. Let x_1, \dots, x_n be a linear extension of L . Consider a principal block B_i with block header h_i . To find $z \wedge h_i$ for each $z \in L$, we do the following.

1. Initialize an array of length n to store the meet of h_i with each element and populate the array with `null` in every entry.
2. Perform a DFS to find $\downarrow h_i$ in L . Note that $\downarrow h_i$ may be considerably larger than B_i . Put the elements of $\downarrow h_i$ (note that this includes h_i) into a linear extension y_1, \dots, y_k by restricting the linear extension of L to these elements.
3. Traverse the nodes in reverse order of this extension (beginning with y_k and ending at y_1). For each node y_j , perform a DFS on the *upset* of that node in the full lattice. For every node z visited during the DFS for node y_j , record that $z \wedge h_i = y_j$ in the array, and then mark z so that it will not be visited by later graph searches. After all the nodes in $\downarrow h_i$ have been processed, restore the lattice by unmarking all nodes.

By this method, the entry for $z \wedge h_i$ in the array is recorded to be the last element in the linear extension of $\downarrow h_i$ that is below z . This must be the correct node because it is below both z and h_i , and every other element below z and h_i occurs earlier in the linear extension. Whenever $z \wedge h_i$ does not exist in the lattice, the array entry for $z \wedge h_i$ is the default value `null`.

The time for this procedure is bounded by the number of edges in the TRG for L because no node is visited more than once over all of the graph searches. Recall that the number of edges in the TRG is $O(n^{3/2})$. Summing over all block headers, the total time to create (\mathcal{A}) is at most $O(n^{3/2} \sqrt{n}) = O(n^2)$.

(B) This can be computed by performing a DFS on the local downset of each node and adding the elements visited to a dictionary for that node.

Initializing and populating the space-efficient dictionary of [3] takes time linear in the number of dictionary entries. Excluding the block headers, the local downsets have at most \sqrt{n} nodes and $O(n^{3/4})$ edges; hence the time spent on all non-block headers is at most $O(n^{7/4})$. The local downsets of the block headers are all disjoint, so the total time required is $O(n^{7/4})$.

(C) Use the same method for (\mathcal{A}) restricted to each block to compute (\mathcal{C}) . The total time is $O(\sum_{i=1}^m |B_i|^2)$, which is no larger than $O(n^2)$.

(D) The method of (\mathcal{A}) can also be used to compute (\mathcal{D}) . For each element z in a principal subblock $S_{i,j}$, find the meet of z with every other element in the subblock in $O(|S_{i,j}|^{3/2})$ time, where z plays the role of h_i in the method for (\mathcal{A}) . It takes $O(|S_{i,j}|^{5/2})$ time to do this for every element in a single subblock and the total time is proportional to

$$\sum_{i=1}^m \sum_{j=1}^{\ell_i} |S_{i,j}|^{5/2} \leq \sum_{i=1}^m \sum_{j=1}^{\ell_i} |S_{i,j}| (\sqrt{n})^{3/2} \leq n^{7/4}.$$

The first inequality uses the fact that each subblock has size at most \sqrt{n} . The second inequality holds because the subblocks are disjoint.

(E) Each linked list can be constructed by performing a DFS on the downset of each element in a residual subblock. This takes $O(n^{7/4})$ time as in the analysis for (\mathcal{B}) .

B Correcting Earlier Work

As stated in the introduction, this paper relies on ideas from the lattice data structure of [20, 21, 22]. These papers contain a mistake that we believe is not easily repaired. The purpose of this section is to summarize their techniques, explain where the error occurs, and argue that it cannot be fixed by a minor modification. We urge the interested reader to consult [22] to confirm this analysis.

We restate their algorithm in the language of this paper. In the interest of a clear and concise explanation, we do not rebuild all the machinery of their work. In particular, we ignore their *double-tree* structure and we only consider blocks made from downsets (in their papers, blocks may be built from upsets or downsets). In our observation, the double-tree structure is necessary only as a null/non-null value check for order testing and meet/join queries (thus a simple dictionary suffices); further, while we have concerns about using both upsets and downsets for blocks, using downsets alone avoids such issues and still satisfies the requirements in their papers (Lemma 4.1 in [22]). We take these liberties for the purpose of quickly coming to the relevant issue. Readers will need to confirm for themselves that our explanation is fundamentally accurate.

Their method relies on a lattice decomposition to build the data structure, and our block decomposition is similar to the basic version of the decomposition described in their papers. Note that what we call “blocks” are called “ideals” in [21] and “clusters” in [22]. They do not decompose the lattice at a second level like our subblock decompositions. The error is introduced in the extended version of their lattice decomposition, which we now describe.

The intuition behind their data structure is that everything would be easier if every block had size $\Theta(\sqrt{n})$, say between \sqrt{n} and $2\sqrt{n}$. If this were the case, then we could afford to explicitly store the meet/join and reachability property between every pair of elements from the same block, as this would use roughly $\sum_{i=1}^{\sqrt{n}} (\sqrt{n})^2 = O(n^{3/2})$ space. This would allow the meet of two elements from the same block to be found in constant time by a simple table lookup. In terms of our meet-finding algorithm from Section 6, this would reduce the time for MEET-IN-BLOCK to a constant and the time for MEET to $O(\sqrt{n})$.

B.1 Dummy Nodes

A block decomposition by itself cannot guarantee anything about the sizes of the blocks except that each is at least \sqrt{n} . They attempt to simulate blocks of size \sqrt{n} by modifying the transitive reduction graph (TRG) of the lattice, creating “dummy nodes” with downsets of size $\Theta(\sqrt{n})$ to act as block headers when none exist naturally.

Dummy nodes are introduced as follows. Suppose a block B is created that has more than $2\sqrt{n}$ elements. Assume that the block header has children c_1, c_2, \dots, c_t in the TRG. Consider the sequence

$$|\downarrow c_1 \cap B|, |(\downarrow c_1 \cup \downarrow c_2) \cap B|, \dots, |(\downarrow c_1 \cup \dots \cup \downarrow c_t) \cap B|.$$

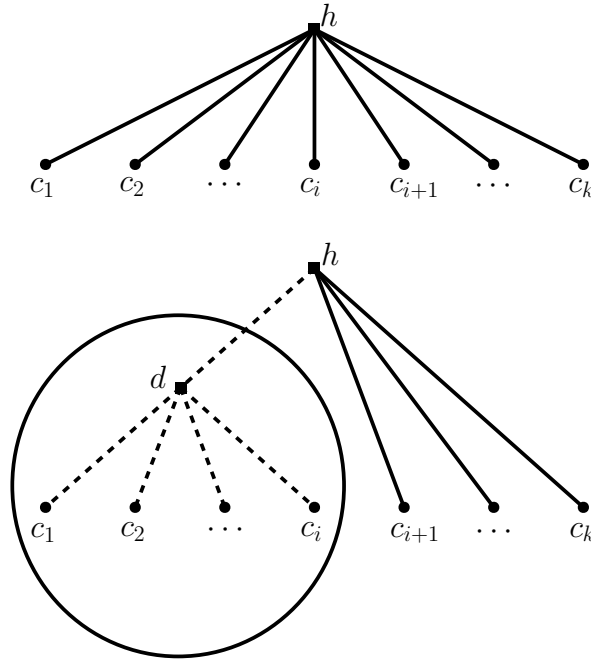
As each of the children is a thin element (its local downset has size less than \sqrt{n}), the difference between adjacent numbers in this sequence is less than \sqrt{n} . Thus, there is some $i \in \{1, \dots, k\}$ such that

$$\sqrt{n} \leq |(\downarrow c_1 \cup \dots \cup \downarrow c_i) \cap B| \leq 2\sqrt{n}.$$

The children c_1, \dots, c_i may be grouped together and the set $(\downarrow c_1 \cup \dots \cup \downarrow c_i) \cap B$ may be considered as an *artificial* block having size $\Theta(\sqrt{n})$. By removing this artificial block and

iterating on the remaining children, B is partitioned into a collection of artificial blocks with sizes between \sqrt{n} and $2\sqrt{n}$ (except that there may be one smaller block at the end). The only difference between these artificial blocks and ordinary principal blocks is that they lack a block header.

To remedy this, a dummy node is introduced at the top of each artificial block. That is, a new element d is created and inserted into the TRG with c_1, \dots, c_i as its in-neighbours and the block header of B as its only out-neighbour.



■ **Figure 4** Dummy nodes are inserted between the block header and its children to simulate blocks of size $\Theta(\sqrt{n})$.

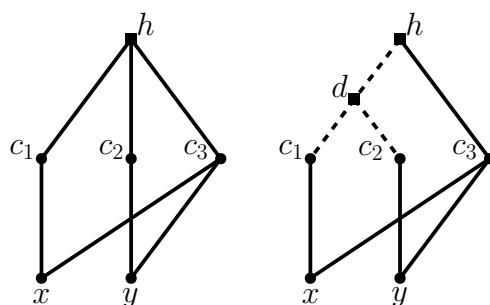
Talamo and Vocca rely on the fact that the graph still represents a partial lattice after adding dummy nodes in this way. They state on page 1794 of [22]:

“By construction, the dag obtained by adding dummy vertices still satisfies the lattice property.”

Unfortunately, this claim is not true in many cases. Consider the stripped-down example in Figure 5. The lattice on the left is changed to the graph on the right by introducing a dummy node as described. However, the graph on the right fails the lattice property because the join of x and y is not well-defined: Both c_3 and d are minimal among elements in $\uparrow x \cap \uparrow y$. Symmetrically, the meet of c_3 and d is not well-defined either. In this case, adding d broke the lattice property.

Although the example is on a very small lattice, it scales easily to any size. Any number of nodes could be added to the original lattice so that $|\downarrow c_1 \cup \downarrow c_2| \in [\sqrt{n}, 2\sqrt{n}]$. The dummy node added in this case would still violate the lattice property.

This detail is easy to overlook, especially since $\downarrow d \cap B$ is necessarily a partial lattice. However, the lattice property may fail in the larger structure when dummy nodes are added. This fundamentally impacts the correctness of their approach.



■ **Figure 5** Inserting a dummy node breaks the lattice property.

B.2 Impact Of The Error

With dummy nodes, it is no longer true that every element has a unique representative in each block. Talamo and Vocca use the following text on page 1789 of [22], “given an external vertex v , the pair $(v, Clus(c))$ univocally identifies a vertex $u \in Clus(c)$ representing either the $LUB(Clus^+(c) \cap Clus^+(v))$ or the $GLB(Clus^-(c) \cap Clus^-(v))$.” In the language of our paper, the claim is that for every block header h , any external element v must have a unique representative $x \wedge h$. Consider again the example in Figure 5 with d as the block header of its downset. The external element c_3 does not have a unique representative in the block headed by d , since the meet of d and c_3 is now undefined.

In [22], this breaks Lemma 3.1 when c is a dummy node, which in turn breaks Lemma 3.3 and implies their data structure C on page 1792 of [22] would need to keep multiple entries for an element-cluster pair in order to guarantee correctness of the reachability algorithm described below it. We see no reason why the number of such representatives stored per element should be small, nor that the total number of representatives stored should be small, which undermines both the proposed query and space complexities.

The same issue arises in Talamo and Vocca’s meet and join algorithms. The algorithm given relies on the unique representative of an element with a block, and without it, neither the $O(n\sqrt{n})$ space bound nor the $O(\sqrt{n})$ time bound on meet or join operations follow in Proposition 6.4 of [22].

Further, if dummy nodes are avoided altogether, the space bound can be $\Omega(n^2)$, as explained on page 1793 of [22].

It has been suggested to us that the issues may be avoided if the dummy nodes are not considered as actual nodes of the lattice itself, but instead as a construct to group small clusters together for a counting reason. That is, the claim is that an $O(n\sqrt{n})$ -space $O(1)$ -time order-testing structure can be made without tangibly introducing dummy nodes. As we have shown in this paper, this is indeed true. However, let us emphasize that the work of Talamo and Vocca does not achieve this. It describes a very different technique that crucially relies on the unique representative property remaining true after grouping clusters using dummy nodes, which does not hold in general regardless of whether dummy nodes are actually inserted into the graph or just used as a conceptual tool. With their techniques, we see no way to achieve their claimed $O(\sqrt{n})$ time meet/join algorithm without their erroneous dummy nodes. We give evidence in the following section as to why this might be infeasible.

B.3 Can It Be Fixed?

It is natural to search for a small change to the dummy node method that will fix this issue, allowing us to effectively perform a block decomposition where every principal block has size $\Theta(\sqrt{n})$. It is especially tempting to do so because it could reduce the time for meet and join

31:22 Space-Efficient Data Structures for Lattices

operations from $O(n^{3/4})$ to $O(\sqrt{n})$, as is claimed in [22]. The dummy node technique also seems like a reasonable approach to handling high-degree lattice nodes, which have often been an obstacle to the approaches we have considered.

There is good reason to expect that this is not possible, relying on some small assumptions. Suppose that there were a correct method of creating artificial principal blocks and that the method still works when we increase the block size from \sqrt{n} to $n^{2/3}$. That is, suppose that we can reliably decompose any lattice into $\Theta(n^{1/3})$ blocks of size $\Theta(n^{2/3})$ (and perhaps some $O(n^{1/3})$ smaller blocks). Note that the structure of the lattice has no impact on the ability to apply this method, thus we assume it applicable to all lattices.

There is the remaining issue of the residual block, however this is not a major difficulty. By adding a top element to the partial lattice (as in a complete lattice), we can treat the residual block in the same fashion as a principal block using the new top element as its block header.

Since the number of lattices on k elements is $2^{\Theta(k^{3/2})}$, it is possible to uniquely identify any such lattice using only $\Theta(k^{3/2})$ bits. Thus, each block of size $\Theta(n^{2/3})$ can be encoded in $\Theta(n)$ bits, and all of the blocks in the decomposition can be encoded in $\Theta(n^{4/3})$ bits. The order between any pair of elements in the same block can be tested, however inefficiently, using the encoding for that block. As well, since there are only $\Theta(n^{1/3})$ block headers, all of the meets between a lattice element and a block header can be stored in $\Theta(n^{4/3})$ space. In other words, we can simulate both (\mathcal{A}) and (\mathcal{B}) in only $O(n^{4/3})$ space.

This information is sufficient to perform order-testing between any pair of elements, and thus it uniquely determines the lattice. Lattices do not permit such a small representation; this would violate the $\Theta(n^{3/2})$ -bit lower bound. This strongly suggests that artificial blocks cannot be simulated without sacrificing the unique representative property, which is essential to the data structure.

Online Embedding of Metrics

Ilan Newman

Department of Computer Science, University of Haifa, Israel

<http://cs.haifa.ac.il/~ilan/>

ilan@cs.haifa.ac.il

Yuri Rabinovich

Department of Computer Science, University of Haifa, Israel

<http://cs.haifa.ac.il/~yuri/>

yuri@cs.haifa.ac.il

Abstract

We study deterministic online embeddings of metric spaces into normed spaces of various dimensions and into trees. We establish some upper and lower bounds on the distortion of such embedding, and pose some challenging open questions.

2012 ACM Subject Classification Networks → Network algorithms

Keywords and phrases Metric spaces, online embedding

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.32

Funding *Ilan Newman*: This Research was supported by The Israel Science Foundation, grant number 497/17.

1 Introduction

The modern theory of low-distortion embeddings of finite metrics spaces into various host spaces began to take shape with the appearance of the classical results of Johnson and Lindenstrauss [7]¹ and Bourgain [4]², in the last decades of the 20'th century. It was soon observed that this theory provides powerful tools for numerous theoretical and practical algorithmic problems. Nowadays, it is a mathematically deep and widely applicable developed theory, whose importance to algorithmic design is well recognized.

In this paper we study a relatively neglected aspect of metric embeddings, the *online* embeddings. In this setting, the vertices of the input finite metric space (V, d) are exposed one by one, together with their distances to the previously exposed vertices. Each newly exposed vertex v is mapped to the host space $(\mathcal{H}, d_{\mathcal{H}})$ before the next vertex is exposed, and without altering the embedding of previously exposed vertices. The quality of the resulting embedding $\phi : V \rightarrow \mathcal{H}$ is measured by its *expansion* and *contraction*:

$$\text{expansion : } \max_{v,u \in V} \frac{d_{\mathcal{H}}(\phi(v), \phi(u))}{d(v, u)} \qquad \text{contraction : } \max_{v,u \in V} \frac{d(v, u)}{d_{\mathcal{H}}(\phi(v), \phi(u))}$$

The product of the two is called the (multiplicative) *distortion* of ϕ . The distortion $\text{dist}(d \hookrightarrow d_{\mathcal{H}})$ of embedding (V, d) into $(\mathcal{H}, d_{\mathcal{H}})$ is the minimum possible distortion of any such mapping ϕ . Since usually (and in this paper in particular) the host space is scalable, $\text{dist}(d \hookrightarrow d_{\mathcal{H}})$ can be alternatively defined in the offline setting as the minimum possible expansion over non-contracting mappings, or the minimum possible contraction over non-expanding mappings.

¹ Any n -point Euclidean metric can be efficiently embedded into $\ell_2^{\frac{\log n}{\epsilon^2}}$ with $(1 + \epsilon)$ -distortion.

² Any n -point metric can be efficiently embedded into the Euclidean space of dimension $O(\log n)$.



In the online setting the above three notions may not, and probably do not, coincide. This is so since the scaling of ϕ cannot be performed in the end, after having finished constructing the entire mapping. For the same reason, in the online setting, the *maximum* between the distortion and the contraction may be a more suitable measure of similarity than the multiplicative distortion. Also, the knowledge of $n = |V|$ in advance matters, and may potentially be of help.

In addition to deterministic online embedding algorithms, it is natural to consider *probabilistic* online embeddings against non-adaptive adversary. In this setting, instead of considering the distortion between d and $d_{\mathcal{H}}$ for a fixed embedding, we consider the expected distortion between d and a random embedding that is selected from some pre-designed distribution.

In this paper we focus only on *deterministic embeddings* into the standard normed spaces $\ell_2, \ell_1, \ell_\infty$ of various dimensions, and into trees. Our results clarify what can be achieved in dimension 1, and in dimension exponential in n . What happens in between is a challenging open problem. We also present a lower bound on online embedding of a size- n metrics into ℓ_2 of unbounded dimension.

It is our hope that the findings of the present paper may provide a good starting point for further studies of deterministic online embeddings.

1.1 Previous Work

To the best of our knowledge, the first result about online embedding appeared implicitly in a paper of the authors [8]. The authors show a $(\sqrt{\log n})$ lower bound on the distortion of an *offline* embedding of a shortest path metric of a certain family of series-parallel graphs $\{D_n\}$ into ℓ_2 . Without ever mentioning the term “online”, the proof, in fact, establishes a lower bound of \sqrt{n} on the distortion of an online embedding of the shortest path metric of a certain family of graphs G_{2n} on $2n$ vertices that are subgraphs of D_n . Although [8] received due attention, and its online implications were noticed e.g. by the authors of [6]³, the explicit statement (which was and still remains state of the art in its context) has never appeared in print, and went largely unnoticed. Here we amend this situation (see Section 2).

Another related result appeared in [1] (Th. 3.1) in a rather unrelated context. It claims the following. Let (V, d) an arbitrary metric space with $|V| = n$. Assume that V is exposed in a random uniform order. Then the greedy online algorithm that attaches each new point v to the closest one among the points exposed so far, say u , by an edge of length $d(v, u)$, produces a random dominating tree T so that $E[d_T]$ expands d by $O(n^2)$.

If the order is fixed, a similar but simpler analysis implies that d_T expands d by at most $O(2^n)$. Since this turns out to be rather tight (up to the basis of the exponent) for a deterministic embedding into a tree, we shall discuss it in more details in Section 3. Another somewhat related notion, that we do not discuss here, is that of terminal embedding and using extension techniques [5].

The first (and, to our knowledge, the only) published paper explicitly dedicated to online embeddings is [6]. Observing that a large part of the offline embedding procedure from [2] can be implemented online, the authors in [6] establish quite strong results for probabilistic online embeddings. Most of these results depend on the so called *aspect ratio* Δ of the input metric d – that is, the ratio between the largest and the smallest distance in it. The main results of [6] are as follows (it is assumed that $|V| = n$):

³ It served as partial motivation for their paper. [Private communication]

1. A metric space (V, d) can be probabilistically online embedded into $\ell_p^{\log n \cdot (\log \Delta)^{1/p}}$ with distortion $O(\log n \cdot \log \Delta)$ for any $p \in [1, \infty]$.
For $p = \infty$, (V, d) can also be embedded in $\ell_\infty^{\log^{O(1)} n}$ with distortion $O(\log n \cdot \sqrt{\log \Delta})$.
On the negative side, (V, d) cannot be online embedded into ℓ_2^D with distortion better than $\Omega(n^{1/D-1})$ even when d is $(1 + \epsilon)$ -close to a submetric of ℓ_2^D .
2. A metric space (V, d) can be probabilistically online embedded into a distribution of a non-contracting ultrametrics (and subsequently tree-metrics) with distortion $O(\log n \cdot \log \Delta)$.
On the negative side, (V, d) cannot be probabilistically online embedded into a distribution of a non-contracting ultrametrics with distortion better than $\min\{n, \log \Delta\}$.

A very recent result [3] also discuss probabilistic online embeddings into trees, in the context of terminal-embedding. In particular they also obtain lower bounds on probabilistic embeddings into trees that are parameterized by the aspect ratio.

1.2 Our Results

We are interested in *deterministic* online embeddings into normed spaces, and in particular in the interplay between the distortion and the dimension of the host space. Unlike [6], we seek bounds independent of the aspect ratio.

1. **Embedding Into ℓ_2 :** There exists a family of metrics $\{d_{2n}\}$ such that each d_{2n} (a metric on $2n$ points) requires distortion \sqrt{n} in any deterministic online embedding into ℓ_2 of *any dimension*. The metrics $\{d_{2n}\}$ are the shortest-path metrics of a family $\{G_{2n}\}$ of weighted series-parallel graphs. These metrics are quite simple; e.g., they embed into the line with a constant universally bounded distortion.

By John's Theorem from the theory of finite-dimensional normed spaces, this implies an $\sqrt{n/D}$ lower bound on online embedding of d_{2n} into any normed space of dimension D . Comparing to the corresponding lower bounds of [6] (and ignoring the restrictions on d), we conclude that their result is stronger for $D = 2, 3$, and incompatible or weaker for other dimensions.

Our only positive results for online embedding into ℓ_2 follow from the embedding into the line.

2. **Embedding Into the line, and into trees.** As mentioned above, a simple greedy online embedding algorithms results in a dominating tree whose metric distorts the input metric d_n , on n points, by at most $O(2^n)$. Using a more complicated argument, we show that d can be online embedded into the *line* with distortion $O(n \cdot 6^n)$.

We also establish a lower bound of $\Omega(2^{n/2})$ for online embedding metrics on n points into trees. The “hard” metrics used in the proof are in fact submetrics of a (continuous) cycle, and they embed (offline) in the line with a constant universally bounded distortion.

3. **Distortion and dimension.** What is the smallest dimension D such that d can be embedded into ℓ_∞^D with distortion at most $1 + \epsilon$?⁴ Our first, rather surprising result, is that even a metric d on 4 points requires $D = \Omega(\log \frac{1}{\epsilon})$ in this setting.

On the positive side, we (efficiently) prove that $D = (\frac{4n}{\epsilon})^n$ suffices.

4. **Isometric online embeddings.** We show that size- n tree metrics d (i.e., arbitrary submetrics of the shortest-path metrics of weighted trees) isometrically online embed into ℓ_1^{n-1} . This implies that such d isometrically online embeds into $\ell_\infty^{2^{n-2}}$ for $n > 1$. One conclusion is that if d probabilistically online embeds with expansion a into a distribution of tree metrics supported on at most k trees, then d embeds with the same expansion into $\ell_1^{k(n-1)}$ and $\ell_\infty^{2^{k(n-1)-1}}$.

⁴ It is well known that any metric d of size n can be isometrically embedded into ℓ_∞^{n-1} . Thus, unlike any other ℓ_p , ℓ_∞ is universal in the sense that any metrics is isometric to its submetric.

Open questions:

- Can every n -points metric be online embedded into ℓ_2 (of some dimension) with $\text{poly}(n)$ small distortion?
- Can every n -points metric be online embedded into ℓ_∞^D , where D is at most polynomial in n , and with $\text{poly}(n)$ distortion. In particular this is open for $D = 2$, with a polynomially small distortion?
- At what rate does the quality of the best online embedding (e.g., into ℓ_∞^D) improve when D grows?

2 A lower bound for embeddings into ℓ_2

As mentioned in the previous section, the following theorem is implied by the proof of the main result of [8]:

► **Theorem 1.** *There is a family of metrics $\{d_{2n}\}$ on $2n$ points for any natural n , that requires expansion $\geq \sqrt{n}$ in any non-contracting online embedding into ℓ_2 of any dimension (including infinite dimension).*

Given an online non-contracting embedding algorithm A , the “hard” $\{d_{2n}\}$ is constructed as follows. It will be the shortest-path metric of the following weighted graph G_{2n} . G_2 is simply unit-weighted K_2 . The graph G_{2n+2} is obtained by choosing an edge $e = (v, u)$ of weight 2^{2-n} in G_{2n} , and replacing it by a 4-cycle $v-x-u-y-v$ with edges of weight 2^{3-n} . It remains to specify the edge e . It is proven in [8], inductively, that one of weight 2^{2-n} edges in G_{2n} is expanded by A , by at least \sqrt{n} . Further, this implies that of the four new edges (v, x) , (x, u) , (u, y) and (y, u) , at least one edge must be expanded by A by at least \sqrt{n} . This is the new edge to be chosen by the adversary.

As mentioned above, the metrics d_{2n} are very simple. E.g., it is an easy matter to verify that each d_{2n} (offline) embeds into the line with distortion ≤ 3 , and isometrically embeds into ℓ_1 .

Currently, we do not know how tight is the above bound, and whether is it at all possible to obtain a polynomially small in n (online) distortion for online embedding into ℓ_2 . We do know that it is possible for tree metrics (in view of Theorem 15), and that in general it is at most exponential (by Theorem 10).

3 Online embedding into trees

In tree embeddings we refer to online embeddings that constructs a tree whose vertices may contain *Steiner points*. That is, the constructed tree, besides the points corresponding to the input metric, may contain additional points. At each step, once a new vertex is exposed, the embedding algorithm picks an existing edge of the tree, subdivide it (without changing its total weight) by creating a new Steiner point, and attaches to it the new vertex by a new edge of a corresponding weight. The new edge is always a leaf, except when the weight is 0.

► **Theorem 2.** *Any metric on n points can be deterministically online embedded into a tree with distortion $\leq 2^{n-1} - 1$, even without using Steiner points.*

Proof. Just connect the new point v to the previously exposed point u that is the closest to v in the metric d , by an edge of weight $d(v, u)$.

The analysis is essentially the same as in [1]. Let \tilde{d} denote the tree metric approximating d . Clearly, \tilde{d} is not-contracting. Let α_k denote the its expansion after k steps. Then, $\alpha_2 = 1$, and $\alpha_{k+1} \leq 2\alpha_k + 1$. Indeed, let x be the new point, and assume it was connected to y . Then, for any previously exposed vertex a ,

$$\begin{aligned} \tilde{d}(a, x) &= \tilde{d}(a, y) + d(x, y) \leq \alpha_k \cdot d(a, y) + d(x, y) \leq \alpha_k \cdot (d(a, x) + d(x, y)) + d(x, y) \\ &\leq (2\alpha_k + 1) \cdot d(a, x) \end{aligned}$$

where the penultimate inequality is by the triangle inequality, and the last one follows from the choice of y .

The recursive formula $\alpha_{k+1} \leq 2\alpha_k + 1$ implies that $\alpha_n \leq 2^{n-1} - 1$. ◀

In view of the above theorem, this is rather tight:

► **Theorem 3.** *There is a class of metrics on n points for which any online embedding algorithm for that class into a tree metric, results in a distortion of at least $2^{(n-4)/2}$. Furthermore, every metric in the class is (offline) embeddable into a line with constant distortion.*

Proof. The metric that will be exposed is a finite submetric of the continuous unit cycle C . Let d_C be the shortest path metric induced on C . We will show that for every $k \geq 1$, the tree that is constructed on the first $4 + 2k$ points distorts d_C on the induced $4 + 2k$ points by no less than 2^k .

Working with the infinite metric space C (instead of a finite metric space), simplifies notions. One may consider the case in which n , the number of points in the metric space that is going to be exposed is given to the algorithm at the beginning. Even then, the following proof works. Moreover, we may restrict ourselves to the finite submetric space of C induced by 2^n points that are uniformly placed on C .

We start with the following simple facts. For two points x, y $P_T(x, y)$ will always refer to the path between x and y in the tree T that would be relevant to the context.

▷ **Claim 4.** Let u_1, u_2, u_3, u_4 be four vertices in a tree $T = (Y, E)$. Let $P(u_i, u_j)$, $1 \leq i < j \leq 4$ be the path in the tree from u_i to u_j . Then either $P(u_1, u_2) \cap P(u_3, u_4) \neq \emptyset$ or $P(u_2, u_3) \cap P(u_1, u_4) \neq \emptyset$.

The lower bound on the distortion will follow from the following claim.

▷ **Claim 5.** Let p, q, r, s be points in a metric space such that $d(p, q), d(r, s) \leq \alpha$ while $d(p, r), d(p, s), d(q, r), d(q, s) \geq \beta$. Assume also that p, q, r, s are embedded into a weighted tree, T , such that $P_T(p, q) \cap P_T(r, s) \neq \emptyset$. Then the tree distance d_T distort d by at least β/α .

Proof. Assume that the expansion is $\gamma \geq 1$. Then in the tree, $d_T(p, q), d_T(r, s) \leq \gamma \cdot \alpha$. In particular it follows that,

$$\sum_{e \in P_1} w(e) + \sum_{e \in P_2} w(e) = d_T(p, q) + d_T(r, s) \leq 2\gamma \cdot \alpha \tag{1}$$

However, as the paths $P_1 = P_T(p, q)$ and $P_2 = P_T(r, s)$ intersect, it follows that their union include the paths $\mathcal{P} = \{P_T(p, s), P_T(s, q), P_T(q, r), P_T(r, p)\}$. More over, every edge in $P_1 \cup P_2$ appears in exactly two of the paths from \mathcal{P} . Hence we conclude that,

$$\sum_{e \in P_T(p,s)} w(e) + \sum_{e \in P_T(s,q)} w(e) + \sum_{e \in P_T(q,r)} w(e) + \sum_{e \in P_T(r,p)} w(e) \leq 2 \sum_{e \in P_1} w(e) + 2 \sum_{e \in P_2} w(e) \tag{2}$$

Where the last inequality may be strict as the edges in the intersection of P_1 and P_2 contribute four times to the right hand side.

Combining Equations (1) and (2), we conclude that for at least one path $P \in \mathcal{P}$, the length of P is at least $\gamma \cdot \alpha$. Assume w.l.o.g that $P = P(p, s)$, then the contraction is at least $\nu = \frac{d(p,s)}{\gamma \cdot \alpha} \geq \frac{\beta}{\gamma \cdot \alpha}$ which implies that the distortion is at least $\nu \cdot \gamma \geq \beta/\alpha$. \triangleleft

We return to the proof of the theorem. Let $d = d_C$ the metric induced by C . For two subsets $S, T \subseteq C$ let $d(S, T) = \min_{s \in S, t \in T} d(s, t)$ (in our applications this minimum will always exist). We fix one point in the cycle $v(0) \in C$ as a reference point, this then defines every point by its distance along the cycle going clockwise. Thus we denote by $v(\alpha)$, $0 \leq \alpha < 1$ the point of length α from $v(0)$ when going along the cycle. For two points $x = v(\alpha), y = v(\beta)$ let $C[x : y]$ be the segment of the cycle on the shortest path between x and y . The mid point of $C[x : y]$ is the point on the geodesic path between x, y which is of equal distance from x and y (this is not well defined only if $d(x, y) = 1/2$, but we will never use this definition in this case).

Fix an online embedding algorithm for n points from C into a tree T , and denote the resulted metric is d_T . The adversary first exposes $u_1 = v(0), u_2 = v(\frac{1}{4}), u_3 = v(\frac{1}{2}), u_4 = v(\frac{3}{4})$. Let T_1 be the tree constructed by the algorithm just after this point. Using fact 4 (with that order on the points) we may assume w.l.o.g that $P(u_1, u_2) \cap P(u_3, u_4) \neq \emptyset$. Note that $d(C[u_1 : u_2], C[u_3 : u_4]) \geq 1/4$.

The adversary will work in phases, each time exposing 2 points. The initial phase (numbered as $k = 0$ exposing 4 points) results in T_0 above. Let $T = T_k$ be the tree that is constructed by the algorithm at steps $k = 1, \dots$ after exposing $4 + 2k$ points. The adversary will always hold two pairs of points that are already exposed $x, y \in C[u_1 : u_2], x', y' \in C[u_3 : u_4]$, maintaining the invariant that: all points in $C[x : y] \setminus \{x, y\}$ and in $C[x' : y'] \setminus \{x', y'\}$ are not exposed, and $P_T(x, y) \cap P_T(x', y') \neq \emptyset$. It will also be the case that $d_C(C[x : y], C[x' : y']) \geq 1/4$, while $d_C(x, y) = d_C(x', y') = 2^{-k-2}$.

For $k = 0$ the points $x = u_1, y = u_2, x' = u_3, y' = u_4$ already comply with the invariants above. Assume that after phase k we already have exposed $4 + 2k$ and the adversary holds x, y, x', y' as required. Then at phase $k+1$ the adversary exposes two new vertices: z that is the mid point in $C[x : y]$ and z_1 that is the mid point of $C[x', y']$. Since $P_T(x, y) \cap P_T(x', y') \neq \emptyset$ then at least one of $P_T(x, z), P_T(z, y)$ intersects $P_T(x', y')$. We replace y with z if $P_T(x, z)$ intersects $P_T(x', y')$, otherwise, we replace x with z . Similarly, we replace either x' or y' with z_1 , so that the resulting two paths still intersect. It is easy to see that the distances are as claimed.

Finally, by Claim 5, applied on x, y, x', y' (in this order) at the end of any phase k , we conclude that the tree distance d_T distort d_C on the four points by at least 2^k .

We end this proof by noting that the actual metric that is exposed is (offline) embeddable into a tree and even into a line with a constant distortion. This can be done by e.g., 'cutting' C at the point $v(7/8)$ and embedding each point x at $v(x)$ in the resulting interval. \blacktriangleleft

4 Embedding into the line

Theorem 3 implies that online embedding of general metrics into the line results in a distortion that in the worst case is at least exponential in the number of points. This is true even for online embedding of tree metrics into the line (using a similar argument as in the proof of Theorem 3). Here we show that any metric (V, d) on n points can be online embedded into the line with distortion that is most exponential (in the number of points exposed so far). We don't assume here that n , the number of points or any upper bound on this number is given in advance.

► **Theorem 6.** *Let (V, d) be a metric, then V can be online embedded into the line (without a priori knowing $n = |V|$), with distortion bounded by $O(n6^n)$.*

Proof. For every point x that is already embedded let $\phi(x)$ be its embedding.

Assume at stage i that x_i is exposed and let z be the closet point to x_i from the previously exposed points, with $d(x_i, z) = d$. Let I be the left most interval of length $3^{-i}d$ that is right of z and is empty of any previously exposed point. We then place x_i in the mid point of I . Note that since there are only $i - 1$ previously exposed points (including z), then there must be such empty interval at distance at most $(i - 2) \cdot 3^{-i}d$.

In the following we call z the *father* of x_i in this embedding and denote it as $father(x_i)$. We are going to bound separately the expansion and the contraction.

Let $\gamma(k)$ denote the bound on the expansion after the k th point is embedded. Bounding $\gamma(k)$ is by a similar argument to the tree embedding. Let x_k be the last point that is embedded, let $z = father(x_k)$ and let y be any previously exposed point. The triangle inequality asserts that,

$$d(y, z) \leq d(y, x_k) + d(x_k, z) \leq 2d(x_k, y) \quad (3)$$

Thus, by the definition of the embedding of x_k and the induction hypothesis,

$$\begin{aligned} |\phi(x_k) - \phi(y)| &\leq |\phi(x_k) - \phi(z)| + |\phi(z) - \phi(y)| \leq \\ &+(k - 1.5) \cdot 3^{-k}d(x_k, z) + \gamma(k - 1)d(z, y) \leq k3^{-k} \cdot d(x_k, y) + \gamma(k - 1)d(z, y) \end{aligned} \quad (4)$$

where the last inequality is by the fact that $d(x_k, z) \leq d(x_k, y)$.

Using equation (3) we get,

$$|\phi(x_k) - \phi(y)| \leq (k3^{-k} + 2\gamma(k - 1)) \cdot d(x_k, y). \quad (5)$$

We get the following recursion on $\gamma(k)$: $\gamma(k) \leq k3^{-k} + 2\gamma(k - 1)$ which implies that $\gamma(k) \leq 3 \cdot 2^k$.

To bound the contraction let a, b be any two exposed points. By the embedding algorithm there are two sequences $z = y_1, y_2, \dots, y_k = a$ and $z = w_1, w_2, \dots, w_\ell = b$ where $y_i = father(y_{i+1})$, $w_i = father(w_{i+1})$ and $\{y_i\}_1^k, \{w_i\}_1^\ell$ are disjoint. A marginal case is when $z = a$ and one of the sequences is empty. The argument for the marginal case will be presented at the end of the proof (after the proof of Claim 8).

Let $\delta_i = d(y_{i-1}, y_i), i = 2, \dots, k$ and $\nu_i = d(w_{i-1}, w_i), i = 2, \dots, \ell$. For any point x let $order(x) = \ell$ if $x = x_\ell$ namely, x is the ℓ exposed point (do not confuse the $order(x)$ with its location in the sequences of y_i 's or w_i 's).

Let $D = \max\{d(x, father(x))\}$ where x ranges over all points except z in the two sequences above, and assume w.l.o.g. that the last exposed point among the two sequences, x_j , for which $d(x_j, father(x_j)) = D$ is y_i (namely, that the maximum is achieved in the sequence that corresponds to a). Let $s = order(y_i)$.

By our algorithm $y_i = x_s$ is embedded in the middle of an empty interval I of size $3^{-s}D$.

We use the following claims.

For $j = 1, \dots, k - i$ let $r_j = order(y_{i+j}) - s$.

► **Claim 7.** For any $j \geq 1$, y_{i+j} is embedded inside I and $0 \leq \phi(y_{i+s}) - \phi(y_i) \leq \frac{|I|}{2} \cdot (1 - 2^{-r_j})$.

Proof. We first describe the situation for the case $j = 1$. The case for larger j is similar. Let $r_1 = order(y_{i+1}) = r$. Namely, $r - 1$ points $x_{s+1}, \dots, x_{s+r-1}$ are exposed after y_i and before y_{i+1} .

Recall that at time i when y_i is embedded, I is empty, and y_i is placed in the middle of I splitting I into two empty intervals I_L, I_R of size $|I|/2$ each. According to the algorithm y_{i+1} needs to be embedded in the middle of an interval of size $\alpha = 3^{-(r+s)} \cdot \delta_{i+1} \leq 3^{-r} \cdot 3^{-s} \cdot D \leq 3^{-r} \cdot |I|$ that is empty at time $s+r$. If $r=1$, namely if y_{i+1} is exposed right after $y_i = x_s$, then obviously there is a α -size empty interval right of $\phi(y_i)$, as I_R is empty at this point and $|I|/2 > \alpha$. Generally, for $r > 1$, some of the $r-1$ points that are exposed between y_i and y_{i+1} may occupy parts of I forcing y_{i+1} to be embedded further to the right.

Each time a point $x \neq y_{i+1}$ is placed in I_R it must be in the middle of an empty interval splitting the right empty interval of I_R into two empty subintervals, hence leaving an empty interval of at least half the size at the right of I_R . Hence after placing at most $r-1$ such points, there will still be an empty interval of size $|I_R|/2^{r-1} > \alpha$. Hence there is a suitable empty interval for y_{i+1} in I_R and it follows that $\phi(y_{i+1}) - \phi(y_i) \leq |I|/2 - 2^{-r+1}|I_R| + \alpha/2 \leq \frac{|I|}{2} \cdot (1 - 2^{-r})$.

In the general case for y_{i+j} the argument is identical except that $r_j - 1$ points might have been embedded into I_R before y_{i+j} . \triangleleft

\triangleright **Claim 8.** $|\phi(a) - \phi(b)| \geq 2^{-(r_k+1)}|I|$.

Proof. Claim 7 asserts that $a = y_k$ is embedded inside I to the right of y_i and $\phi(a) - \phi(y_i) \leq \frac{|I|}{2} \cdot (1 - 2^{-r_k})$.

We now consider the place where b is embedded. Let t be the largest so that w_t is exposed before y_i . Again, since I is empty when y_i is exposed, w_t must be embedded to the right or to the left of I . If w_t is embedded to the right of I then b , that is embedded right of w_t , is right of I and hence $\phi(b) - \phi(a) \geq \frac{|I|}{2} \cdot 2^{-r_k}$ implying the claim.

On the other hand, if w_t is embedded to the left of I , then by a similar calculation that is done in Claim 7, all points w_{t+j} are embedded at most at distance $\frac{|I|}{2}$ from w_t . (Since they all are exposed after y_i and in particular have $\nu_{t+j} < D$ and $\text{order}(w_{t+j}) > s$). Since $\phi(y_i) - \phi(w_t) \geq |I|/2$ by the assumption that w_t is left of I , we conclude that $\phi(a) - \phi(b) \geq \phi(a) - \phi(y_i) + \phi(y_i) - \phi(w_t) \geq \frac{|I|}{2} \cdot 2^{-r_k}$ in this case too.

This completes the proof of Claim 8. \triangleleft

Now with this lower bound on $|\phi(a) - \phi(b)|$, to bound the contraction it is enough to upper bound $d(a, b)$. Indeed,

$$d(a, b) \leq \sum_1^k d(y_i, y_{i-1}) + \sum_1^\ell d(w_i, w_{i-1}) \leq n \cdot D \implies$$

$$\frac{d(a, b)}{|\phi(a) - \phi(b)|} \leq \frac{nD}{2^{-(r_k+1)}|I|} \leq n2^{r_k+1} \cdot 3^s \leq n3^n$$

To complete the proof, consider the case where one of the sequences is empty. Namely w.l.o.g $z = b$. If b is exposed before y_i then we are at the same situation as in Claim 8, implying the same lower bound on $\phi(y_i) - \phi(b)$. If $z = y_i$ then Claim 7 asserts that $a = y_k$ is embedded in I , and $\phi(a) - \phi(b) \geq \sum_{j=2}^k (\phi(y_j) - \phi(y_{j-1})) \geq \sum_2^k \delta_j \cdot 3^{-r_j}/2$, where the inequality is by the fact that for every j , y_j is embedded to the right of y_{j-1} at distance at least $3^{-r_j} \delta_j/2$. But this last expression is at least $\frac{3^{-n}}{2} \sum_2^k \delta_j \geq 3^{-n} d(a, b)/2$ proving that in this case the contraction is bounded by $2 \cdot 3^n$ as well.

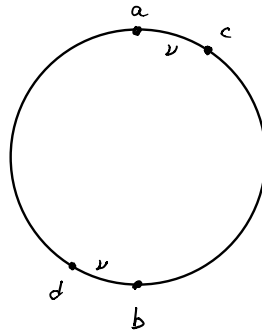
This completes the proof of the Theorem. \blacktriangleleft

5 Online embedding into ℓ_∞ with $(1 + \epsilon)$ distortion

It is well known that any metric on n points can be (offline) embedded isometrically online into ℓ_∞^{n-1} . It therefore comes as a surprise that even the metrics on 4 points cannot be online isometrically embedding into ℓ_∞ of any finite dimension. This will be proven using a special class of 4-points metrics that are submetrics of a continuous cycle.

► **Theorem 9.** *There exists μ on four points for which any online embedding into ℓ_∞^D incurs a distortion $(1 + \Omega(1/4^{2D}))$. Consequently, to ensure a distortion $(1 + \epsilon)$, one needs $\Omega(\log(1/\epsilon))$ dimensions.*

Proof. The metrics under discussion look as depicted in Figure 1. They are all defined by 4 points on the cycle whose circumference is of size 2. All these metrics contain two antipodal points a, b as in Figure 1, that are exposed first and with $d(a, b) = 1$. After a, b are embedded the next two points c, d are exposed. c, d are also antipodal and are defined by the distance $d(a, c) = \nu$.



■ **Figure 1** The metric μ_4 . a, b are exposed first and then ν is set (defining c, d).

We view ϕ as D online non-contacting embeddings into the line $\Phi = \{\psi_1, \dots, \psi_d\}$, where $\psi_i : \{a, b, c, d\} \rightarrow \mathbb{R}, i = 1, \dots, D$. The adversary reveals first the antipodal points a, b . It will then choose ν appropriately, and reveal the corresponding antipodal points c, d .

Let $\delta = 4^{-(D+1)}$. Assume that a, b are exposed and w.l.o.g., $0 = \psi(a) \leq \psi(b)$ for every $\psi \in \Phi$. Moreover, we may assume that every ψ_i is not expanding by more than $1 + \delta$, as otherwise we are done. Hence by multiplying by $\frac{1}{1+\delta}$ we may assume that every $\psi \in \Phi$ is non-expanding.

We partition the interval $[0, 1]$ into $d + 1$ sets $B_0 = (1 - 4^{-D}, 1]$, $B_i = (1 - 4^{i-D}, 1 - 4^{i-D-1}]$, $i = 1, \dots, D - 1$, and finally $B_D = [0, 3/4]$.

After exposing a, b , $\psi(b)$ is determined for every $\psi \in \Phi$. This partitions Φ into $D + 1$ classes $\tilde{B}_0, \dots, \tilde{B}_d$ by letting $\psi \in \tilde{B}_i$ if $\psi(b) \in B_i$. Hence for some $i \in [D + 1]$, $\tilde{B}_i = \emptyset$. Fix such an i , and set $\nu = 4^{i-D}/3$, which define c and d .

Consider first the case $j > i$, then for $\psi \in \tilde{B}_j$, $\psi(b) < 1 - 4^{i-D} \leq 1 - 3\nu$.

Since ψ is non-expanding, $\mu(a, c) = \nu$ implies that $\psi(c) \in I_c = [-\nu, \nu]$. Similarly, $\psi(d) \in I_d = [1 - \psi(b) - \nu, 1 - \psi(b) + \nu]$. But $\max\{|y - x|, y \in I_d, x \in I_c\} \leq 1 - \psi(b) + 2\nu$. Hence the contraction of $\mu(c, d)$ in this case is at least $\frac{1}{1 - \psi(b) + 2\nu} \geq \frac{1}{1 - 4^{i-D-1}}$.

On the other hand, for $\psi \in \tilde{B}_j$ and $j < i$, $\psi(b) \geq 1 - 4^{i-1-D}$. But then since ψ is non-expanding, it follows that $\psi(d) \leq 1 - \nu$ (on account of $\mu(a, d)$), and $\psi(c) \geq \psi(b) - (1 - \nu) = \nu - (1 - \psi(b))$. It follows that the $\psi(d) - \psi(c) \leq 1 - 2\nu + (1 - \psi(b)) \leq 1 - \frac{2 \cdot 4^{i-D}}{3} + 4^{i-1-D} \leq 1 - 4^{i-D-1}$. Hence each such ψ contracts $\mu(c, d)$ by at least $\frac{1}{1 - 4^{i-D-1}}$.

32:10 Online Embedding of Metrics

We conclude that for the above setting of ν , every ψ contracts $\mu(c, d)$ by at least $\frac{1}{1-4^{-(D+1)}}$. Recall that we have started the proof by multiplying Φ by $\frac{1}{1+\delta}$. Hence the distortion of Φ is at least $\frac{1}{(1+\delta)(1-4^{-(D+1)})} \geq \frac{1}{1-4^{-2(D+1)}}$. \blacktriangleleft

Let us note, without providing more details here, that for metrics μ as above the about lower bound is tight and cannot be strengthened. We conjecture that every metric on 4 points can be online embedded in ℓ_∞^D with distortion $1 + \exp(-D)$.

Next we show a result complementary to Theorem 9. That is – any metric can be embedded into ℓ_∞ with distortion arbitrary close to 1, using large enough dimension.

► Theorem 10. *Let (V, d) be any metric space, $\epsilon > 0$ an arbitrary small constant, and let $n = |V|$ (or an upper bound on $|V|$) be known in advance. Then using $(\frac{2n}{\ln(1+\epsilon)})^{2n}$ coordinates one can embed d online in a 1-Lipschitz embedding with contraction bounded by $(1 + \epsilon)$.*

Proof. The proof idea is to approximate the universal embedding of d into ℓ_∞^∞ .

Let $V' \subseteq V$ be points from V and $z \in V'$ one fixed point. Let ϕ_z^U (U here stands for *universal*) be the following embedding of V' into the line. $\phi_z^U(z) = 0$ and for every $x \in V'$, $\phi_z^U(x) = d(z, x)$.

The following claim is standard and immediate from the definition.

► **Claim 11.** ϕ_z^U is a 1-Lipschitz embedding of d and for every $x \in V'$ it does not distort $d(z, x)$.

It follows then that if for every $z \in V$ there is a coordinate on which ϕ_z^U is realized, then the embedding is an isometry of d in ℓ_∞ . Hence, ℓ_∞ of dimension $n - 1$ is universal for any metric space on n points.

Here we will online approximate each of these coordinates by preparing in advance a coordinate (in fact a collection of coordinates) for each possible new z . Suppose that the points a_1, \dots, a_k are exposed (not necessarily in that order), and that for a new point z , there is a line (coordinate) in which a_1, \dots, a_k are embedded in increasing order that is consistent with the distance order to z . We show below that under some restrictions on the embedding of these first k points, there is an augmentation of this embedding to any possible consistent z .

Since there are only finite number of ordering of the first k points with respect to their distance from z , we will prepare in advance a line (in fact, a set of lines) for every possible ordering. This will allow us to embed every possible new coming z .

We start with the following Claim asserting that a consistent ordering on the line can be augmented to a new point, under some suitable restriction.

► **Claim 12.** Let $k < n$ and $\{a_1, \dots, a_k\} \subset V$ a set of arbitrary points. Let $\delta > 0$ be a small constant and $1 \leq \ell_i \leq \frac{n}{\delta}$, $i = 2, \dots, k$, a sequence of integers. Let $\tilde{\phi}$ a fixed 1-Lipschitz embedding of a_1, \dots, a_k on the line with $\tilde{\phi}(a_{i+1}) = \tilde{\phi}(a_i) + \frac{\ell_{i+1}-1}{n} \cdot \delta d(a_i, a_{i+1})$, $i = 1, \dots, k-1$. Then for any $z \in V \setminus \{a_1, \dots, a_k\}$ such that

1. $d(z, a_i) \leq d(z, a_{i+1})$, $i = 1, \dots, k-1$.
2. For every $i = 1, \dots, k-1$,

$$\frac{(\ell_{i+1} - 1)\delta}{n} \cdot d(a_{i+1}, a_i) < d(z, a_{i+1}) - d(z, a_i) \leq \frac{\ell_{i+1}\delta}{n} \cdot d(a_{i+1}, a_i)$$

The augmentation of $\tilde{\phi}$ with $\tilde{\phi}(z) = \tilde{\phi}(a_1) - d(z, a_1)$ is 1-Lipschitz, contracting the distances $d(z, a_i)$, $i = 1, \dots, k$ by at most $e^{2\delta}$.

We call such $\tilde{\phi}$ “additive shifted approximation” of ϕ_z^U .

Proof. We will show that for any possible z for which the premises of the Claim hold, the augmented embedding $\tilde{\phi}$ above is an approximation of ϕ_z^U .

Fix a_1, \dots, a_k and let $\tilde{\phi}$ be the augmented embedding for an arbitrary z for which the assumptions of the claim hold. Let $c_i = e^{2\delta i/n}$. We prove by induction on $i = 1, \dots, k$ that

$$c_i d(z, a_i) \leq \tilde{d}(z, a_i) = \tilde{\phi}(a_i) - \tilde{\phi}(z) \leq d(z, a_i). \quad (6)$$

Which will assert that the embedding is 1-Lipschitz and with contraction of $d(z, a_k)$ which is at most c_k .

Indeed for $i = 1$, $\tilde{d}(z, a_1) = d(z, a_1)$.

Assume that Equation (6) is already proved for i and let us prove it for $i + 1$.

By definition, $\tilde{d}(z, a_{i+1}) = \tilde{d}(z, a_i) + \frac{(\ell_{i+1}-1)\delta}{n} \cdot d(a_i, a_{i+1}) \leq d(z, a_i) + d(z, a_{i+1}) - d(z, a_i) \leq d(z, a_{i+1})$, where the inequality follows from the induction hypothesis that $\tilde{d}(z, a_i) \leq d(z, a_i)$ and the 2nd condition in the claim. This establish the fact that the mapping is non-expanding.

The contraction c_{i+1} is bounded by (again using condition 2 of the claim)

$$\frac{d(z, a_{i+1})}{\tilde{d}(z, a_{i+1})} \leq \frac{d(z, a_i) + \ell_{i+1} \cdot \frac{\delta}{n} \cdot d(a_{i+1}, a_i)}{\tilde{d}(z, a_i) + (\ell_{i+1} - 1) \cdot \frac{\delta}{n} \cdot d(a_{i+1}, a_i)} \quad (7)$$

Next we note that by assumption $d(z, a_{i+1}) - d(z, a_i) \leq \ell_{i+1} \cdot \frac{\delta}{n} \cdot d(a_i, a_{i+1})$, while by the triangle inequality $d(a_i, a_{i+1}) \leq d(z, a_{i+1}) + d(z, a_i)$. Combining these two conditions implies that,

$$d(z, a_i) \geq \frac{1}{2} \cdot (1 - \ell_{i+1} \cdot \frac{\delta}{n}) d(a_i, a_{i+1}) \quad (8)$$

Plugging this in Equation (7), using that by induction $\tilde{d}(z, a_i) \geq d(z, a_i)/c_i$, we get,

$$\begin{aligned} c_{i+1} &\leq c_i \cdot \frac{d(z, a_i) + \ell_{i+1} \cdot \frac{\delta}{n} \cdot d(a_i, a_{i+1})}{d(z, a_i) + c_i \cdot (\ell_{i+1} - 1) \frac{\delta}{n} \cdot d(a_i, a_{i+1})} \\ &\leq c_i \cdot \frac{d(z, a_i) + \ell_{i+1} \cdot \frac{\delta}{n} \cdot d(a_i, a_{i+1})}{d(z, a_i) + (\ell_{i+1} - 1) \frac{\delta}{n} \cdot d(a_i, a_{i+1})} \leq \\ &c_i \cdot \left(1 + \frac{\frac{\delta}{n} \cdot d(a_i, a_{i+1})}{d(z, a_i) + (\ell_{i+1} - 1) \frac{\delta}{n} \cdot d(a_i, a_{i+1})}\right) \end{aligned}$$

Using again equation (8) for $d(z, a_i)$ in the denominator we get,

$$c_{i+1} \leq c_i \cdot \left(1 + \frac{\frac{2\delta}{n}}{1 + (\ell_{i+1} - 2) \frac{\delta}{n}}\right)$$

The last expression is the largest when $\ell_{i+1} = 1$ for which we get $c_{i+1} \leq c_i \cdot (1 + 2\frac{\delta}{n})$ and the claim follows. \triangleleft

To complete the proof, we will show that when exposing the $k + 1$ point z , having already embedded the first k points in a 1-Lipschitz embedding, there is a coordinate for which the points are placed along the line according to non-decreasing order of the distances from z , and with pairwise distances $\tilde{\phi}(a_{i+1}) - \tilde{\phi}(a_i)$ that are approximated as in the second item of the Claim.

Indeed fix $\epsilon > 0$ and let $\delta < \frac{1}{2} \ln(1+\epsilon)$, namely, such that $e^{2\delta} < 1+\epsilon$. For every $i = 2, \dots, n$ (note that n needs to be known in advance), we will make sure that after exposing the k th point, for any permutation $\pi \in \mathcal{S}_k$, and any setting of $(s_2, \dots, s_k) \in \{0, 1, \dots, \lfloor \frac{n}{\delta} \rfloor\}^k$ we have

32:12 Online Embedding of Metrics

at least one coordinate, namely a line and a 1-Lipschitz embedding $\tilde{\psi}$ of a_i , $i = 1, \dots, k$ such that the points appear in order as specified by the permutation π , and in which $\psi(a_{\pi(i+1)}) - \psi(a_{\pi(i)}) = s_{i+1} \cdot \frac{\delta}{n} d(a_{\pi(i+1)}, a_{\pi(i)})$. Note, that we assume inductively that for all coordinates (that is, lines) these embedding are required to be 1-Lipschitz.

Having such situation, once z is exposed, assume that a_1, \dots, a_k is a re-enumeration of the points by their distance to z , and let $s_i, i = 2, \dots, k$ be such that $\frac{(\ell_{i+1}-1)\delta}{n} \cdot d(a_{i+1}, a_i) < d(z, a_{i+1}) - d(z, a_i) \leq \frac{\ell_{i+1}\delta}{n} \cdot d(a_{i+1}, a_i)$. By the triangle inequality, such sequence $s_i, i = 2, \dots, k$ exists, and hence by assumption there is a line and an embedding of a_1, \dots, a_k for which the conditions of Claim 12 hold with respect to z . Then Claim 12 asserts that z can be placed in that line and the corresponding augmented embedding that is a “online” shifted additive approximation of ϕ_z^U is 1-Lipschitz, and with bounded contraction as needed.

For every other line we only need to place z so that it will remain 1-Lipschitz. Indeed since the embedding of a_1, \dots, a_k is 1-Lipschitz on the line, it is folklore that z can be placed too, so to result in a 1-Lipschitz embedding (e.g., by using the Helly property for the line).

One last thing to observe, is that in order to take care for future points, we also need to make sure that any relevant order of $\nu \in \mathcal{S}_{k+1}$, namely including z and any set of relevant integers s_2, \dots, s_{k+1} is also realized. To do this, for any possible such set of integers, and any permutation $\pi \in \mathcal{S}_k$, we prepare enough identical copies of the same embedding of $\{a_1, \dots, a_{k+1}\} \setminus \{z\}$ so to be able to place z in any of the corresponding $k+1$ intervals, and in any of the possible $\frac{n}{\delta}$ placers in the interval, so to cover all possible sequences s_2, \dots, s_{k+1} . Thus to estimate the required dimension, let $f(k)$ denote the number of lines needed for step k , in which we assume that every order and every sequence of numbers s_1, \dots, s_k is realized. By the previous discussion we need $f(k+1) = f(k) \cdot (k+1) \cdot \frac{n}{\delta}$ lines for step $k+1$. Since $f(1) = 1$, the recurrence implies that $f(n) = n! \cdot (\frac{n}{\delta})^{n-1}$. We conclude that the dimension needed for the online embedding above is at most $f(n) < (\frac{n}{\delta})^{2n}$ in order to embed any n -point metric. ◀

6 Isometric online embeddings

We conclude with a number of remarks on isometric embeddings. First, observe that the tree metrics d_T (like the Euclidean metrics) are essentially rigid, i.e., there exists an essentially unique (minimal) weighted tree T^* with Steiner points realizing d_T as its submetric. Moreover, this T^* can be constructed in an online manner, regardless of the order of exposure.

► **Theorem 13.** *Every tree metric d_T can be isometrically embedded into a metric of a weighted tree T^* (using Steiner points). The knowledge of n is not required.*

Skipping the details, the embedding at each step, given a new point x , introduces a new Steiner point y_x into the tree constructed so far, and attaches x to y_x by a new edge of weight w_x . Interestingly, the use of Steiner points is essential:

► **Lemma 14.** *There is a family of tree metrics that suffer an exponential distortion in every online embedding into a tree that does not use Steiner points, even when n is known in advance.*

The proof of Lemma 14 is based on the same ideas as in the proof of Theorem 3. We omit further details from this draft.

Next, we claim that tree metrics isometrically embed online into ℓ_1 .

► **Theorem 15.** *Every tree metric on n points is isometrically online embeddable into ℓ_1^{n-1} .*

We do not present a proof of this theorem here. The general idea is to follow the isometric embedding into a weighted tree T^* as outlined above. The invariant property of the embedding is that the adjacent points in the tree will differ by a simple coordinate. Thus, adding a new Steiner point will require no increase in dimension. Attaching a new point x to the corresponding Steiner point y_x involves taking the vector representing y_x , and adding to it a new coordinate with value w_x . The vectors constructed so far are assumed to have value 0 on this coordinate.

Our last remark is that if a metric d is online-embeddable into ℓ_1 of an a priori known dimension $D(n)$, then d can also be online embedded into ℓ_∞ of dimension $2^{D(n)-1}$. This is so, since the embeddings: $x = (x_1, \dots, x_D) \rightarrow (\langle x, \epsilon^1 \rangle, \dots, \langle x, \epsilon^{2^D} \rangle)$, where ϵ^i range over all possible choices of D -dimensional ± 1 vectors, is an isometry from ℓ_1^D into $\ell_\infty^{2^D}$, and, moreover, it is online constructible. 2^D can be improved to 2^{D-1} by fixing the first sign to be 1. Thus, e.g.,

► **Theorem 16.** *Every tree metric on n points is isometrically online embeddable into $\ell_\infty^{2^{n-2}}$.*

References

- 1 Sanjeev Arora, László Lovász, Ilan Newman, Yuval Rabani, Yuri Rabinovich, and Santosh S. Vempala. Local versus global properties of metric spaces. *SIAM J. Comput.*, 41(1):250–271, 2012. doi:10.1137/090780304.
- 2 Yair Bartal. On approximating arbitrary metrics by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998. doi:10.1145/276698.276725.
- 3 Yair Bartal, Nova Fandina, and Seeun William Umboh. Online probabilistic metric embedding: A general framework for bypassing inherent bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1538–1557, 2020. doi:10.1137/1.9781611975994.95.
- 4 Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel J. Math.*, 52:46–52, 1985. doi:10.1007/BF02776078.
- 5 Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theor. Comput. Sci.*, 697:1–36, 2017. doi:10.1016/j.tcs.2017.06.021.
- 6 Piotr Indyk, Avner Magen, Anastasios Sidiropoulos, and Anastasios Zouzias. Online embeddings. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2010. doi:10.1007/978-3-642-15369-3_19.
- 7 William B. Johnson and Joram Lindenstrauss. Extensions of 1-lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26:198–206, 1984.
- 8 Ilan Newman and Yuri Rabinovich. A lower bound on the distortion of embedding planar metrics into euclidean space. *Discrete Comput. Geom.*, 29:77–81, 2002. doi:10.1007/s00454-002-2813-5.

Primal-Dual 2-Approximation Algorithm for the Monotonic Multiple Depot Heterogeneous Traveling Salesman Problem

S. Rathinam

Texas A & M University, College Station, TX 77843, USA
srathinam@tamu.edu

R. Ravi

Carnegie Mellon University, Pittsburgh, PA 15213, USA
ravi@cmu.edu

J. Bae

Michigan Technological University, Houghton, MI 49931, USA
bae@mtu.edu

K. Sundar

Los Alamos Laboratory, NM 87545, USA
kaarthik@lanl.gov

Abstract

We study a Multiple Depot Heterogeneous Traveling Salesman Problem (MDHTSP) where the cost of the traveling between any two targets depends on the type of the vehicle. The travel costs are assumed to be symmetric, satisfy the triangle inequality, and are monotonic, *i.e.*, the travel costs between any two targets monotonically increases with the index of the vehicles. Exploiting the monotonic structure of the travel costs, we present a 2-approximation algorithm based on the primal-dual method.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Approximation Algorithm, Heterogeneous Traveling Salesman Problem, Primal-dual Method

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.33

Funding *R. Ravi*: This material is based upon work supported in part by the U. S. Office of Naval Research under award number N00014-18-1-2099.

1 Introduction

We consider a generalization of a multiple Traveling Salesman Problem (TSP) involving heterogeneous vehicles, where the cost of traveling between any two locations depends on the type of the vehicle. Given a set of targets, the initial location (depot) and metric travel costs corresponding to each vehicle, the objective is to find a tour for each vehicle such that each target is visited exactly once by some vehicle and the sum of the travel costs of all the vehicles is minimum. This problem is referred to as the Multiple Depot Heterogeneous Traveling Salesman Problem (MDHTSP) and is widely studied in the unmanned vehicle community [4, 5, 10, 11, 13–16, 20, 22].

MDHTSP is a generalization of the classic TSP and is NP-Hard. Therefore, we are interested in developing approximation algorithms for the MDHTSP. Henceforth, we assume the travel costs for each vehicle are symmetric and satisfy the triangle inequality unless otherwise mentioned. For covering all targets with multiple TSPs when all the vehicles are identical, there are several constant-factor approximation algorithms in the literature [6, 12, 17, 21]. Generally, most of these algorithms follow a three-step procedure: In the first



© S. Rathinam, R. Ravi, J. Bae, and K. Sundar;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 33; pp. 33:1–33:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

step, the tour requirements are relaxed to find an optimal constrained forest. In the second step, the edges in the constrained forest are doubled, or in special cases, a matching problem is solved (à la Christofides) for a subset of vertices and the corresponding edges are added to the forest to obtain an Eulerian graph for each vehicle. In the final step, the Eulerian graph for each vehicle is used to obtain an Eulerian walk and the repeated visits are shortcut to find a tour. Similar procedures are also used to find approximation algorithms for multiple Hamiltonian Path Problems in [3, 18].

No constant factor approximation algorithms are known for the MDHTSP. In [22], a $\frac{3n}{2}$ -approximation algorithm is presented for the MDHTSP where n denotes the number of vehicles (not the number of nodes in the metric). For a related variant of MDHTSP where all the vehicles start from the same depot and the objective is to minimize the makespan (maximum travel cost of any vehicle), a constant-factor approximation algorithm is presented in [8]. For the same makespan objective, when the vehicles are functionally heterogeneous¹, a $(2\lceil \ln(n) \rceil + 1)$ -approximation algorithm has been presented recently in [23].

Monotonic MDHTSP. We are interested in the special case of the MDHTSP in which some restriction is placed on the travel costs of the vehicles. Intuitively, we assume the vehicles are ordered and the costs are monotonic. Formally, let $D = \{d_1, d_2, \dots, d_n\}$ denote the n depots (initial locations) of the vehicles. Let T represent the set of targets. For each $i \in \{1, \dots, n\}$, let $V_i := T \cup \{d_i\}$ denote the set of vertices corresponding to the i^{th} vehicle, and let E_i denote the set of all the edges that join any two distinct vertices in V_i . For $i \in \{1, \dots, n\}$, let the cost of traversing an edge $e \in E_i$ for the i^{th} vehicle be denoted by cost_e^i . We assume the travel costs for each vehicle satisfy the triangle inequality and are monotonic *i.e.*, for any two vehicles $i < j$, $\text{cost}_e^i \leq \text{cost}_e^j$. However, we do not assume that they are proportional (*i.e.* $\text{cost}_e^i = \rho \cdot \text{cost}_e^j$). A tour for vehicle i is given by the sequence $(d_i, u_{i1}, \dots, u_{il_i}, d_i)$ where $u_{ij} \in T$ for $j = 1, \dots, l_i$ and l_i denotes the number of targets visited by vehicle i . The travel cost of vehicle i is equal to $\text{cost}_{(d_i, u_{i1})}^i + \sum_{j=1}^{l_i-1} \text{cost}_{(u_{ij}, u_{i(j+1)})}^i + \text{cost}_{(u_{il_i}, d_i)}^i$ if $l_i \geq 1$, and is equal to 0 otherwise. The objective is to find a tour for each vehicle such that each target is visited exactly once by some vehicle and the sum of travel costs of all the vehicles is minimum.

Even though the travel costs are monotonic, the partitioning of the targets amongst the vehicles is still non-trivial because the vehicles start their tours from different initial locations or depots. This is an important special case that naturally arises in the following practical applications: (1) If the vehicles are modelled as ground robots [19] traveling with the same speed but with different turning radius constraints and r_i denotes the minimum turning radius of vehicle i , then the vehicles can be ordered such that $r_1 \leq r_2 \leq \dots \leq r_n$. In this case, the travel costs (the shortest distances required to travel between targets subject to the turning radius constraints) are monotonic. (2) If the vehicles travel at different speeds and the travel cost for any vehicle between two targets is defined as the ratio of the Euclidean distance between the targets and the speed of the vehicle, then the travel costs satisfy the case of proportional costs, which are also monotonic. (3) If each vehicle has a fuel capacity and the vehicles are allowed to refuel at gas stations, then the cost of traveling between any two targets subject to the refueling constraints also increases as the fuel capacity of a vehicle decreases [9]. Here, again, the travel costs are monotonic.

¹ The travel costs for the vehicles may be the same but there are compatibility constraints between vehicles and targets.

When $n = 2$ and the travel costs are monotonic, a 2-approximation algorithm was presented for MDHTSP in [2] extending the primal-dual algorithm for the prize collecting TSP [7] to the two vehicle case. Similar to the prize-collecting TSP in which any target not visited by a vehicle must pay a penalty, for the two vehicle case, any target not visited by the first vehicle is on the tour of the second vehicle that is modeled by some form of penalty.

For the multiple vehicle case we address, it is not difficult to design a primal-dual algorithm to find a feasible solution where each target is connected to some depot. However, the key to proving a good approximation ratio is in the pruning procedure of such an algorithm so that the edges retained after the pruning procedure can be paid for by an appropriate dual solution. A closely related problem where a similar pruning procedure arises is the Prize Collecting Steiner Tree problem (PCST)². In the primal-dual algorithm for the PCST [7], the greedy growth of the duals must be frozen due to the constraints represented by the penalties on the nodes, particularly when the total sum of duals associated with any subset of vertices reaches the total sum of penalties for this set. How such frozen components are handled in terms of whether they are connected to the final solution tree or not is the crucial step of the pruning procedure in [7]. This is accomplished by a labeling procedure which was also used in [2] for the two vehicle case. However, its direct generalization to the many vehicles case we address is much more involved. In this paper, we provide an alternate simpler view of the pruning procedure for the PCST and re-purpose it for our multi-vehicle extension. This is the main technical novelty in our work.

Contributions. We present a primal-dual 2-approximation algorithm for the n vehicle case of monotonic MDHTSP. Like the prior work [2], we use a primal-dual approach, but avoid the use of the prize-collecting TSP as explained above. The heart of our result is a 2-approximation for the Heterogeneous Spanning Forest (HSF) problem of finding a minimum cost collection of n trees from the depots covering all the targets among the different graphs corresponding to the vehicles³. The following is a summary of our key technical steps.

1. While multiple LP relaxations are possible for the MDHTSP, we present an LP relaxation and a dual that allows the primal-dual method to construct a HSF with a special nesting structure among its components (Lemma 3).
2. This structure is then used to prove that pruning appropriate edges from the output of the main loop of the primal-dual method will result in a feasible HSF (Lemma 4 and Theorem 5).
3. Finally, we show that the value of the dual can be decomposed into the sum of the dual values corresponding to each of the vehicles (Lemma 8). This allows us to decompose the proof of the bound on the cost of the edges in each tree in terms of its corresponding dual value.

Putting together the above components, we show that the cost of the HSF constructed using the proposed algorithm is at most the optimal LP relaxation cost of the MDHTSP. Short-cutting the Euler tours on the trees in this HSF provides a 2-approximation algorithm for the MDHTSP for the n -vehicle case (Theorem 7).

² Given a graph, a depot node, a penalty for each node and a cost of each edge in the graph, the objective of the PCST is to find a tree containing the depot such that the sum of the cost of all the edges present in the tree and the penalty of all the nodes not present in the tree is minimum.

³ This result does not require the different costs to be metric, but only that they are monotonic across the vehicles. The metric condition is only used in converting the forests to tours.

2 LP Relaxation for MDHTSP and its Dual

We use two sets of integer variables that will be later relaxed to formulate an LP relaxation for the MDHTSP. The first set of variables denoted by x_e^i determines whether edge $e \in E_i$ is present in the tour of vehicle $i \in \{1, \dots, n\}$. The second set of variables z_U^i is defined for $i = 1, \dots, n-1$ and any $U \subseteq T$. Specifically, z_U^i is equal to 1 if U is the subset of all the targets visited by vehicles $i+1, \dots, n$; otherwise, z_U^i is equal to 0. Refer to Fig. 1 for an illustration of these variables.

Note that $\sum_{U \subseteq T} z_U^i = 1 \forall i \in \{1, \dots, n-1\}$, i.e., every vehicle of index i up to $n-1$ picks a single subset of targets that will be covered by vehicles $i+1$ or later.

The following proposition is a simple consequence of the fact that the z -variables can be used to identify subsets of targets that need to be covered by vehicle i . For any $S \subseteq T$ and $i = 1, \dots, n$, let $\delta_i(S) := \{(u, v) : u \in S, v \in V_i \setminus S\}$.

► **Proposition 1.** *Any feasible solution to the MDHTSP satisfies the following constraints:*

$$\sum_{e \in \delta_i(S)} x_e^i \geq 2 \sum_{U: S \subseteq U \subseteq T} (z_U^{i-1} - z_U^i) \quad \forall S \subseteq T, |S| \geq 1, i \in \{1, \dots, n\};$$

$$z_T^0 = 1; \quad z_U^0 = 0 \quad \forall U \neq T; \quad z_U^n = 0 \quad \forall U \subseteq T.$$

Proof. Both $\sum_{U: S \subseteq U \subseteq T} z_U^{i-1}$ and $\sum_{U: S \subseteq U \subseteq T} z_U^i$ can either be 0 or 1. The constraint is trivially satisfied except for the case when $\sum_{U: S \subseteq U \subseteq T} z_U^{i-1} = 1$ and $\sum_{U: S \subseteq U \subseteq T} z_U^i = 0$. But $\sum_{U: S \subseteq U \subseteq T} z_U^{i-1} = 1$ can occur only if all the targets in S are visited by vehicles in $\{i, \dots, n\}$. Also, $\sum_{U: S \subseteq U \subseteq T} z_U^i = 0$ can occur only if there is at least one target in S visited by a vehicle in $\{1, \dots, i\}$. Therefore, if $\sum_{U: S \subseteq U \subseteq T} z_U^{i-1} = 1$ and $\sum_{U: S \subseteq U \subseteq T} z_U^i = 0$, there is at least one target in S that must be visited by vehicle i . This is implied by the constraint as it reduces to $\sum_{e \in \delta_i(S)} x_e^i \geq 2$ which is true. ◀

The LP relaxation of the MDHTSP that we work with contains only the constraints from the above proposition.

$$Cost_{lp} = \min \sum_{i=1}^n \sum_{e \in E_i} cost_e^i x_e^i \quad (1)$$

$$\sum_{e \in \delta_i(S)} x_e^i \geq 2 \sum_{U: S \subseteq U \subseteq T} (z_U^{i-1} - z_U^i) \quad \forall S \subseteq T, i = 1, \dots, n, \quad (2)$$

$$z_T^0 = 1; \quad z_U^0 = 0 \quad \forall U \neq T; \quad z_U^n = 0 \quad \forall U \subseteq T \quad (3)$$

$$x_e^i \geq 0 \quad \forall e \in E_i, i = 1, \dots, n, \quad (4)$$

$$z_U^i \geq 0 \quad \forall U \subseteq T, i = 1, \dots, n-1. \quad (5)$$

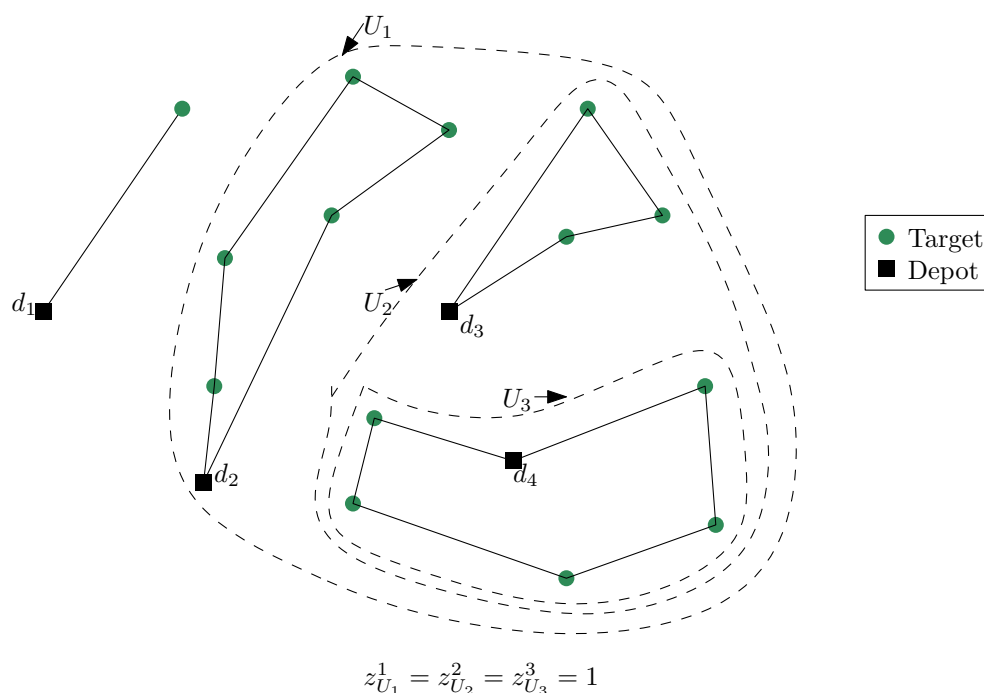
The dual of the above linear program is as follows:

$$\max 2 \sum_{S \subseteq T} Y_1(S) \quad (6)$$

$$\sum_{S: e \in \delta_i(S)} Y_i(S) \leq cost_e^i \quad \forall e \in E_i, \forall i = 1, \dots, n, \quad (7)$$

$$\sum_{S: S \subseteq U} Y_i(S) \leq \sum_{S: S \subseteq U} Y_{i+1}(S) \quad \forall U \subseteq T, \forall i = 1, \dots, n-1, \quad (8)$$

$$Y_i(S) \geq 0 \quad \forall S \subseteq T, \forall i = 1, \dots, n. \quad (9)$$



■ **Figure 1** An illustration of a feasible solution involving four vehicles. In this example, $U_1 \supseteq U_2 \supseteq U_3$.

Other than the usual packing constraints in (7), the more interesting constraints (8) restrict the dual value of the subsets of any set U in a cheaper cost level to be at most that accumulated in any higher cost level.

3 Primal-Dual Algorithm

We define some terms before presenting the main steps of the primal-dual algorithm. The algorithm maintains a forest of edges defined over the targets and the respective depot for each vehicle separately, by using the known primal-dual moat-growing procedure separately and simultaneously in each of the graphs (V_i, E_i) [1, 7].

Let $F_i(t)$ be the forest obtained in (V_i, E_i) for the vehicle $i \in \{1, \dots, n\}$ at the end of iteration t of the main loop. For any two distinct vehicles $i, j \in \{1, \dots, n\}, i < j$ and any iteration t , consider (connected) components C_i in forest $F_i(t)$ and C_j in forest $F_j(t)$. Because of the monotonicity of costs across the indices, typically the edges between targets in the lower level i will become packed and hence chosen in the forest before those in the higher cost level j . C_i is considered as an ancestor of C_j , or C_j is a descendant of C_i if $C_i \supseteq C_j$. Note that C_j cannot be a descendant of C_i if it contains a depot. Note carefully that we have the subset inclusion only among the targets in the components in different levels but not among the set of tight edges connecting them.

A component can either be active, inactive or frozen. A component is **active** at the start of an iteration if its dual variable will be allowed to increase during the iteration without violating any of the constraints in the dual problem. A component is **inactive** if it either contains a depot or if any of its ancestor contains a depot. A component is **frozen** if it stopped growing due to the constraint in (8), *i.e.*, each descendant of this component is not active as it contains targets connected to some higher level depots. If a component is inactive or frozen at the start of an iteration, its dual value will not change during the iteration.

Initialization. Initially, each forest consists of singleton components which is either a depot or a target. The singleton components with only targets are active; any component with a depot is inactive. The dual variables corresponding to all the active components are initialized to zero.

Main loop. In each iteration of the primal-dual algorithm, the dual variables of all active components in all the forests (in all graphs (V_i, E_i)) are increased simultaneously as much as possible by the same amount until at least one of the constraints in the dual problem becomes tight. If multiple constraints in both (7) and (8) become tight simultaneously during iteration t , only one constraint either in (7) or (8) is chosen and processed based on the following procedure.

- If constraints in (7) become tight, then a tight constraint corresponding to the vehicle **with the least vehicle number** (say i) is chosen. The edge corresponding to the chosen constraint is added to the forest corresponding to vehicle i merging two components at its ends (say C_1 and C_2). If both C_1 and C_2 do not contain a depot, then the merged component is active. If one of the components contains a depot (say $d_i \in C_2$), then the merged component and all its descendants (specifically the descendants of C_1) become inactive.
- If a constraint in (8) becomes tight (*i.e.*, it risks being violated if the current dual variables all continue to grow) for some vehicle i , then the corresponding component is deactivated and becomes frozen.

The main loop of the algorithm stops when each component in all the forests is either inactive or frozen. We will detail some properties of the components before describing the pruning step of the algorithm in Section 3.1.

Remarks.

1. If a component C_1 merges with a component C_2 that contains a depot, the merged component and its descendants are deactivated and will never become active again in the main loop.
2. It is straightforward to compute the maximum possible increase (Δ_1) in the dual variables that do not violate the constraints in (7) using standard techniques involving internal variables for each target in polynomial time. Since we do not grow dual variables more than this amount, it is straightforward to verify that the dual solution we construct obeys the constraints in (7). Specifically, we can use internal variables $p_i(u)$ defined for each target u and vehicle i in the following way: All these internal variables are first set to zero during the initialization. Suppose, at the start of an iteration, C_1 and C_2 are two components corresponding to vehicle i , and $u \in C_1$ and $v \in C_2$. Assume at least one of these components is active. Let $active(C)$ denote if a component C is active or not. For $j = 1, 2$, $active(C_j) = 1$ if C_j is active and is equal to 0 otherwise. Then, the maximum amount by which the dual variable corresponding to C_1 or C_2 can be increased before violating the constraint corresponding to edge $e = (u, v)$ is given by $\frac{Cost_e^i - p_i(u) - p_i(v)}{active(C_1) + active(C_2)}$. This amount can be computed for all such candidate edges and the least of these amounts is equal to Δ_1 . During an iteration, if the primal-dual algorithm decides to increase the dual variable of each active component by Δ , then $p_i(u)$ for each target u and vehicle i will be set to $p_i(u) + \Delta$ if the component containing u is active; otherwise $p_i(u)$ doesn't change.
3. Similarly, we can compute the maximum possible increase (Δ_2) in the dual variables that do not violate the constraints in (8), using internal variables defined for each component again in polynomial time. Specifically, we use two internal variables $\bar{Y}_i(C)$ and $Bound_i(C)$

for each component $C \in F_i(t)$, $i = 1, \dots, n-1$ defined⁴ as follows: $\bar{Y}_i(C) := \sum_{S:S \subseteq C} Y_i(C)$ and $Bound_i(C) := \sum_{S:S \subseteq C} Y_{i+1}(C)$. All the internal variables are first set to zero during the initialization ($t = 0$). For any $i = 1, \dots, n-1$, suppose at the end of iteration t , an active component $C \in F_i(t)$ has m_C active descendants in $F_{i+1}(t)$. Then, during iteration $t+1$, the constraint corresponding to C in (8) can become tight only if $m_C = 0$. In the case $m_C = 0$, the maximum amount by which the dual variable of C can be increased without violating its constraint is given by $Bound_i(C) - \bar{Y}_i(C)$. This amount can be computed for each of components in $\{C : C \text{ is active and } m_C = 0, C \in F_i(t+1), i = 1, \dots, n-1\}$ and the least of these amounts is equal to Δ_2 . In addition, if the primal-dual algorithm decides to increase the dual variable of each active component by Δ during iteration $t+1$, before any merger occurs, for all $i = 1, \dots, n-1$, for all active $C \in F_i(t)$, $\bar{Y}_i(C) \leftarrow \bar{Y}_i(C) + \Delta$ and $Bound_i(C) \leftarrow Bound_i(C) + m_C \Delta$; in addition, if two components $C_1, C_2 \in F_i(t)$ merge, $\bar{Y}_i(C_1 \cup C_2) \leftarrow \bar{Y}_i(C_1) + \bar{Y}_i(C_2)$ and $Bound_i(C_1 \cup C_2) \leftarrow Bound_i(C_1) + Bound_i(C_2)$.

Observations on the Main Loop. We review a few facts that follow from the running of the main loop. Consider any $i \in \{1, \dots, n\}$ and a vertex $u \in T$. Let $C_i(t, u)$ denote the component containing u in the forest corresponding to vehicle i at the start of iteration t of the main loop. Let $active_i(t, u)$ be an indicator denoting if $C_i(t, u)$ is active or not. That is, $active_i(t, u) = 1$ if $C_i(t, u)$ is active and $active_i(t, u) = 0$ otherwise.

► **Lemma 2.** *The main loop of the primal-dual algorithm terminates in $2n(|T|+1)$ iterations.*

Proof. At the start of the main loop, the number of components in all the forests is $n(|T|+1)$ and the number of active components in all the forests is $n|T|$. During each iteration of the main loop, the sum of the number of components and the number of active components in all the forests decreases by at least 1. Therefore, the main loop will require at most $2n|T|+n$ iterations. ◀

Note that components in smaller index vehicles typically merge before their corresponding analogues in the larger indices since the distances are shorter in the smaller index metric. Hence, in addition to the laminar structure on the target nodes among these components for a fixed vehicle, there is an inclusion relation among these components when viewing them across the vehicle indices (after we ignore the depots that are present only in their corresponding graphs). The following lemma shows that at any time t' in the main loop of the algorithm, for $j > i$, the set of connected components for vehicle index j are contained in those for vehicle index i . Let the main loop of the primal-dual procedure terminate after t_f iterations. Also, let Δ_t denote the amount by which the dual value of each active component is increased during iteration t .

► **Lemma 3.** *In any iteration $t' = 1, \dots, t_f$, for any vehicles $i, j \in \{1, \dots, n\}, i < j$ and any target $u \in T$, the following relations hold true: $active_i(t', u) \geq active_j(t', u)$ and $C_i(t', u) \supseteq C_j(t', u)$ if $d_j \notin C_j(t', u)$.*

Proof. We prove this lemma by induction on t' . For $t' = 1$, the lemma is trivially satisfied. Assume the lemma is true for iterations $t' = 1, \dots, t$ for some $t < t_f$.

Suppose that in iteration t , the constraint in (7) becomes tight for some edge (u, v) corresponding to vehicle i . There are two cases.

⁴ In the special case when C only consists of d_i , we define $Bound_i(C) = 0$.

- *The merged component does not contain d_i* : In this case, the merged component will be active at the end of the iteration and will be an ancestor to the descendants of $C_i(t, u)$ and $C_i(t, v)$. In addition, if $i \geq 2$, $C_{i-1}(t, u)$ must be the same component as $C_{i-1}(t, v)$. If this is not true, for edge $e = (u, v)$,

$$\begin{aligned} \text{cost}_e^{i-1} &= \sum_{t'=1}^{t-1} (\text{active}_{i-1}(t', u) + \text{active}_{i-1}(t', v)) \Delta_{t'} \\ &\leq \text{cost}_e^i - \sum_{t'=1}^{t-1} (\text{active}_i(t', u) + \text{active}_i(t', v)) \Delta_{t'}. \end{aligned} \quad (10)$$

Therefore, during iteration t , the algorithm would have added (u, v) to the forest corresponding to vehicle $i - 1$ due to our rule in processing the vehicle with the least index in the main loop, which is a contradiction. One can now verify that for any target u , $\text{active}_i(t + 1, u) \geq \text{active}_j(t + 1, u)$ and $C_i(t + 1, u) \supseteq C_j(t + 1, u)$ if $d_j \notin C_j(t + 1, u)$.

- *Merged component contains d_i* : If $d_i \in C_i(t, u)$, then the merged component and all the descendants of the merged component also become inactive. Again, one can verify the lemma is true for iteration $t + 1$.

Suppose a constraint in (8) becomes tight for some component C . If C is frozen, then C cannot have any descendants that are active. Again, it is easy to check that the lemma is true for iteration $t + 1$. ◀

► **Lemma 4.** *Consider a frozen component C corresponding to vehicle $i \in \{1, \dots, n - 1\}$ at the start of iteration t . Consider any target $u \in C$. Then, either $C_{i+1}(t, u)$ is frozen and $C_{i+1}(t, u) \subseteq C$ or $C_{i+1}(t, u)$ contains d_{i+1} and $C_{i+1}(t, u) \setminus \{d_{i+1}\} \subseteq C$.*

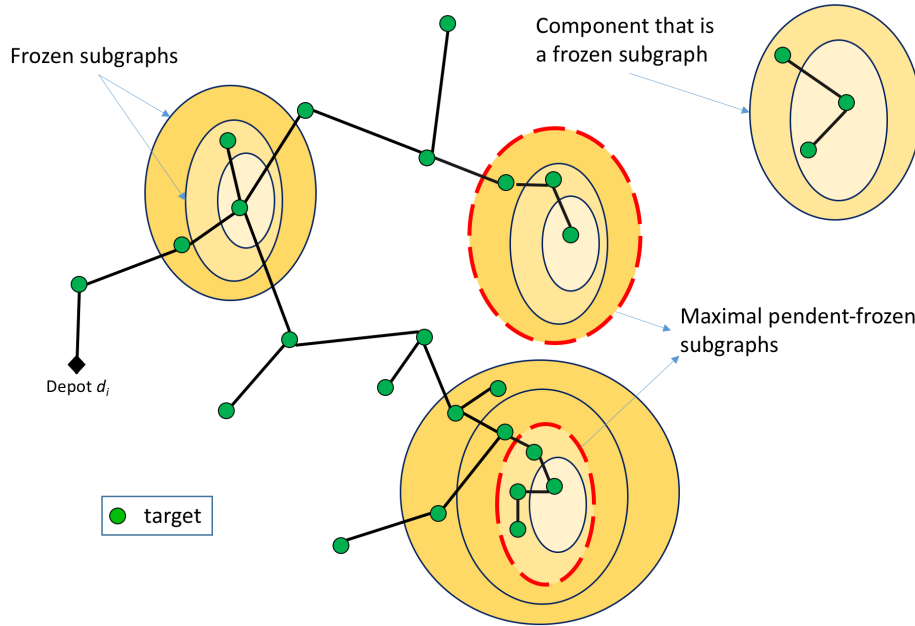
Proof. If $d_{i+1} \notin C_{i+1}(t, u)$, using Lemma 3, $C_{i+1}(t, u) \subseteq C_i(t, u) = C$; also, since $C_{i+1}(t, u)$ is not active and cannot⁵ have an ancestor that is connected to a depot, $C_{i+1}(t, u)$ is frozen. If $d_{i+1} \in C_{i+1}(t, u)$, then any target $v \in C_{i+1}(t, u) \setminus \{d_{i+1}\}$ must also belong to C . If this is not the case, there is an edge joining targets w and v in component $C_{i+1}(t, u)$ such that $w \in C$ and $v \notin C$. From Lemma 3, using a similar argument as in (10), this is not possible. ◀

3.1 Pruning

For $i = 1, \dots, n$, the pruning phase of the primal-dual procedure selects a subgraph \bar{F}_i of $F_i(t_f)$ such that each target is connected to exactly one of the depots. We need a few definitions before describing the pruning procedure. The degree of a subgraph C of forest F is defined as $|\{(u, v) : u \in C, v \notin C, (u, v) \in F\}|$. A subgraph C of $F_i(t_f)$ is referred to as a **frozen subgraph** if C is a component that is frozen during some iteration $t \leq t_f$. A subgraph C of $F_i(t_f)$ is referred to as a **pendent-frozen subgraph** if C is a frozen subgraph and its degree is equal to 1 (Ref to Fig. 2). A **maximal pendent-frozen subgraph** is a pendent-frozen subgraph $C \in F_i(t)$ such that there is no other pendent-frozen subgraph $C' \in F_i(t)$ with $C' \supset C$. Given vehicle i , iteration t of the main loop and a component C spanning a subset of targets, let $F_i^C(t)$ be a subgraph of $F_i(t)$ induced by d_i and the targets in C . It follows from this definition that if C was frozen during iteration t of the main loop for vehicle i , then for all $j = i, i + 1, \dots, n$, $F_j^C(t) = F_j^C(t_f)$.

The pruning procedure is implemented in n iterations. Let $i := 1$ at the start of the pruning procedure and let $G_1 = F_1(t_f)$.

⁵ If $C_{i+1}(t, u)$ has an ancestor that is connected to a depot, then C also has an ancestor that is connected to a depot which makes C inactive and this is not possible.



■ **Figure 2** An illustration of forest G_i corresponding to vehicle i . Each shaded region corresponds to a frozen subgraph.

1. Remove all the frozen subgraphs that are components (not containing d_i) from G_i . Furthermore, in the tree containing d_i in G_i , remove all the maximal pendent-frozen subgraphs. The resultant pruned tree is \bar{F}_i (Refer to Figs. 2, 3).
2. If $i = n$, stop. Else, let G_{i+1} be the union of all the subgraphs $F_{i+1}^C(t_C)$ obtained for every frozen subgraph C frozen at time t_C and discarded from G_i in the previous step. Set $i = i + 1$ and go to step 1.

Intuitively, frozen components contain targets connected to depots in higher index graphs and hence the pruning step discards them for processing in an appropriate (later) iteration. Similarly, maximal pendent-frozen subgraphs must be pruned so as to ensure that no inactive component in this index contributes degree one in the standard degree-based inductive argument for the 2-approximation ratio in the primal-dual method [7].

3.2 Feasibility

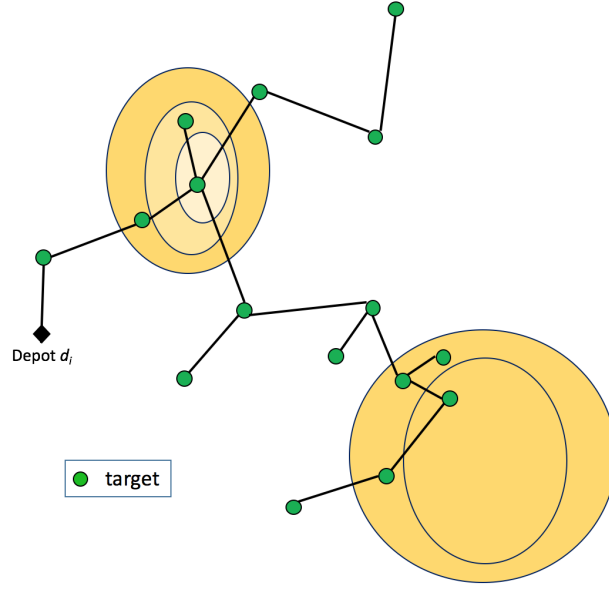
We are ready to prove the following main theorem.

► **Theorem 5.** *Each target in T is connected to exactly one of the depots in $\{\bar{F}_1, \dots, \bar{F}_k\}$, i.e., $\{\bar{F}_1, \dots, \bar{F}_k\}$ is a feasible Heterogeneous Spanning Forest (HSF).*

Proof. We will prove the Theorem by induction on the stages of the pruning procedure.

► **Lemma 6.** *Consider any iteration k of the pruning procedure. Let the subset of targets connected to d_i in \bar{F}_i be denoted as T_i . Then, each target in $\bigcup_{i=1}^k T_i$ is connected to exactly one of the depots in $\{\bar{F}_1, \dots, \bar{F}_k\}$ and the remaining targets are contained in the frozen subgraphs discarded in G_k .*

Proof. Clearly the lemma is true for $k = 1$. Assume that the lemma is true for $k = k' - 1$. We will now show that the lemma is true for $k = k'$ also. Applying Lemma 4 to each discarded frozen subgraph in $G_{k'-1}$, all the targets in $T \setminus \bigcup_{i=1}^{k'-1} T_i$ are either present in the



■ **Figure 3** Output forest \bar{F}_i after removing the maximal pendent-frozen subgraphs and frozen subgraphs that are components from G_i .

frozen components of $G_{k'}$ or connected to $d_{k'}$ using only targets from $T \setminus \bigcup_{i=1}^{k'-1} T_i$. By the definition of the pruning step on $G_{k'}$, the targets that are pruned away do not affect the connectivity from $d_{k'}$ to the targets retained in $T_{k'}$. Hence the induction step is proved. ◀

By the above lemma, when $k = n$, we see that the targets T_i covered in the various iterations form a partition of T . ◀

4 Approximation Guarantee

► **Theorem 7.** *The approximation ratio of the primal-dual algorithm for MDHTSP is 2.*

We show that the dual value of the LP relaxation can be equivalently written as the sum of the dual values corresponding to each of the vehicles (Lemma 8). This will allow us to bound the cost of the edges in each forest with respect to its dual value (Lemma 9). The approximation ratio will readily follow from these results.

Consider any vehicle $i \in \{1, \dots, n-1\}$. Let C_{i1}, \dots, C_{im_i} denote the discarded, frozen subgraphs of G_i (as defined in the pruning procedure) for vehicle i . Also, let the subset of targets in all these discarded components be U_i . To simplify the ensuing derivation (with a slight abuse of notation), we also refer to C as a subset of targets present in component C .

► **Lemma 8.**

$$\sum_{S \subseteq T} Y_1(S) = \sum_{S \subseteq T, S \not\subseteq U_1} Y_1(S) + \sum_{j=2}^{n-1} \sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S) + \sum_{S \subseteq U_{n-1}} Y_n(S).$$

Proof. Note that

$$\sum_{S \subseteq T} Y_1(S) = \sum_{S \subseteq T, S \not\subseteq U_1} Y_1(S) + \sum_{S \subseteq U_1} Y_1(S). \quad (11)$$

For any $j = 2, \dots, n-1$,

$$\sum_{S \subseteq U_{j-1}} Y_{j-1}(S) = \sum_{k=1}^{m_{j-1}} \sum_{S \subseteq C_{(j-1)k}} Y_{j-1}(S).$$

Each $C_{(j-1)k}$ is a frozen component. Therefore, its corresponding constraint in (8) is tight.

$$\begin{aligned} \Rightarrow \sum_{S \subseteq U_{j-1}} Y_{j-1}(S) &= \sum_{k=1}^{m_{j-1}} \sum_{S \subseteq C_{(j-1)k}} Y_j(S) = \sum_{S \subseteq U_{j-1}} Y_j(S) \\ &= \sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S) + \sum_{S \subseteq U_j} Y_j(S). \end{aligned}$$

Applying the above relation recursively in equation (11), the lemma follows. \blacktriangleleft

For any $j = 1, \dots, n$, let $Cost(\bar{F}_j) = \sum_{e \in \bar{F}_j} cost_e^j$.

► **Lemma 9.** For any $j = 2, \dots, n-1$,

$$Cost(\bar{F}_j) \leq 2 \sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S).$$

Proof.

$$Cost(\bar{F}_j) = \sum_{e \in \bar{F}_j} cost_e^j = \sum_{e \in \bar{F}_j} \sum_{S: e \in \delta_j(S)} Y_j(S) = \sum_{S \subseteq T} Y_j(S) |\delta_j(S) \cap \bar{F}_j|.$$

Note that from Lemma 3, any $S \subseteq T$ that loaded an edge $e \in \bar{F}_j$ must be a subset of U_{j-1} . In addition, $|\delta_j(S) \cap \bar{F}_j| = 0$ for any $S \subseteq U_j$, since U_j was discarded in the pruning. Therefore, we get

$$Cost(\bar{F}_j) = \sum_{S \subseteq T} Y_j(S) |\delta_j(S) \cap \bar{F}_j| = \sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S) |\delta_j(S) \cap \bar{F}_j|.$$

Therefore, the lemma reduces to proving that

$$\sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S) |\delta_j(S) \cap \bar{F}_j| \leq 2 \sum_{S \subseteq U_{j-1}, S \not\subseteq U_j} Y_j(S). \quad (12)$$

The above result can be proved by induction on the main loop, using the usual degree argument for such primal-dual algorithms [7]. At the start of any iteration t and vehicle j , let A denote the set of active components defined as follows: $A := \{C : C \text{ is active, } C \subseteq U_{j-1}, C \not\subseteq U_j\}$. Similarly, let I denote the set of inactive or frozen components defined as follows $I := \{C : C \text{ is inactive or frozen, } C \subseteq U_{j-1}, C \not\subseteq U_j\}$. Form a graph H with components in $A \cup I$ as vertices and $e \in \bar{F}_j \cap \delta_j(C)$ for $C \in A \cup I$ as edges.

Let $deg(u)$ represent the degree of a vertex u in graph H . Let the dual variable of each active component during the iteration increase by Δ_t . Due to this dual increase, the right hand side of the inequality (12) will increase by $2\Delta_t|A|$, whereas the left hand side of the inequality (12) will increase by $\Delta_t \sum_{u \in A} deg(u)$. Therefore, the lemma is proved if we can show that $\sum_{u \in A} deg(u) \leq 2|A|$.

Note that H is a tree that spans all the the components in $A \cup I$. Therefore, $deg(u) \geq 1$ for any component u in $A \cup I$. There is exactly one inactive component in I , and this inactive component contains d_j ⁶. In addition, for any vertex u that represents a frozen component,

⁶ A component in a forest corresponding to vehicle j can also be inactive if its ancestor is connected to a depot. But from Lemma 3, such a component never becomes active again and never gets connected to d_j .

$\deg(u) \geq 2$ due to the pruning procedure that discards maximal pendant-frozen subgraphs. Therefore,

$$\begin{aligned} \sum_{u \in A} \deg(v) &= \sum_{u \in A \cup I} \deg(u) - \sum_{u \in I} \deg(u) \\ &= \sum_{u \in A \cup I} \deg(u) - \sum_{u \in \{C: C \in I, d_j \notin C\}} \deg(u) - \sum_{u \in \{C: C \in I, d_j \in C\}} \deg(u) \\ &\leq 2(|A| + |I| - 1) - 2(|I| - 1) - 1 \\ &< 2|A|. \end{aligned} \quad \blacktriangleleft$$

Similarly, one can also show that $\text{Cost}(\bar{F}_1) \leq 2 \sum_{S \subseteq T, S \not\subseteq U_1} Y_1(S)$ and $\text{Cost}(\bar{F}_n) \leq 2 \sum_{S \subseteq U_{n-1}} Y_n(S)$. Hence, using Lemma 8, we get

$$\begin{aligned} \sum_{i=1}^n \text{Cost}(\bar{F}_i) &\leq 2 \sum_{S \subseteq T, S \not\subseteq U_1} Y_1(S) + 2 \sum_{j=1}^{n-2} \sum_{S \subseteq U_j, S \not\subseteq U_{j+1}} Y_{j+1}(S) + 2 \sum_{S \subseteq U_{n-1}} Y_n(S) \\ &= 2 \sum_{S \subseteq T} Y_1(S) \leq \text{Cost}_{lp}. \end{aligned}$$

Therefore, the cost of the constructed HSF will be at most the optimal cost of the MDHTSP. Doubling the edges in the constructed HSF and short cutting the repeated visits to the targets in an Euler walk suitably leads to a 2-approximation algorithm for the MDHTSP.

References

- 1 Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM journal on Computing*, 24(3):440–456, 1995.
- 2 Jung Yun Bae and Sivakumar Rathinam. A primal-dual approximation algorithm for a two depot heterogeneous traveling salesman problem. *Optimization Letters*, 10, 2015. doi:10.1007/s11590-015-0924-1.
- 3 Jungyun Bae and Sivakumar Rathinam. Approximation algorithms for multiple terminal, hamiltonian path problems. *Optimization Letters*, 6(1):69–85, 2012. doi:10.1007/s11590-010-0252-4.
- 4 Bruno N. Coelho, Vitor N. Coelho, Igor M. Coelho, Luiz S. Ochi, Roozbeh Haghazari K., Demetrius Zuidema, Milton S.F. Lima, and Adilson R. da Costa. A multi-objective green uav routing problem. *Comput. Oper. Res.*, 88(C):306–315, December 2017. doi:10.1016/j.cor.2017.04.011.
- 5 R. Doshi, S. Yadlapalli, S. Rathinam, and S. Darbha. Approximation algorithms and heuristics for a 2-depot, heterogeneous hamiltonian path problem. *International Journal of Robust and Nonlinear Control*, 21(12):1434–1451, 2011. doi:10.1002/rnc.1701.
- 6 A.M. Frieze. An extension of christofides heuristic to the k-person travelling salesman problem. *Discrete Applied Mathematics*, 6(1):79–83, 1983. doi:10.1016/0166-218X(83)90102-6.
- 7 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, April 1995.
- 8 I. Gortz, M. Molinaro, V. Nagarajan, and R. Ravi. *Capacitated Vehicle Routing with Non-uniform Speeds*, volume 6655 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011.
- 9 Samir Khuller, Azarakhsh Malekian, and Julián Mestre. To fill or not to fill: The gas station problem. *ACM Trans. Algorithms*, 7(3):36:1–36:16, July 2011. doi:10.1145/1978782.1978791.

- 10 David Levy, Kaarthik Sundar, and Sivakumar Rathinam. Heuristics for routing heterogeneous unmanned vehicles with fuel constraints. *Mathematical Problems in Engineering*, 2014, 2014.
- 11 Parikshit Maini and P.B. Sujit. On cooperation between a fuel constrained uav and a refueling ugv for large scale mapping applications. In *2015 International Conference on Unmanned Aircraft Systems*, 2015. doi:10.1109/ICUAS.2015.7152432.
- 12 W. Malik, S. Rathinam, and S. Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Operations Research Letters*, 35:747–753, 2007.
- 13 P. Oberlin, S. Rathinam, and S. Darbha. Today’s traveling salesman problem. *IEEE Robotics and Automation Magazine*, 17(4):70–77, December 2010.
- 14 Alena Otto, Niels A. H. Agatz, James F. Campbell, Bruce L. Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018. doi:10.1002/net.21818.
- 15 Stefan Poikonen, Xingyin Wang, and Bruce Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017. doi:10.1002/net.21746.
- 16 Amritha Prasad, Shreyas Sundaram, and Han-Lim Choi. Min-max tours for task allocation to heterogeneous agents. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1706–1711, 2018. doi:10.1109/CDC.2018.8619118.
- 17 S. Rathinam, R. Sengupta, and S. Darbha. A resource allocation algorithm for multivehicle systems with nonholonomic constraints. *IEEE Transactions Automation Science and Engineering*, 4:98–104, 2007. doi:10.1109/TASE.2006.872110.
- 18 Sivakumar Rathinam and Raja Sengupta. 3/2-approximation algorithm for two variants of a 2-depot hamiltonian path problem. *Operations Research Letters*, 38(1):63–68, 2010. doi:10.1016/j.orl.2009.10.001.
- 19 J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, 145(2):367–393, 1990.
- 20 Kaarthik Sundar and Sivakumar Rathinam. An exact algorithm for a heterogeneous, multiple depot, multiple traveling salesman problem. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 366–371. IEEE, 2015.
- 21 Zhou Xu and Brian Rodrigues. An extension of the christofides heuristic for the generalized multiple depot multiple traveling salesmen problem. *European Journal of Operational Research*, 257(3):735–745, 2017. doi:10.1016/j.ejor.2016.08.054.
- 22 S. Yadlapalli, S. Rathinam, and S. Darbha. 3-approximation algorithm for a two depot, heterogeneous traveling salesman problem. *Optimization Letters*, pages 1–12, 2010.
- 23 Miao Yu, Viswanath Nagarajan, and Siqian Shen. An approximation algorithm for vehicle routing with compatibility constraints. *Operations Research Letters*, 46(6):579–584, 2018. doi:10.1016/j.orl.2018.10.002.

On the Parameterized Complexity of Grid Contraction

Saket Saurabh

The Institute Of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway
saket@imsc.res.in

Uéverton dos Santos Souza

Fluminense Federal Universidade, Niterói, Brazil
ueverton@ic.uff.br

Prafullkumar Tale

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
prafullkumar.tale@mpi-inf.mpg.de

Abstract

For a family of graphs \mathcal{G} , the \mathcal{G} -CONTRACTION problem takes as an input a graph G and an integer k , and the goal is to decide if there exists $F \subseteq E(G)$ of size at most k such that G/F belongs to \mathcal{G} . Here, G/F is the graph obtained from G by contracting all the edges in F . In this article, we initiate the study of GRID CONTRACTION from the parameterized complexity point of view. We present a fixed parameter tractable algorithm, running in time $c^k \cdot |V(G)|^{\mathcal{O}(1)}$, for this problem. We complement this result by proving that unless ETH fails, there is no algorithm for GRID CONTRACTION with running time $c^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$. We also present a polynomial kernel for this problem.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Grid Contraction, FPT, Kernelization, Lower Bound

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.34

Funding Saket Saurabh: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819416), and Swarnajayanti Fellowship (No DST/SJF/MSA01/2017-18).

Prafullkumar Tale: This research is a part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement SYSTEMATICGRAPH (No. 725978).



Acknowledgements Some part of this project was completed when the third author was a Senior Research Fellow at The Institute of Mathematical Sciences, HBNI, Chennai, India. The project started when the second and the third authors were visiting Prof. Michael Fellows at the University of Bergen, Bergen, Norway. Both the authors would like to thank Prof. Michael Fellows for the invitation.

1 Introduction

Graph modification problems are one of the central problems in graph theory that have received a lot of attention in theoretical computer science. Some of the important graph modification operations are vertex deletion, edge deletion, and edge contraction. For graph G , any graph that can be obtained from G by using these three types of modifications is called a *minor* of G . If only the first two types of modification operations are allowed then resulting graph is said to a *subgraph* of G . If the only third type of modification is allowed then the resulting graph is called a *contraction* of G .

For two positive integer r, q , the $(r \times q)$ -grid is a graph in which every vertex is assigned a unique pair of the form (i, j) for $1 \leq i \leq r$ and $1 \leq j \leq q$. A pair of vertices (i_1, j_1) and



© Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 34; pp. 34:1–34:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(i_2, j_2) are adjacent with each other if and only if $|i_1 - i_2| + |j_1 - j_2| = 1$. There has been considerable attention to the problem of obtaining a grid as a minor of the given graph. We find it surprising that the very closely related question of obtaining a grid as a contraction did not receive any attention. In this article, we initiate a study of this problem from the parameterized complexity point of view.

The *contraction* of edge uv in simple graph G deletes vertices u and v from G , and replaces them by a new vertex, which is made adjacent to vertices that were adjacent to either u or v . Note that the resulting graph does not contain self-loops and multiple edges. A graph G is said to be *contractible* to graph H if H can be obtained from G by edge contractions. Equivalently, G is contractible to H if $V(G)$ can be partitioned into $|V(H)|$ many connected sets, called witness sets, and these sets can be mapped to vertices in H such that adjacency between witness sets is consistent with their mapped vertices in H . If such a partition of $V(G)$ exists then we call it H -witness structure of G . A graph G is said to be k -contractible to H if H can be obtained from G by k edge contractions. For a family of graphs \mathcal{G} , the \mathcal{G} -CONTRACTION problem takes as an input a graph G and an integer k , and the objective is to decide if G is k -contractible to a graph H in \mathcal{G} .

Related Work. Early papers of Watanabe et al. [20, 21], Asano and Hirata [3] showed \mathcal{G} -CONTRACTION is NP-Complete for various class of graphs like planar graphs, outer-planar graphs, series-parallel graphs, forests, chordal graphs. Brouwer and Veldman proved that it is NP-Complete even to determine whether a given graph can be contracted to a path of length four or not [5]. In the realm of parameterized complexity, \mathcal{G} -CONTRACTION has been studied with the parameter being the number of edges allowed to be contracted. It is known that \mathcal{G} -CONTRACTION admits an FPT algorithm when \mathcal{G} is set of paths [15], trees [15], cactus [17], cliques [6], planar graphs [12] and bipartite graphs [14, 13]. For a fixed integer d , let $\mathcal{H}_{\geq d}$, $\mathcal{H}_{\leq d}$ and $\mathcal{H}_{=d}$ denote the set of graphs with minimum degree at least d , maximum degree at most d , and d -regular graphs, respectively. Golovach et al. [11] and Belmonte et al. [4] proved that \mathcal{G} -CONTRACTION admits an FPT algorithm when $\mathcal{G} \in \{\mathcal{H}_{\geq d}, \mathcal{H}_{\leq d}, \mathcal{H}_{=d}\}$. When \mathcal{G} is split graphs or chordal graphs, the \mathcal{G} -CONTRACTION is known to be W[1]-hard [2] and W[2]-hard [18, 6], respectively. To the best of our knowledge, it is known that \mathcal{G} -CONTRACTION admits a polynomial kernel only when \mathcal{G} is a set of paths [15] or set of paths or cycle i.e. $\mathcal{H}_{\leq 2}$ [4]. It is known that \mathcal{G} does not admit a polynomial kernel, under standard complexity assumptions, when \mathcal{G} is set of trees [15], cactus [16], or cliques [6].

Our Contribution. In this article we study parameterized complexity of GRID CONTRACTION problem. We define the problem as follows.

GRID CONTRACTION	Parameter: k
Input: Graph G and integer k	
Question: Is G k -contractible to a grid?	

To the best of our knowledge, the computation complexity of the problem is not known nor it is implied by the existing results regarding edge contraction problems. We prove that the problem is indeed NP-Complete (Theorem 21). We prove that there exists an FPT algorithm which given an instance (G, k) of GRID CONTRACTION runs in time $4^{6k} \cdot |V(G)|^{\mathcal{O}(1)}$ and correctly concludes whether it is a YES instance or not (Theorem 20). We complement this result by proving that unless ETH fails there is no algorithm for GRID CONTRACTION with running time $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ (Theorem 21). We present a polynomial kernel with $\mathcal{O}(k^4)$ vertices and edges for GRID CONTRACTION (Theorem 27).

Our Methods. Our FPT algorithm for GRID CONTRACTION is divided into two phases. In the first phase, we introduce a restricted version of GRID CONTRACTION problem called BOUNDED GRID CONTRACTION. In this problem, along with a graph G and an integer k , an input consists of an additional integer r . The objective is to determine whether graph G can be k -contracted to a grid with r rows. We present an FPT algorithm parameterized by $(k+r)$ for this problem. This algorithm is inspired by the exact exponential algorithm for PATH CONTRACTION in [1]. It is easy to see that an instance (G, k) is a YES instance of GRID CONTRACTION if and only if (G, k, r) is a YES instance of BOUNDED GRID CONTRACTION for some r in $\{1, 2, \dots, |V(G)|\}$. In the second phase, given an instance (G, k) of GRID CONTRACTION we produce polynomially many instances of BOUNDED GRID CONTRACTION such that – (a) the input instance is a YES instance if and only if at least one of the produced instances is a YES instance and (b) for any produced instance, say (G', k', r) , we have $k' = k$ and $r \in \{1, 2, \dots, 2k + 5\}$. We prove that all these instances can be produced in time polynomial in the size of the input. An FPT algorithm for GRID CONTRACTION is a direct consequence of these two results. We use techniques presented in the second phase to obtain a polynomial kernel for GRID CONTRACTION.

We present a brief overview of the FPT algorithm for BOUNDED GRID CONTRACTION. *Boundary vertices* of a subset S of $V(G)$ are the vertices in S which are adjacent to at least one vertex in $V(G) \setminus S$. A subset S of $V(G)$ is *nice* if both $G[S], G - S$ are connected, and $G[S]$ can be contracted to a $(r \times q)$ -grid with all boundary vertices in S in an end-column for some integer q . In other words, a subset S of $V(G)$ is nice if it is a union of witness sets appearing in first few columns in some grid witness structure of G . See Definition 9. The objective is to keep building a *special partial solution* for some nice subsets. In this special partial solution, all boundary vertices of a particular nice subset are contained in bags appearing in an end-column. This partial solution is then extended to the remaining graph. The central idea is – *for a nice subset S of graph G , if $G[S]$ can be contracted to a grid such that all boundary vertices of S are in an end bag then how one contract $G[S]$ is irrelevant.* This allows us to store one solution for $G[S]$ and build a dynamic programming table nice subsets of vertices. The running time of such an algorithm depends on the following two quantities (i) the number of possible entries in the dynamic programming table, and (ii) time spent at each entry. We prove that *to bound both these quantities as a function of k , it is sufficient to know the size of neighborhood of S and the size of the union of witness sets in an end-column in a grid contraction of $G[S]$ which contains all boundary vertices of S .*

In the second phase, we first check whether a given graph G can be k -contracted to a grid with r rows for $r \in \{1, 2, \dots, 2k + 5\}$ using the algorithm mentioned in the previous paragraph. If for any value of r it returns YES then we can conclude that (G, k) is a YES instance of GRID CONTRACTION. Otherwise, we argue that there exists a *special separator* S in G which induces a $(2 \times q)$ grid for some positive integer p . We prove that it is safe to contract q vertical edges in $G[S]$. Let G' be the graph obtained from G by contracting these parallel edges. Formally, we argue that G is k -contractible to a $(r' \times q)$ -grid if and only if G' is k -contractible to $((r' - 1) \times q)$ -grid. We keep repeating the process of finding a special separator and contracting parallel edges in it until one of the following things happens – (a) The resultant graph is k -contractible to a $(r' \times q)$ -grid for some $r' < 2k + 5$. (b) The resultant graph does not contain a special separator. We argue that in Case (b), it is safe to conclude that (G, k) is a NO instance for GRID CONTRACTION.

Organization of the paper. We present some preliminary notations which will be used in rest of the paper in Section 2. We present a crucial combinatorial lemma in Section 3. As mentioned earlier, this algorithm is divided into two phases. We present the first and the

second phase in Section 4 and 5, respectively. Section 5 also contains an FPT algorithm for GRID CONTRACTION. We prove that the dependency on the parameter in the running time of this algorithm is optimal, up to a constant factor, unless ETH fails in Section 6. In Section 7, we present a polynomial kernel for GRID CONTRACTION problem.

Due to space constraints we have omitted the proofs of the statements marked with (\star) . We present them in a full version of the paper.

2 Preliminaries

For a positive integer k , $[k]$ denotes the set $\{1, 2, \dots, k\}$.

2.1 Graph Theory

In this article, we consider simple graphs with a finite number of vertices. For an undirected graph G , sets $V(G)$ and $E(G)$ denote its set of vertices and edges respectively. Two vertices u, v in $V(G)$ are said to be *adjacent* if there is an edge uv in $E(G)$. The neighborhood of a vertex v , denoted by $N_G(v)$, is the set of vertices adjacent to v and its degree $d_G(v)$ is $|N_G(v)|$. The subscript in the notation for neighborhood and degree is omitted if the graph under consideration is clear. For a set of edges F , set $V(F)$ denotes the collection of endpoints of edges in F . For a subset S of $V(G)$, we denote the graph obtained by deleting S from G by $G - S$ and the subgraph of G induced on the set S by $G[S]$. For two subsets S_1, S_2 of $V(G)$, we say S_1, S_2 are adjacent if there exists an edge with one endpoint in S_1 and other in S_2 . For a subset S of $V(G)$, let $\Phi(S)$ denotes set of vertices in S which are adjacent with at least one vertex outside S . Formally, $\Phi(S) = \{s \in S \mid N(s) \setminus S \neq \emptyset\}$. These are also called *boundary vertices* of S .

A *path* $P = (v_1, \dots, v_l)$ is a sequence of distinct vertices where every consecutive pair of vertices is adjacent. For two vertices v_1, v_2 in G , $dist(v_1, v_2)$ denotes the length of a shortest path between these two vertices. A graph is called *connected* if there is a path between every pair of distinct vertices. It is called *disconnected* otherwise. A set S of $V(G)$ is said to be a *connected set* if $G[S]$ is connected. For two vertices v_1, v_2 in G , a set S is called (v_1-v_2) -*separator*, if any v_1-v_2 paths intersects S . If a set is a (v_1-v_2) -separator as well as (v_3-v_4) -separator then we write it as $\{(v_1-v_2), (v_3-v_4)\}$ -separator.

For two positive integer r, q , the $(r \times q)$ -grid is a graph on $r \cdot q$ vertices. The vertex set of this graph consists of all pairs of the form (i, j) for $1 \leq i \leq r$ and $1 \leq j \leq q$. A pair of vertices (i_1, j_1) and (i_2, j_2) are adjacent with each other if and only if $|i_1 - i_2| + |j_1 - j_2| = 1$. We say that such graph is a grid with r rows and q columns. It is called a $(r \times q)$ -grid and is denoted by $\boxplus_{r \times q}$. We use \boxplus to denote a grid with unspecified number of rows and columns. The vertices in grid \boxplus are denoted by $\boxplus[i, j]$ or simply by $[i, j]$. Note that the grid with exactly one row is a path. To remove some corner cases, we consider grids that have at least two rows and two columns. Any grid contains exactly four vertices that have degree two. These vertices are called *corner vertices*. Let $t_1 = [1, 1]$, $t_2 = [1, q]$, $t_3 = [r, q]$, and $t_4 = [r, 1]$ be the corner vertices in grid $\boxplus_{r \times q}$.

► **Observation 2.1** (\star) . *If \hat{S} is a connected $\{(t_1-t_4), (t_2-t_3)\}$ -separator in $\boxplus_{r \times q}$ then its size is at least q . Moreover, if $|\hat{S}| = q$ then it corresponds to a row in $\boxplus_{r \times q}$.*

2.2 Graph Contraction

The *contraction* of edge uv in G deletes vertices u and v from G , and adds a new vertex, which is made adjacent to vertices that were adjacent to either u or v . Notice that no self-loop or parallel edge is introduced in this process. The resulting graph is denoted

by G/e . For a given graph G and edge $e = uv$, we formally define G/e in the following way: $V(G/e) = (V(G) \cup \{w\}) \setminus \{u, v\}$ and $E(G/e) = \{xy \mid x, y \in V(G) \setminus \{u, v\}, xy \in E(G)\} \cup \{wx \mid x \in N_G(u) \cup N_G(v)\}$. Here, w is a new vertex which was not in $V(G)$. Note that an edge contraction reduces the number of vertices in a graph by exactly one. Several edges might disappear due to one edge contraction. For a subset of edges F in G , graph G/F denotes the graph obtained from G by contracting each connected component in the sub-graph $G' = (V(F), F)$ to a vertex.

► **Definition 1** (Graph Contraction). *A graph G is said to be contractible to graph H if there exists an onto function $\psi : V(G) \rightarrow V(H)$ such that following properties hold.*

- For any vertex h in $V(H)$, graph $G[W(h)]$ is connected and not empty, where set $W(h) := \{v \in V(G) \mid \psi(v) = h\}$.
- For any two vertices h, h' in $V(H)$, edge hh' is present in H if and only if there exists an edge in G with one endpoint in $W(h)$ and another in $W(h')$.

We say graph G is contractible to H via mapping ψ . For a vertex h in H , set $W(h)$ is called a *witness set* associated with/corresponding to h . We define H -*witness structure* of G , denoted by \mathcal{W} , as collection of all witness set. Formally, $\mathcal{W} = \{W(h) \mid h \in V(H)\}$. A witness structure \mathcal{W} is a partition of vertices in G . If a *witness set* contains more than one vertex then we call it *big* witness-set, otherwise it is *small/singleton* witness set.

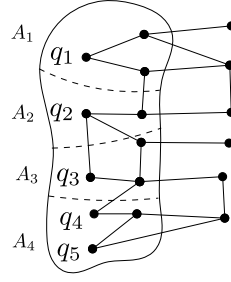
If graph G has a H -witness structure then graph H can be obtained from G by a series of edge contractions. For a fixed H -witness structure, let F be the union of spanning trees of all witness sets. By convention, the spanning tree of a singleton set is an empty set. To obtain graph H from G , it is necessary and sufficient to contract edges in F . We say graph G is k -*contractible* to H if cardinality of F is at most k . In other words, H can be obtained from G by at most k edge contractions. The following observations are immediate consequences of definitions.

► **Observation 2.2** (*). *If graph G is k -contractible to graph H via mapping ψ then following statements are true.*

1. $|V(G)| \leq |V(H)| + k$.
2. Any H -witness structure of G has at most k big witness sets.
3. For a fixed H -witness structure, the number of vertices in G which are contained in big witness sets is at most $2k$.
4. If S is a $(x_1 - x_2)$ -separator in G then $\psi(S)$ is a $(\psi(s_1) - \psi(s_2))$ -separator in H .
5. If S is a separator in G such that there are at least two connected components of $G \setminus S$ which has at least $k + 1$ vertices, then $\psi(S)$ is a separator in H .

2.3 Parameterized Complexity

An instance of a parameterized problem comprises of an input I , which is an input of the classical instance of the problem and an integer k , which is called as the parameter. A problem Π is said to be *fixed-parameter tractable* or in FPT if given an instance (I, k) of Π , we can decide whether or not (I, k) is a YES instance of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$. Here, $f(\cdot)$ is some computable function whose value depends only on k . We say that two instances, (I, k) and (I', k') , of a parameterized problem Π are *equivalent* if $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$. A *reduction rule*, for a parameterized problem Π is an algorithm that takes an instance (I, k) of Π as input and outputs an instance (I', k') of Π in time polynomial in $|I|$ and k . If (I, k) and (I', k') are equivalent instances then we say the reduction rule is *safe*. A parameterized problem Π admits a kernel of size $g(k)$ (or $g(k)$ -kernel) if there is a polynomial



■ **Figure 1** An example of a 4-slab. See Definition 2. For $Q = \{q_1, q_2, q_3, q_4, q_5\}$ and its partition $\mathcal{P}_4(Q) = \{\{q_1\}, \{q_2\}, \{q_3\}, \{q_4, q_5\}\}$, A is an $(\mathcal{P}_4(Q), \alpha, \beta)$ -4-slab.

time algorithm (called *kernelization algorithm*) which takes as an input (I, k) , and in time $|I|^{\mathcal{O}(1)}$ returns an equivalent instance (I', k') of Π such that $|I'| + k' \leq g(k)$. Here, $g(\cdot)$ is a computable function whose value depends only on k . For more details on parameterized complexity, we refer the reader to the books of Downey and Fellows [8], Flum and Grohe [9], Niedermeier [19], and the more recent books by Cygan et al. [7] and Fomin et al. [10].

3 Combinatorial Lemma

We introduce the notion of r -slabs which can be thought of as connected components with special properties. A r -slab is a connected set which can be partitioned into r connected subsets such that the adjacency between these parts and their neighbourhood follows certain pattern. For an integer r and a set A , an ordered r -partition is a list of subsets of A whose union is A . We define r -slab as follows.

► **Definition 2 (r -Slab).** A r -slab in G is an ordered r -partition of a connected set A , say A_1, A_2, \dots, A_r , which satisfy following conditions.

- For every i in $[r]$, set A_i is a non-empty set and $G[A_i]$ is connected.
- For $i \neq j$ in $[r]$, sets A_i, A_j are adjacent if and only if $|i - j| = 1$.
- For every i in $[r]$, define $B_i = N(A_i) \setminus A$. For $i \neq j$ in $[r]$, sets B_i, B_j are mutually disjoint and if B_i and B_j are adjacent then $|i - j| = 1$.

We denote a r -slab by $\langle A_1, A_2, \dots, A_r \rangle$. For a r -slab $\langle A_1, A_2, \dots, A_r \rangle$, set A denotes union of all A_i s. We note that every connected subset of G is an 1-slab.

For positive integers α, β , a connected set A in graph G is called an (α, β) -connected set if $|A| \leq \alpha$ and $|N(A)| \leq \beta$. For a non-empty set $Q \subseteq V(G)$ a connected set A in G is a (Q) -connected set if $Q \subseteq A$. We generalize these notations for r -slab as follows.

► **Definition 3 ((α, β) - r -slab).** For a graph G and integers α, β , a r -slab $\langle A_1, A_2, \dots, A_r \rangle$ is said to be an (α, β) - r -slab if $|A| \leq \alpha$ and $|N(A)| \leq \beta$.

For a set Q , let $\mathcal{P}_r(Q) = \{Q_1, Q_2, \dots, Q_r\}$ denotes its ordered r -partition. An ordered r -partition is said to be *valid* if for any two vertices $u \in Q_i$ and $v \in Q_j$, u, v are adjacent implies $|i - j| \leq 1$.

► **Definition 4 ($\mathcal{P}_r(Q)$ - r -slab).** For a graph G , a subset Q of $V(G)$ and its ordered valid partition $\mathcal{P}_r(Q) = \{Q_1, Q_2, \dots, Q_r\}$, a r -slab $\langle A_1, A_2, \dots, A_r \rangle$ in G is said to be a $\mathcal{P}_r(Q)$ - r -slab if Q_i is a subset of A_i for every i in $[r]$.

See Figure 1 for an example. We combine properties mentioned in previous two definitions to define specific types of r -slabs.

► **Definition 5** ($(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab). For a graph G , a non-empty subset Q of $V(G)$, its ordered valid partition $\mathcal{P}_r(Q) = \{Q_1, Q_2, \dots, Q_r\}$, and integers α, β , a r -slab $\langle A_1, A_2, \dots, A_r \rangle$ in G is a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab if it is an (α, β) - r -slab as well as a $\mathcal{P}_r(Q)$ - r -slab.

We mention following two observations which are direct consequences of the definition.

► **Observation 3.1.** Let $\langle A_1, A_2, \dots, A_r \rangle$ be a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab in graph G . If a vertex v is in $N(A)$ then $\langle A_1, A_2, \dots, A_r \rangle$ is a $(\mathcal{P}_r(Q), \alpha, \beta - 1)$ - r -slab in graph $G - \{v\}$.

For a graph G , consider a vertex v and let $G' = G - \{v\}$. For a non-empty subset Q' of $V(G')$, its ordered partition $\mathcal{P}_r(Q') = \{Q'_1, Q'_2, \dots, Q'_r\}$, and integers α, β , let $\langle A'_1, A'_2, \dots, A'_r \rangle$ be a $(\mathcal{P}_r(Q'), \alpha, \beta)$ - r -slab in G' .

► **Observation 3.2.** If vertex v satisfy following two properties then $\langle A'_1, A'_2, \dots, A'_r \rangle$ is a $(\mathcal{P}_r(Q'), \alpha, \beta + 1)$ - r -slab in G .

- Vertex v is adjacent with exactly one part, say A'_i , of the r -slab
- For any vertex u in $N'_G(A'_j) \setminus A'$, if u and v are adjacent in G then $|i - j| \leq 1$.

Definition 5 generalizes the notation of (Q, α, β) -connected set defined in [1]. In the same paper, authors proved that there is an algorithm that given a graph G on n vertices, a non-empty set $Q \subseteq V(G)$, and integers α, β , enumerates all (Q, α, β) -connected sets in G in time $2^{\alpha - |Q| + \beta} \cdot n^{\mathcal{O}(1)}$. We present similar combinatorial lemma for $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs.

► **Lemma 6.** There is an algorithm that given a graph G on n vertices, a non-empty set $Q \subseteq V(G)$, its ordered partition $\mathcal{P}_r(Q) = \{Q_1, Q_2, \dots, Q_r\}$, and integers α, β , enumerates all $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G in time $4^{\alpha - |Q| + \beta} \cdot n^{\mathcal{O}(1)}$.

Proof. Let $N(Q) = \{v_1, v_2, \dots, v_p\}$. Arbitrarily fix a vertex v_l in $N(Q)$. We partition $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G based on whether v_l is contained in it or not. In later case, such $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab is also a $(\mathcal{P}_r(Q), \alpha, \beta - 1)$ - r -slab in $G - \{v\}$. We now consider the first case. Let i be the smallest integer in $[r]$ such that v_l is adjacent with Q_i . Note that, by definition, if v_l is present in a $\mathcal{P}_r(Q)$ - r -slab then it can be part of either A_{i-1} , A_i or A_{i+1} . We encode this fact by moving v_l to either Q_{i-1} , Q_i or Q_{i+1} . Let $\mathcal{P}_r^{i-1}(Q \cup \{v_l\})$, $\mathcal{P}_r^i(Q \cup \{v_l\})$ and $\mathcal{P}_r^{i+1}(Q \cup \{v_l\})$ be r -partitions of $Q \cup \{v_l\}$ obtained from $\mathcal{P}_r(Q)$ by adding v_l to set Q_{i-1} , Q_i and Q_{i+1} , respectively. Formally, these three sets are defined as follows.

- $\mathcal{P}_r^{i-1}(Q \cup \{v_l\}) := \{Q_1, \dots, Q_{i-1} \cup \{v_l\}, Q_i, Q_{i+1}, \dots, Q_r\}$
- $\mathcal{P}_r^i(Q \cup \{v_l\}) := \{Q_1, \dots, Q_{i-1}, Q_i \cup \{v_l\}, Q_{i+1}, \dots, Q_r\}$
- $\mathcal{P}_r^{i+1}(Q \cup \{v_l\}) := \{Q_1, \dots, Q_{i-1}, Q_i, Q_{i+1} \cup \{v_l\}, \dots, Q_r\}$

Algorithm. We present a recursive enumeration algorithm which takes $(G, \mathcal{P}_r(Q), \alpha, \beta)$ as an input and outputs a set, say \mathcal{A} , of all $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab in G . The algorithm initializes \mathcal{A} to an empty set. The algorithm returns \mathcal{A} if one of the following statements is true: (i) $\mathcal{P}_r(Q)$ is not a valid partition of Q , (ii) $\alpha - |Q| < 0$ or $\beta < 0$, (iii) there is a vertex v_l in $N(Q)$ which is adjacent with Q_i and Q_j for some i, j in $[r]$ such that $|i - j| \geq 2$. If $\alpha - |Q| + \beta = 0$, the the algorithm checks if $\mathcal{P}_r(Q)$ is a $(\mathcal{P}_r(Q), 0, 0)$ - r -slabs in G . If it is the case then the algorithm returns singleton set containing $\mathcal{P}_r(Q)$ otherwise it returns an empty set. If there is a vertex v_l in $N(Q)$ which is adjacent with Q_{i-1} , Q_i and Q_{i+1} for some i in $[r]$ then the algorithm calls itself on instance $(G, \mathcal{P}_r^i(Q \cup \{v_l\}), \alpha, \beta)$ where $\mathcal{P}_r^i(Q \cup \{v_l\})$ is r -partition as defined above. It returns the set obtained on this recursive call as the output. If there

are no such vertices in $N(Q)$, then for some $l \in \{1, \dots, |N(Q)|\}$, the algorithm creates four instances viz $(G - \{v_l\}, \mathcal{P}_r(Q), \alpha, \beta - 1)$ and $(G, \mathcal{P}_r^{i_0}(Q \cup \{v_l\}), \alpha, \beta)$ for $i_0 \in \{i - 1, i, i + 1\}$. The algorithm calls itself recursively on these four instances. Let $\mathcal{A}_l^v, \mathcal{A}_l^{i-1}, \mathcal{A}_l^i$, and \mathcal{A}_{i+1} be the set returned, respectively, by the recursive call of the algorithm. The algorithm adds all elements in $\mathcal{A}_l^{i-1} \cup \mathcal{A}_l^i \cup \mathcal{A}_l^{i+1}$ to \mathcal{A} . For every $(\mathcal{P}_r(Q), \alpha, \beta - 1)$ - r -slabs $\langle A'_1, A'_2, \dots, A'_r \rangle$ in \mathcal{A}_l^v , the algorithm checks whether it is a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G using Observation 3.2. If it is indeed a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G then it adds it to \mathcal{A} . The algorithm returns \mathcal{A} at the end of this process.

We now argue the correctness of the algorithm. For every input instance $(G, \mathcal{P}_r(Q), \alpha, \beta)$ we define its measure as $\mu((G, \mathcal{P}_r(Q), \alpha, \beta)) = \alpha - |Q| + \beta$. We proceed by the induction hypothesis that the algorithm is correct on any input whose measure is strictly less than $\alpha - |Q| + \beta$. Consider the base cases $\alpha - |Q| + \beta = 0$. In this case, the only possible $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab is $\mathcal{P}_r(Q)$. The algorithm checks this and returns the correct answer accordingly. We consider the case when $\alpha - |Q| + \beta \geq 1$. Every $(\mathcal{P}_r(Q \cup \{v_l\}), \alpha, \beta)$ - r -slab is also a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab. The algorithm adds a r -slab in \mathcal{A}_l^v to \mathcal{A} only if it is a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G . Hence the algorithm returns a set of $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G . In remaining part we argue that every $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs is enumerated by the algorithm.

By Definition 2, no vertex in closed neighbourhood of a r -slab can be adjacent to two non-adjacent parts of a r -slab. Hence, if there is a vertex v_l in $N(Q)$ which is adjacent with Q_i and Q_j for some i, j in $[r]$ such that $|i - j| \geq 2$ then the algorithm correctly returns an empty set. Suppose there exists a vertex v in $N(Q)$ which is adjacent with Q_{i-1}, Q_i and Q_{i+1} for some i in $[r]$. By Definition 2, any r -slab containing $\mathcal{P}_r(Q)$ must contains v in it. In this case, the number of $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab is same as the number of $(\mathcal{P}_r^i(Q \cup \{v\}), \alpha, \beta)$ - r -slab where $\mathcal{P}_r^i(Q \cup \{v\})$ is the r -partition of $Q \cup \{v\}$ obtained from $\mathcal{P}_r(Q)$ by adding v to Q_i . The measure for input instance $(G, \mathcal{P}_r^i(Q \cup \{v\}), \alpha, \beta)$ is strictly smaller than $\alpha - |Q| + \beta$. Hence by induction hypothesis, the algorithm correctly computes all $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab.

Consider the case when there is no vertex which is adjacent with Q_{i-1}, Q_i and Q_{i+1} for any i in $[r]$. Let v_l be a vertex in $N(Q)$ and there is an integer i in $[p]$ such that i is the smallest integer, and v_l is adjacent with Q_i . As mentioned earlier, either v_l is a part of $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab or not. In first case, by Definition 2, v_l can be part of A_{i-1}, A_i or A_{i+1} in any $\mathcal{P}_r(Q)$ - r -slab. The measure of input instance $(G, \mathcal{P}_r^{i_0}(Q \cup \{v\}), \alpha, \beta)$ is $\alpha - |Q| + \beta - 1$. Hence by induction hypothesis, the algorithm correctly enumerates all $(\mathcal{P}_r^{i_0}(Q \cup \{v\}), \alpha, \beta)$ - r -slabs in G_l . Consider a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab $\langle A_1, A_2, \dots, A_r \rangle$ in G which does not contain v_l . By Observation 3.1, $\langle A_1, A_2, \dots, A_r \rangle$ is a $(\mathcal{P}_r(Q), \alpha, \beta - 1)$ - r -slab in $G - \{v\}$. By induction hypothesis, the algorithm correctly computes all $(\mathcal{P}_r(Q), \alpha, \beta - 1)$ - r -slabs in $G - \{v\}$. Since $\langle A_1, A_2, \dots, A_r \rangle$ is a $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slab in G , vertex v_l satisfy both the properties mentioned in Observation 3.2. Hence algorithm adds $\langle A_1, A_2, \dots, A_r \rangle$ to the set \mathcal{A}_l . Hence, we can conclude that the algorithm correctly enumerates all $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in G .

Using the induction hypothesis that the algorithm correctly outputs the set of all $(\mathcal{P}_r(Q), \alpha, \beta)$ - r -slabs in time $4^{\alpha - |Q| + \beta} \cdot n^{\mathcal{O}(1)}$, the running time of the algorithm follows. This concludes the proof of the lemma. \blacktriangleleft

We use following corollary of Lemma 6.

► Corollary 7. *There is an algorithm that given a graph G on n vertices and integers α, β , enumerates all (α, β) - r -slab in G in time $4^{\alpha + \beta} \cdot n^{\mathcal{O}(1)}$.*

4 An FPT algorithm for Bounded Grid Contraction

In this section, we present an FPT algorithm for BOUNDED GRID CONTRACTION. We formally define the problem as follows.

BOUNDED GRID CONTRACTION Input: Graph G and integers k, r Question: Is G k -contractible to a grid with r rows?	Parameter: k, r
---	--------------------------

We start with a definition of nice subsets mentioned in the Introduction section. As mentioned before, vertices of a nice subset corresponds to witness sets in first few columns of a grid-witness structure of the input graph. Hence boundary vertices of a nice set corresponds to witness sets in some column of a grid. Note that we are interested in the grids that have exactly r -rows. Hence, we use the notation of r -slab defined in previous section to formally define nice sets. Consider a r -slab $\langle D_1, D_2, \dots, D_r \rangle$ which corresponds to a column in some grid that can be obtained from the input graph with at most k edge contraction. By Observation 2.2, an edge contraction reduces the number of vertices by exactly one. As there are $3r$ many vertices in three adjacent rows in a grid, the size of closed neighborhood of D in G is at most $k + 3r$. Thus, we can focus our attention on r -slabs with bounded closed neighborhood. We define k -potential r -slabs as follows.

► **Definition 8** (*k -Potential r -Slab*). For a given graph G and integers k, r , a r -slab $\langle D_1, D_2, \dots, D_r \rangle$ is said to be a k -potential r -slab of G if it satisfies following two conditions:

- $|D| + |N(D)| \leq k + 3r$; and
- $G - D$ has at most two connected components.

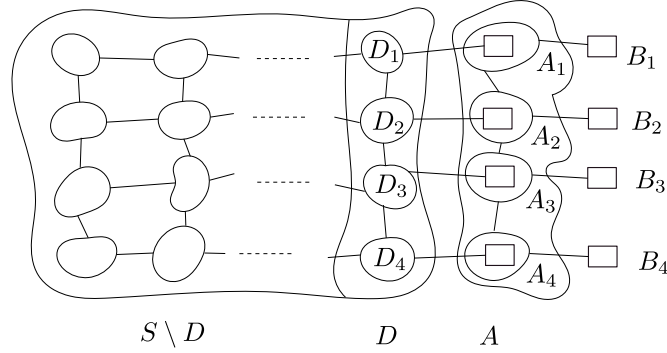
Here, $D = D_1 \cup D_2 \cup \dots \cup D_r$.

► **Definition 9** (*Nice Subset*). A subset S of $V(G)$ is said to be a nice subset of G if there exists a k -potential r -slab, say $\langle D_1, D_2, \dots, D_r \rangle$, such that D is a subset of S and $G[S \setminus D]$ is one of the connected components of $G - D$. We say that r -slab $\langle D_1, D_2, \dots, D_r \rangle$ is responsible for nice subset S .

Since $\langle D_1, D_2, \dots, D_r \rangle$ is a k -potential r -slab, both $G[S]$ and $G - S$ are connected. There may be more than one k -potential r -slabs responsible for a nice subset. We define a pair of nice sets and k -potential r -slabs responsible for it.

► **Definition 10** (*Valid Tuple*). A tuple $(S, \mathcal{P}_r(D))$ is called a valid tuple if S is a nice subset and $\mathcal{P}_r(D) \equiv \langle D_1, D_2, \dots, D_r \rangle$ is a k -potential r -slab responsible for it.

Let \mathcal{V}_k be the set of all valid tuples. For a valid tuple $(S, \mathcal{P}_r(D))$ in \mathcal{V}_k , we define a collection of k -potential r -slabs which is denoted by $\mathcal{A}[(S, \mathcal{P}_r(D))]$. This set can be thought of as a collection of “potential column extenders” for S . See Figure 2. In other words, we can append a k -potential- r -slab in $\mathcal{A}[(S, \mathcal{P}_r(D))]$ to get a grid witness structure of a larger graphs containing S . Let $\mathcal{P}_r(A)$ be a k -potential- r -slab in $\mathcal{A}[(S, \mathcal{P}_r(D))]$. Intuitively speaking, $\mathcal{P}_r(A)$ is the “new” column to be “appended” to a grid witness structure of $G[S]$, to obtain a grid witness structure for $G[S \cup A]$. Hence if $G[S]$ can be k' -contracted to a grid then $G[S \cup A]$ can be $k' + (|A| - r)$ -contracted to a grid. For improved analysis, we concentrate on subset $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$ of $\mathcal{A}[(S, \mathcal{P}_r(D))]$ defined for integers a, b . The set $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$ is a collection of k -potential r -slabs of size at most a which have at most b neighbors outside S . We impose additional condition that $a + b + |D|$ is at most $k + 3r$ for improved analysis. Formally, $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))] = \{ \langle A_1, A_2, \dots, A_r \rangle \mid |A| \leq a, |N(A) \setminus S| \leq b, \text{ where } A = A_1 \cup A_2 \cup \dots \cup A_r \text{ and for every } D_i \text{ in } \mathcal{P}_r(D), (N(D_i) \setminus S) \subseteq A_i, \text{ and } a + b + |D| \leq k + 3r \}$.



■ **Figure 2** All sets with smooth (non-rectangular) boundary are connected. Set A is a possible extension of nice subset S . In other words, A is an element in $\mathcal{A}_{|A|,|B|}[(S, \mathcal{P}_r(D))]$. See paragraph before Lemma 11.

Algorithm. The algorithm takes a graph G on n vertices and integers k, r as input and outputs either **True** or **False**. The algorithm constructs a dynamic programming table Γ in which there is an entry corresponding to every index $[(S, \mathcal{P}_r(D)); k']$ where $(S, \mathcal{P}_r(D))$ is a valid tuple in \mathcal{V}_k and k' is an integer in $\{0\} \cup [k]$. It initialize values corresponding to all entries to **False**.

(*for-loop Initialization*) For a tuple $(S, \mathcal{P}_r(D)) \in \mathcal{V}_k$ such that $S = D$ and $k' \geq |S| - r = |D| - r$, the algorithm sets $\Gamma[(S, \mathcal{P}_r(D)); k'] = \mathbf{True}$.

(*for-loop Table*) The algorithm processes indices in the table in chronologically increasing order. It first checks the size of S , then the size of D , followed by k . Ties are broken arbitrarily. At table index $[(S, \mathcal{P}_r(D)); k']$, if $\Gamma[(S, \mathcal{P}_r(D)); k']$ is **False** then the algorithm continues to next tuple. If $\Gamma[(S, \mathcal{P}_r(D)); k']$ is **True** then it runs the following for-loop at this index.

(*for-loop at Index*) The algorithm computes the set $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$ for every pair of integers $a (\geq r), b (\geq 0)$ which satisfy following properties (1) $a + b + |D| \leq k + 3r$, (2) $k' + a - r \leq k$, and (3) $|N(S)| \leq a$. For every k -potential r -slab $\mathcal{P}_r(A)$ in $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$, the algorithm sets $\Gamma[(S \cup A, \mathcal{P}_r(A)); k_1]$ to **True** for every $k_1 \geq k' + (a - r)$.

If $\Gamma[(V(G), \mathcal{P}_r(D)); k']$ is set to **True** for some $\mathcal{P}_r(D)$ and k' then the algorithm returns **True** otherwise it returns **False**. This completes the description of the algorithm.

Recall that for a given connected subset S of $V(G)$, $\Phi(S)$ denotes its boundary vertices i.e. set of vertices in S which are adjacent with at least one vertex outside S .

► **Lemma 11.** For every tuple $(S, \mathcal{P}_r(D))$ in \mathcal{V}_k and integer k' in $\{0\} \cup [k]$, the algorithm assign $\Gamma[(S, \mathcal{P}_r(D)); k'] = \mathbf{True}$ if and only if $k' + |N(S)| - r \leq k$ and there is a $(r \times q)$ -grid witness structure of $G[S]$, for some integer q , such that $\mathcal{P}_r(D)$ is collection of witness sets in an end-column and $\Phi(S)$ is in D .

Proof. We prove the lemma by induction on $|S| + k'$ for indices $((S, \mathcal{P}_r(D)); k')$ in the dynamic programming table. For the induction hypothesis, we assume that for a positive integer z the algorithm computes $\Gamma[(S, \mathcal{P}_r(D)); k']$ correctly for each $(S, \mathcal{P}_r(D))$ in \mathcal{V}_k and k' in $0 \cup [k]$ for which $|S| + k' \leq z$.

Consider the base case when $|S| = |D| = r$ and $k' = 0$. Since $D \subseteq S$, we have $S = D$. This implies $\mathcal{P}_r(S) = \mathcal{P}_r(D)$ is a r -slab. Any connected subset of a graph can be contracted to a vertex by contracting a spanning tree. Hence, $G[S]$ can be contracted to a $(r \times 1)$ -grid

by contracting $|D| - r$ many edges. This implies that the values assigned by the algorithm in (*for-loop Initialization*) are correct. We note that once the algorithm sets a particular value to **True**, it does not change it afterwards.

Assuming induction hypothesis, we now argue that the computation of $\Gamma[\cdot]$ for indices of the form $[(S_1, \mathcal{P}_r(D_1)); k_1]$ where $|S_1| + k_1 = z + 1$ are correct. Note that if $[(S_1, \mathcal{P}_r(D_1)); k_1]$ is an entry in the table then $(S_1, \mathcal{P}_r(D_1))$ is a valid tuple in \mathcal{V}_k and k_1 is an integer in the set $\{0\} \cup [k]$.

(\Rightarrow) Assume that $G[S_1]$ is k_1 -contractible to a $(r \times q)$ -grid such that all vertices in $\Phi(S_1)$ are in an end-column $\mathcal{P}_r(D_1)$ and $k_1 + |N(S_1)| - r \leq k$. We argue that the algorithm sets $\Gamma[(S_1, \mathcal{P}_r(D_1)); k_1]$ to **True**. Let $G[S_1]$ be k_1 -contractible to a $(r \times q)$ -grid. If $q = 1$ then $D_1 = S_1$ and in this case algorithm correctly computes $\Gamma[(S_1, \mathcal{P}_r(D)); k_1]$. Consider the case when $q \geq 2$. Let $\mathcal{W} = \{W_{ij} \mid (i, j) \in [r] \times [q]\}$ be a $(r \times q)$ -grid structure of G such that $\mathcal{P}_r(D)$ is collection of witness sets in an end-column and $\Phi(S_1)$ is a subset of D . Define W_j^c as union of all witness sets in column j . Formally, $W_j^c = \bigcup_{i=1}^r W_{ij}$. Hence, $\mathcal{W} = W_1^c \cup W_2^c \cup \dots \cup W_{q-1}^c \cup W_q^c$ and $\mathcal{P}_r(D) = W_q^c$. Consider set $S_0 = W_1^c \cup W_2^c \cup \dots \cup W_{q-1}^c$. Since $q \geq 2$, S_0 is a non-empty set. Let $k_0 = k_1 - (|W_q^c| - r)$. We argue that $[(S_0, W_{q-1}^c); k_0]$ is an index in the table and $|S_0| + k_0 \leq z$. As \mathcal{W} is a k_1 -grid witness structure, $|W_q^c| - r \leq k_1$ and hence k_0 is a non-negative integer. Since $G[W_q^c]$ is a connected graph, $G - W_{q-1}^c$ has exactly two connected components viz $G[W_1^c \cup \dots \cup W_{q-2}^c]$ and the component containing W_q^c . As \mathcal{W} is a k_1 -grid witness structure, $|W_{q-2}^c| + |W_{q-1}^c| + |W_q^c| \leq k_1 + 3r \leq k + 3r$ and $N(W_{q-1}^c) \subseteq W_{q-2}^c \cup W_q^c$. (We note that W_{q-2}^c may not exist but this does not change the argument. For the sake of clarity, we do not consider this as separate case.) Since $|W_{q-1}^c| + |N(W_{q-1}^c)| \leq k + 3r$ and $G - W_{q-1}^c$ has at most two connected components, W_{q-1}^c is a k -potential r -slab. Note that $\langle W_{1j}, W_{2j}, \dots, W_{rj} \rangle$ is the r -partition of k -potential r -slab W_{q-1}^c . Hence (S_0, W_{q-1}^c) is a tuple in \mathcal{V}_k and $((S_0, W_{q-1}^c); k_0)$ is an index in the table. Since W_q^c is not an empty set, $|S_0| + k_0 \leq |S_1| - |W_q^c| + k_1 - (|W_q^c| - r) \leq z + 1 + r - 2|W_q^c|$ as $|S_1| + k_1 = z + 1$. Since $|W_q^c| \geq r \geq 1$, we conclude $|S_0| + k_0 \leq z$. Note that $\mathcal{W} \setminus \{W_q^c\}$ is a $(k_1 - |W_q^c| + r)$ -grid witness structure for $G[S_0]$. This implies that $G[S_0]$ is k_0 -contractible to a grid with W_{q-1}^c as collection of bags in an end-column and $k_0 + |N(S_0)| - r \leq k_1 \leq k$. Moreover, $S_0 = S_1 \setminus W_q^c$, $\Phi(S_0)$ is contained in W_{q-1}^c . By the induction hypothesis, the algorithm has correctly set $\Gamma[(S_0, W_{q-1}^c); k_0]$ to **True**. Let $x_0 = |W_{q-1}^c|$, $a = |W_q^c|$ and $b = |W_q^c \setminus N(S_0)| = |N(S_1)|$. We first claim that $x_0 + a + b \leq k + 3r$. Note that $|W_{q-1}^c| + |W_q^c| \leq k_1 + 2r$ and $k_1 + b \leq k + r$. Hence $|W_{q-1}^c| + |W_q^c| + b = x_0 + a + b \leq k + 3r$. At index $[(S_0, W_{q-1}^c); k_0]$, the algorithm computes $\mathcal{A}_{a,b}[(S_0, W_{q-1}^c)]$. Clearly, W_q^c is one of the sets in $\mathcal{A}_{a,b}[(S_0, W_{q-1}^c)]$ as for every i in $[r]$, $N(W_{i,q-1}) \setminus S_0$ is contained in W_{iq} and $G[W_{iq}]$ is a connected graph. Hence the algorithm sets $\Gamma[(S_1, W_q^c), k_1] = \Gamma[(S_1, \mathcal{P}_r(D)), k_1]$ to **True**.

(\Leftarrow) To prove other direction, we assume that the algorithm sets $\Gamma[(S_1, \mathcal{P}_r(A)); k_1]$ to **True**. We argue that $G[S_1]$ is k_1 -contractible to a grid such that $\mathcal{P}_r(A)$ is a collection of witness sets in an end-column in a witness structure; $\Phi(S_1)$ is in A ; and $k_1 + |N(S_1)| - r \leq k$. If $\Gamma[(S_1, \mathcal{P}_r(A)); k_1]$ is set to **True** in the (*for-loop Initialization*) then, as discussed in first paragraph, this is correct. Consider the case when the value at $\Gamma[(S_1, \mathcal{P}_r(A)); k_1]$ is set to **True** when the algorithm was processing at index $[(S_0, \mathcal{P}_r(D)); k_0]$. Note that value at $[(S_0, \mathcal{P}_r(D)); k_0]$ has been set **True** by the algorithm as otherwise it will not change any value while processing this index. Note that $|A| = a$ and $|N(S_1)| = b$. Since a is a positive integer and $k_0 + a - r \leq k_1$ (because (*for-loop at Index*) updates only for such values), we know $|S_0| + k_0 \leq |S_1| + k_1 - 2a + r = z + 1 - 2a + r$. Since $a \geq r \geq 1$, we get $|S_0| + k_0 \leq z$. By the induction hypothesis, algorithm has correctly computed value at $[(S_0, \mathcal{P}_r(D)); k_0]$. Hence $G[S_0]$ can be k_0 -contracted to a grid such that $\Phi(S_0)$ is in D and there exists a grid witness structure, say \mathcal{W}_0 , such that $\mathcal{P}_r(D)$ is a collection of witness sets in an end-column. The induction hypothesis also implies and $k_0 + |N(S_0)| - r \leq k + 3r$.

34:12 On the Parameterized Complexity of Grid Contraction

Let $\mathcal{P}_r(A) = \langle A_1, A_2, \dots, A_r \rangle$ be the r -partition of A in $\mathcal{A}_{a,b}[(S_0, \mathcal{P}_r(A))]$ at which *for-loop at Index* changes the value at $\Gamma[(S_1, \mathcal{P}_r(A)); k_1]$. By construction, every D_i in $\mathcal{P}_r(D)$, D_i is contained in A_i . Since $\Phi(S_0)$ is contained in D , no vertex in $S_0 \setminus D$ is adjacent with any vertex in A . Since $\mathcal{P}_r(A)$ is a r -slab, $\mathcal{W}_0 \cup \{A_1, A_2, \dots, A_r\}$ is a grid witness structure of $G[S_1]$. Moreover, since $N(S_0)$ is in A , $\Phi(S_1)$ is contained in A . Hence, $G[S_1]$ can be k_1 -contractible to a grid with all vertices in $\Phi(S)$ in a A and there exists a witness structure for which $\mathcal{P}_r(A)$ is a collection of witness sets in an end-columns. It remains to argue that $k_1 + |N(S_1)| - r \leq k$. We prove this for the case $k_1 = k_0 + a - r$ as $k_1 > k_0 + a - r$ case follows from the definition of k_1 -contractibility. Let $x_0 = |D|$. As x_0 is the size of an end-column in \mathcal{W}_0 , we have $x_0 - r \leq k_0$. As algorithm only considers a, b such that $x_0 + a + b \leq k + 3r$, substituting $a = k_1 - k_0 + r$ and $b = |N(S_1)|$ we get $x_0 + k_1 - k_0 + r + |N(S_1)| \leq k + 3r$. Using $x_0 - r \leq k_0$, we get the desired bound.

This completes the proof of the lemma. \blacktriangleleft

► **Lemma 12.** *Given a graph G on n vertices and integers k, r , the algorithm terminates in time $4^{k+3r} \cdot n^{\mathcal{O}(1)}$.*

Proof. We first describe an algorithm that given a graph G on n vertices and integers k, r , enumerates all valid tuples in time $4^{k+3r} \cdot n^{\mathcal{O}(1)}$. The algorithm computes all r -slabs in G which satisfy first property in Definition 8 using Corollary 7. For every r -slabs, it checks whether it satisfy the second property in Definition 8 to determine whether it is a k -potential r -slab or not. For a k -potential r -slab $\mathcal{P}_r(D) \equiv \langle D_1, D_2, \dots, D_r \rangle$, if $G - D$ has exactly one connected component, say C_1 , then it adds $(V(C_1) \cup D, \mathcal{P}_r(D))$ and $(D, \mathcal{P}_r(D))$ to set of valid tuples. If $G - D$ has two connected components, say C_1, C_2 , then it adds $(V(C_1) \cup D, \mathcal{P}_r(D))$ and $(V(C_2) \cup D, \mathcal{P}_r(D))$ to the set of valid tuples. This completes the description of the algorithm. Note that the algorithm returns a set of valid tuples. For a k -potential r -slab $\mathcal{P}_r(D) \equiv \langle D_1, D_2, \dots, D_r \rangle$, $G - D$ has at most two connected components. Hence any k -potential r -slab is responsible for at most two nice subsets. By definition of nice subsets, for any nice subset there exists a k -potential r -slab responsible for it. Hence the algorithm constructs the set of all valid tuples. The algorithm spends polynomial time for each r -slab it constructs. Hence, the running time of the algorithm follows from Corollary 7.

The algorithm can compute the table and complete *for-loop Initialization* in time $4^{k+3r} \cdot n^{\mathcal{O}(1)}$ using the algorithm mentioned in above paragraph. We now argue that the *for-loop Table* takes $4^{k+3r} \cdot n^{\mathcal{O}(1)}$ time to complete. We partition the set of valid tuples \mathcal{V}_k using the sizes of the neighborhood of connected component and size of r -slab in a tuple. For two fixed integers x, y , define $\mathcal{V}_k^{x,y} := \{(S, \mathcal{P}_r(D)) \in \mathcal{V}_k \mid |D| \leq x \text{ and } |N(S)| \leq y\}$. In other words, $\mathcal{V}_k^{x,y}$ collection of all nice subsets whose neighborhood is of size y and there is a k -potential r -slab of size x responsible for it. Alternatively, $\mathcal{V}_k^{x,y}$ is a collection of k -nice subsets for which there is a (x, y) - r -slab is responsible for it. Since the number of (x, y) - r -slabs are bounded (Corollary 7) and each k -potential r -slab is responsible for at most two nice subsets, $|\mathcal{V}_k^{x,y}|$ is bounded by $4^{x+y} \cdot n^{\mathcal{O}(1)}$.

For each $(S, \mathcal{P}_r(D)) \in \mathcal{V}_k^{x,y}$, the algorithm considers every pair of integers $a(> 0), b(\geq 0)$, such that $x + a + b \leq k + 3$ and $|N(S)| = y \leq a$, and computes the set $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$. By Lemma 6, set $\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]$ can be computed in time $4^{a+b-|N(S)|} \cdot n^{\mathcal{O}(1)}$. The algorithm spends time proportional to $|\mathcal{A}_{a,b}[(S, \mathcal{P}_r(D))]|$ for *for-loop at Index*. Hence for two fixed integers x, y , algorithm spends

$$\sum_{\substack{a,b \\ x+a+b \leq k+3r}} 4^{x+y} \cdot 4^{a+b-y} \cdot n^{\mathcal{O}(1)} = \sum_{\substack{a,b \\ x+a+b \leq k+3r}} 4^{x+a+b} \cdot n^{\mathcal{O}(1)} = 4^{k+3r} \cdot n^{\mathcal{O}(1)}$$

time to process all valid tuples in $\mathcal{V}_k^{x,y}$. Since there are at most $\mathcal{O}(k^2)$ feasible values for x, y , the overall running time of algorithm is bounded by $4^{k+3r} \cdot n^{\mathcal{O}(1)}$. This concludes the proof. \blacktriangleleft

The following theorem is implied by Lemmas 11, 12, and the fact that $(V(G), \mathcal{P}_r(D))$ is a tuple in \mathcal{V}_k for some D .

► **Theorem 13.** *There exists an algorithm which given an instance (G, k, r) of BOUNDED GRID CONTRACTION runs in time $4^{k+3r} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a YES instance or not. Here, n is the number of vertices in G .*

5 An FPT algorithm for Grid Contraction

In this section, we present an FPT algorithm for GRID CONTRACTION. Given instance (G, k) of GRID CONTRACTION is a YES instance if and only if (G, k, r) is a YES instance of BOUNDED GRID CONTRACTION for some r in $\{1, 2, \dots, |V(G)|\}$. For $r < 2k + 5$, we can use algorithm presented in Section 4 to check whether given graph can be contracted to grid with r rows or not in FPT time. A choice of this threshold will be clear in the latter part of this section. If algorithm returns YES then we can conclude that (G, k) is a YES instance of GRID CONTRACTION. If not then we can correctly conclude that if G is k -contractible to a grid then the resulting grid has at least $2k + 5$ rows. This information allows us to find two rows in G which can safely be contracted. We need the following generalized version of GRID CONTRACTION to state these results formally.

ANNOTATED BOUNDED GRID CONTRACTION

Parameter: k, r

Input: Graph G , integers k, r, q , and a tuple (x_1, x_2, x_3, x_4) of four different vertices in $V(G)$

Question: Is G k -contractible to $\boxplus_{r \times q}$ such that there is a $\boxplus_{r \times q}$ -witness structure of G in which the witness sets containing x_1, x_2, x_3 , and x_4 correspond to four corners in $\boxplus_{r \times q}$?

Assume that G is k -contractible to $\boxplus_{r \times q}$ with desired properties via mapping ψ . Let t_1, t_2, t_3 , and t_4 be corners in $\boxplus_{r \times q}$ such that $t_1 \equiv [1, 1], t_2 \equiv [1, q], t_3 \equiv [r, q]$, and $t_4 \equiv [r, 1]$. There are $4!$ ways in which vertices in $\{x_1, x_2, x_3, x_4\}$ can be uniquely mapped to corners $\{t_1, t_2, t_3, t_4\}$. For the sake of simplicity, we assume that we are only interest in the case in which x_1, x_2, x_3, x_4 are mapped to t_1, t_2, t_3 , and t_4 respectively. In other words, $\psi(x_i) = t_i$ for all $i \in \{1, 2, 3, 4\}$.

We can modify the algorithm presented in Section 4 obtain an algorithm for ANNOTATED BOUNDED GRID CONTRACTION problem which is fixed parameter tractable when parameterized by $(k + r)$ (\star).

► **Lemma 14.** *There exists an algorithm which given an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$ of ANNOTATED BOUNDED GRID CONTRACTION runs in time $4^{k+3r} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a YES instance or not. Here, n is the number of vertices in G .*

In the case, when $r < 2k + 5$ the algorithm mentioned in the above lemma is fixed parameter tractable when the parameter is k alone. When $r \geq 2k + 5$, we argue that if $(G, k, r, q, (x_1, x_2, x_3, x_4))$ is a YES instance then there exists a *horizontal decomposition* of G (Lemma 16). We formally define horizontal decomposition as follows.

34:14 On the Parameterized Complexity of Grid Contraction

► **Definition 15** (Horizontally-Decomposable). *Consider an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$ of ANNOTATED BOUNDED GRID CONTRACTION. A graph G is said to be horizontally-decomposable if $V(G)$ can be partitioned into four non-empty parts C_{12}, S_u, S_v , and C_{34} which satisfies following properties.*

- *The graphs $G[C_{12}], G[C_{34}]$ are connected and $x_1, x_2 \in C_{12}, x_3, x_4 \in C_{34}$.*
- *The graph $G[S_u \cup S_v]$ is a $2 \times q$ grid with S_u, S_v correspond to vertices in its two rows.*
- *C_{12} and C_{34} are the two connected components of $G \setminus (S_u \cup S_v)$.*
- *$N(C_{12}) = S_u$ and $N(C_{34}) = S_v$.*

► **Lemma 16** (\star). *Consider an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$ of ANNOTATED BOUNDED GRID CONTRACTION such that $2k+5 \leq r$. If it is a YES instance then there exists a horizontal decomposition of G .*

Consider an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$, let $(C_{12}, S_u, S_v, C_{34})$ be a horizontal decomposition of G . Reduction Rule 5.1 contracts all the edges across S_u, S_v . Note that in the resulting instance, r is decreased by one.

► **Reduction Rule 5.1.** *For an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$, let $(C_{12}, S_u, S_v, C_{34})$ be a horizontal decomposition of G . Let $S_u (= \{u_1, u_2, \dots, u_q\})$ and $S_v (= \{v_1, v_2, \dots, v_q\})$. Let G' be the graph obtained from G by contracting all the edges in $\{u_j v_j \mid j \in [q]\}$. Return instance $(G', k, r-1, q, (x_1, x_2, x_3, x_4))$.*

As S_u, S_v are $\{(x_1 - x_4), (x_2 - x_3)\}$ -separators in G , by Observation 2.2, sets $\psi(S_u), \psi(S_v)$ are $\{(t_1 - t_4), (t_2 - t_3)\}$ -separators in $\boxplus_{r \times q}$. We argue that $\psi(S_u)$ and $\psi(S_v)$ correspond to two consecutive rows and it was safe to contract edges across S_u, S_v .

► **Lemma 17** (\star). *Reduction Rule 5.1 is safe.*

It remains to argue that Reduction Rule 5.1 can be implemented in polynomial time. In Lemma 19, we argue there exists an algorithm that can find a horizontal decomposition, if exists, in polynomial time. We use the following structural lemma to prove the previous statement.

► **Lemma 18** (\star). *Given two adjacent vertices u_1, v_1 in G , there is at most one subset S of $V(G)$ such that (a) $G[S]$ is a $(2 \times q)$ grid, (b) u_1, v_1 are two vertices in the first column of $G[S]$, and (c) each row in S is a separator in G . Moreover, if such a subset exists then it can be found in polynomial time.*

► **Lemma 19** (\star). *There exists an algorithm which given an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$ of ANNOTATED BOUNDED GRID CONTRACTION runs in polynomial time and either returns a horizontal decomposition of G or correctly concludes that no such decomposition exists.*

We are now in a position to present main result of this section.

► **Theorem 20.** *There exists an algorithm which given an instance (G, k) of GRID CONTRACTION runs in time $4^{6k} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a YES instance or not. Here, n is the number of vertices in G .*

Proof. The algorithm starts with checking whether graph G is k -contractible to a path using the algorithm in [15]. If it is then the algorithm returns YES else it creates polynomially many instances of ANNOTATED BOUNDED GRID CONTRACTION by guessing all possible values of r, q, x_1, x_2, x_3, x_4 . It processes these instances with increasing values of r . Ties are broken arbitrarily. For $r < 2k + 5$, the algorithm check whether $(G, k, r, q, (x_1, x_2, x_3, x_4))$ is a YES

instance of ANNOTATED BOUNDED GRID CONTRACTION using Lemma 14. For $r \geq 2k + 5$, the algorithm checks whether there exists a horizontal decomposition of G using Lemma 19. If there exists a horizontal decomposition of G then the algorithm applies Reduction Rule 5.1 to obtain another instance of ANNOTATED BOUNDED GRID CONTRACTION with a smaller value of r . The algorithm repeats the above step until $r < 2k + 5$ or the graph in a reduced instance does not have a horizontal decomposition. In the first case, it checks whether a reduced instance is a YES instance or not using Lemma 14. In the second case, it continues to the next instance created at the start of the algorithm. The algorithm returns YES if at least one of the instances of ANNOTATED BOUNDED GRID CONTRACTION is a YES instance.

It is easy to see that an instance (G, k) of GRID CONTRACTION is a YES instance if and only if there exists integers r, q in $\{1, 2, \dots, |V(G)|\}$ and four vertices x_1, x_2, x_3, x_4 in $V(G)$ such that $(G, k, r, q, (x_1, x_2, x_3, x_4))$ is a YES instance of ANNOTATED BOUNDED GRID CONTRACTION. Lemma 17 implies the correctness of the step where the algorithm repeatedly applies Reduction Rule 5.1 and check whether the reduced instance is a YES instance of ANNOTATED BOUNDED GRID CONTRACTION or not. Consider an instance $(G, k, r, q, (x_1, x_2, x_3, x_4))$ such that $r > 2k + 5$ and there is no horizontal decomposition of G . By Lemma 16, the algorithm correctly concludes that it is a NO instance and continues to the next instance. This implies the correctness of the algorithm. The running time of the algorithm is implied by Lemmas 14, 19 and the fact that the algorithm presented in [15] runs in time $2^{k+o(k)} \cdot n^{\mathcal{O}(1)}$. ◀

6 NP-Completeness and Lower Bounds

In this section, we prove that GRID CONTRACTION problem is NP-Complete. We also argue that the dependency on the parameter in the running time of the algorithm presented in Section 5 is optimal, up to constant factors in the exponent, under a widely believed hypothesis. Brouwer and Veldman presented a reduction from HYPERGRAPH 2-COLORABILITY problem to H -CONTRACTION problem [5]. We present a reduction from NAE-SAT problem to HYPERGRAPH 2-COLORABILITY problem. We argue that the reduction used by Brouwer and Veldman can be used to reduce the HYPERGRAPH 2-COLORABILITY problem to GRID CONTRACTION problem. Using these reductions and the fact there is no *sub-exponential* time algorithm for NAE-SAT, we obtain desired results.

▶ **Theorem 21.** *GRID CONTRACTION is NP-Complete. Moreover, unless ETH fails, it can not be solved in time $2^{o(n)}$, where n is the number of vertices in an input graph.*

7 Kernelization

In this section, we present a polynomial kernel for the GRID CONTRACTION problem. In Section 5, we reduced an instance of GRID CONTRACTION to polynomially many instances of ANNOTATED BOUNDED GRID CONTRACTION such that the original instance is a YES instance if and only one of these instances is a YES instance. One can argue that exhaustively application of Reduction Rule 5.1 leads to a *Turing Compression*¹ of the size $\mathcal{O}(k^2)$. We use the similar approach, but with weaker bounds, to obtain a kernel of size $\mathcal{O}(k^4)$.

If the input graph is not connected then we can safely conclude that we are working with a NO instance. The following reduction rule checks two more criteria in which it is safe to return a NO instance.

¹ Please see, for example, [10, Chapter 22] for formal definition.

34:16 On the Parameterized Complexity of Grid Contraction

- **Reduction Rule 7.1.** For an instance (G, k) , if
- there exists a vertex in G whose degree is more than $k + 5$, or
 - there are $6k + 1$ vertices in G whose degrees are more than 5,
- then return a trivial NO instance.

- **Lemma 22** (\star). Reduction Rule 7.1 is safe.

We define $k_o = (4k + 8) \cdot (k + 1) + 1$. Consider an instance (G, k) on which Reduction Rule 7.1 is not applicable. If G has at most $k_o^2 + k + 1$ vertices then we can argue that we have a kernel of the desired size. Consider a case when $|V(G)| \geq k_o^2 + k + 1$. We argue that in this case, if (G, k) is a YES instance then there exists a large grid separator in a graph G (Lemma 24).

- **Definition 23** ($(p \times t)$ -grid-separator). Consider an instance (G, k) of GRID CONTRACTION. A subset S of $V(G)$ is called a $(p \times t)$ -grid-separator of G if it has following three properties.
- $G[S] = \Gamma_{p \times t}$.
 - Graph $G - S$ has exactly two connected components, say C_1 and C_2 .
 - $|V(C_1)|, |V(C_2)| \geq k + 1$ and $N(C_1) = R_1, N(C_2) = R_p$, where R_1, R_p are the first and last row in $G[S]$.

- **Lemma 24** (\star). Consider an instance (G, k) of GRID CONTRACTION such that $|V(G)| \geq k_o^2 + k + 1$. If (G, k) is a YES instance then there exists a $((4k + 6) \times t)$ -grid-separator in G for some integer t .

We argue that if there is a large grid that is a separator in G then we can safely contract two consecutive rows in this grid.

- **Reduction Rule 7.2.** For an instance (G, k) , let S be a $((4k + 6) \times t)$ -grid-separator of G for some integer t . Let $S_u (= \{u_1, u_2, \dots, u_t\})$ and $S_v (= \{v_1, v_2, \dots, v_t\})$ be two consecutive internal rows in S . Let G' be the graph obtained from G by contracting all the edges in $\{u_j v_j \mid j \in [t]\}$. Return instance (G', k) .

We prove that the reduction rule is safe along the same line as that of Lemma 17.

- **Lemma 25** (\star). Reduction Rule 7.2 is safe.

The following lemma, which is analogous to Lemma 19, is essential to argue that Reduction Rule 7.2 can be applied in polynomial time.

- **Lemma 26** (\star). There exists an algorithm which given an instance (G, k) of GRID CONTRACTION and integers p, t runs in polynomial time and either returns a $(p \times t)$ -grid-separator of G or correctly concludes that no such separator exists.

We are now in a position to present the main result of the section.

- **Theorem 27** (\star). GRID CONTRACTION admits a kernel with $\mathcal{O}(k^4)$ vertices and edges.

References

- 1 Akanksha Agrawal, Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Prafullkumar Tale. Path contraction faster than 2^n . *The 46th International Colloquium on Automata, Languages and Programming (ICALP 2019)*, 2019.
- 2 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. *ACM Transactions on Computation Theory (TOCT)*, 11(3):1–22, 2019.
- 3 Takao Asano and Tomio Hirata. Edge-Contraction Problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983.
- 4 Rémy Belmonte, Petr A. Golovach, Pim Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014.
- 5 Andries Evert Brouwer and Henk Jan Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.
- 6 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *International Symposium on Parameterized and Exact Computation*, pages 97–109. Springer, 2013.
- 7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized complexity*. Springer-Verlag, 2013.
- 9 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- 10 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 11 Petr A Golovach, Marcin Kamiński, Daniël Paulusma, and Dimitrios M Thilikos. Increasing the minimum degree of a graph by contractions. *Theoretical computer science*, 481:74–84, 2013.
- 12 Petr A. Golovach, Pim van 't Hof, and Daniel Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013.
- 13 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for bipartite contraction. *Inf. Process. Lett.*, 113(22–24):906–912, 2013.
- 14 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013.
- 15 Pinar Heggernes, Pim Van't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014.
- 16 R. Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 23:1–23:14, 2016.
- 17 R Krithika, Pranabendu Misra, and Prafullkumar Tale. An FPT algorithm for contraction to cactus. In *International Computing and Combinatorics Conference*, pages 341–352. Springer, 2018.
- 18 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *International Symposium on Parameterized and Exact Computation*, pages 243–254, 2013.
- 19 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.
- 20 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981.
- 21 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and-contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983.

Simplification of Polyline Bundles

Joachim Spoerhase 

Aalto University, Finland
University of Würzburg, Germany
joachim.spoerhase@aalto.fi

Sabine Storandt

University of Konstanz, Germany
sabine.storandt@uni-konstanz.de

Johannes Zink 

University of Würzburg, Germany
zink@informatik.uni-wuerzburg.de

Abstract

We propose and study a generalization to the well-known problem of polyline simplification. Instead of a single polyline, we are given a set of ℓ polylines possibly sharing some line segments and bend points. Our goal is to minimize the number of bend points in the simplified bundle with respect to some error tolerance δ (measuring Fréchet distance) but under the additional constraint that shared parts have to be simplified consistently. We show that polyline bundle simplification is *NP*-hard to approximate within a factor $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$ where n is the number of bend points in the polyline bundle. This inapproximability even applies to instances with only $\ell = 2$ polylines. However, we identify the sensitivity of the solution to the choice of δ as a reason for this strong inapproximability. In particular, we prove that if we allow δ to be exceeded by a factor of 2 in our solution, we can find a simplified polyline bundle with no more than $\mathcal{O}(\log(\ell + n)) \cdot OPT$ bend points in polytime, providing us with an efficient bi-criteria approximation. As a further result, we show fixed-parameter tractability in the number of shared bend points.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Polyline Simplification, Bi-criteria Approximation, Hardness of Approximation, Geometric Set Cover

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.35

Funding *Joachim Spoerhase*: Funded by European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 759557).

Sabine Storandt: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 50974019 – TRR 161.

1 Introduction

Visualization of geographical information is a task of high practical relevance, e.g., for the creation of online maps. Such maps are most helpful if the information is neatly displayed and can be grasped quickly and unambiguously. This means that the full data often needs to be filtered and abstracted. Many important elements in maps like borders, streets, rivers, or trajectories are displayed as polylines (also known as polygonal chains). For such a polyline, a simplification is supposed to be as sparse as possible and as close to the original as necessary.

A simplified polyline is usually constructed by a subset of bend points of the original polyline such that the (local) distance to the original polyline does not exceed a specifiable value according to a given distance measure, e.g., Fréchet distance or the Hausdorff distance.



© Joachim Spoerhase, Sabine Storandt, and Johannes Zink;
licensed under Creative Commons License CC-BY

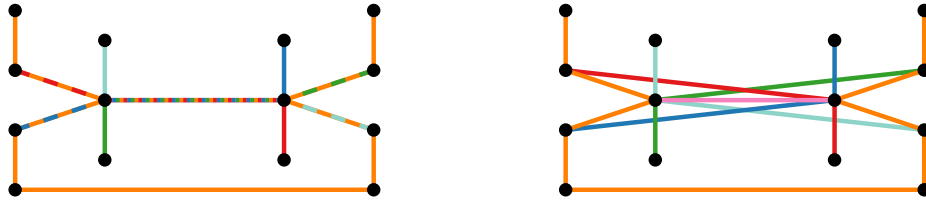
17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 35; pp. 35:1–35:20

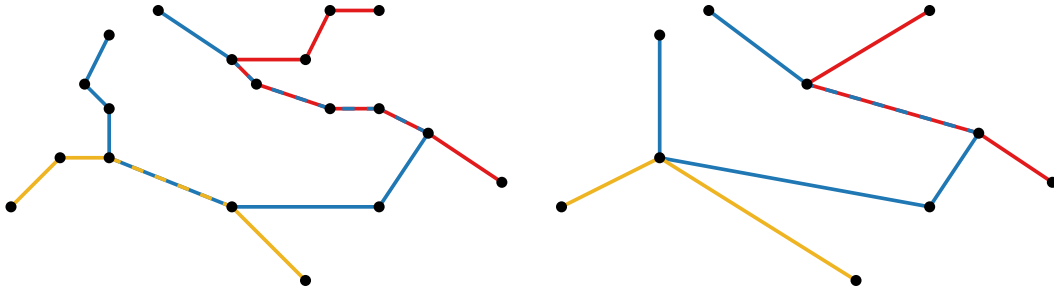
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Example where the total complexity increases if each polyline is simplified independently. Left: Initial bundle of polylines. Right: Bundle of independently simplified polylines.



■ **Figure 2** Example of a bundle of three polylines before and after consistent simplification.

The first such algorithm, which is still of high practical importance, was proposed by Ramer [16] and by Douglas and Peucker [7]. Hershberger and Snoeyink [13] proposed an implementation of this algorithm that runs in $\mathcal{O}(n \log n)$ time, where n is the number of bend points in the polyline. It is a heuristic algorithm as it does not guarantee optimality (or something close to it) in terms of retained bend points. An optimal algorithm in this sense was first proposed by Imai and Iri [14]. Chan and Chin [5] improved the running time of this algorithm to $\mathcal{O}(n^2)$ for the Hausdorff distance. For the Fréchet distance, the optimal solution can be determined in time $\mathcal{O}(n^3)$ as described by Godau [10].

We remark that all of these algorithms consider the distance *segment-wise*. This is, the distance between each segment of the simplification and its corresponding sub-polyline of the input polyline does not exceed the given threshold. We adhere to this widespread approach. Intuitively and from an application point of view, it makes sense to map a point p of the input polyline only to a point of a segment of the simplification “spanning over” p with respect to the input polyline as this ensures a certain degree of locality. However, the general unrestricted approach has also received attention in the literature. Here, the Hausdorff or Fréchet distance between the input polyline and the simplification as a whole polyline is considered. For the (undirected) Hausdorff distance, this problem becomes *NP-hard* [17] and for the Fréchet distance, there is an $\mathcal{O}(kn^5)$ time algorithm, where k is the output complexity of the simplification [17]. The problem variant where in addition the requirement is dropped that all bend points of the simplification must be bend points of the input polyline, is called a *weak* simplification. Agarwal et al. [1] show that an optimal simplification under the segment-wise Fréchet distance with distance threshold δ , as computable using the algorithm by Imai and Iri, has no more bend points than an optimal weak simplification with distance threshold $\delta/4$. We note that computing the Fréchet distance between two polylines can be solved in polynomial time [2], but may become *NP-hard* when considering additional properties like allowing to take shortcuts, which replace outliers in one of the polylines [3].

From a Single Polyline to a Bundle of Polylines

On a map, there are usually multiple polylines to display. Such polylines may share bend points (bends) and line segments between bends (segments) sectionwise. We call them a *bundle* of polylines. One example is a schematic map of a public transport network where bus lines are the polylines and these share some of the stations and legs.

One might consider simplifying the polylines of a bundle independently. This has some drawbacks, though. On the one hand, the total complexity might even increase when the shared parts are simplified in different ways; see Figure 1. On the other hand, it might suggest a misleading picture when we remove common segments and bends of some polylines, but not of all. Therefore, we require that a bend in a simplification of a bundle of polylines is either kept in all polylines containing it or discarded in all polylines. Our goal is then to minimize the total number of bend points that have to be kept. In Figure 2, we give an example of a simplification of a bundle of polylines.

Related Work

Polyline bundles were studied before in different contexts. In [4], the goal is to interfere a concise graph which represents all trajectories in a given bundle sufficiently well. But this approach primarily aims at retrieving split and merge points of trajectories correctly and does not produce a simplification of each trajectory in the bundle. Methods for map generation based on movement trajectories [12] have a similar scope but explicitly allow to discard outliers and to unify sufficiently similar trajectories, which is not allowed in our setting.

Agarwal et al. [1] describe an $\mathcal{O}(n \log n)$ time approximation algorithm for (classical) polyline simplification under the Fréchet distance. It is an approximation algorithm in the sense that the output simplification for distance threshold δ has at most as many bends as an optimal solution with distance threshold $\delta/2$. In Theorem 10, we also relate the size of our approximate solution respecting a distance threshold of δ to an optimal solution with distance threshold $\delta/2$.

There is also a multitude of polyline simplification problem variants for single polylines which involve additional constraints. One important variant is the computation of the smallest possible simplification of a single polyline which avoids self-intersection [6]. Another practically relevant variant is the consideration of topological constraints. For example, if the polyline represents a country border, important cities within the country should remain on the same side of the polyline after simplification. It was proven that those problem variants are hard to approximate within a factor $n^{\frac{1}{5}-\epsilon}$ [8]. Hence, in practice, they are typically tackled with heuristic approaches [8, 9].

Note that the only allowed inputs to those problem variants are either a single polyline without self-intersections or a set of polylines without self-intersections and without common bends or segments (except for common start or end points). In contrast, we explicitly allow non-planar inputs and polyline bundles in which bends and segments may be shared among multiple polylines. We also remark that the known results on hardness of approximation of these problems heavily rely on the constraint that feasible solutions are still non-intersecting. Since we do not require this, we have to resort to different techniques.

Contribution

We introduce the optimization problem of polyline bundle simplification, where we are given ℓ polylines on an underlying set of n points as well as an error bound δ and seek to find a simplified polyline bundle with the smallest possible number of remaining points, where each simplified polyline has a Fréchet distance of no more than δ to the original polyline and the simplification is consistent for shared parts.

While the optimal simplification of a single polyline can be computed in polynomial time, we show that polyline bundle simplification is NP -hard to approximate within a factor $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. This result applies already to bundles of two polylines, hence excluding an efficient FPT-algorithm depending on parameter ℓ .

On the positive side, we show that this strong inapproximability can be overcome when relaxing the error bound δ slightly. In particular, we design an efficient bi-criteria approximation algorithm. Here, we allow the simplified polylines in our solution to have a Fréchet distance of 2δ instead of only δ to the original polylines. We can then approximate the optimal solution for the original choice of δ within a factor logarithmic in the input size. As the choice of δ for real-world problems often is made in a rather ad hoc fashion and uncertainties with respect to the precision of the input polylines have to be factored in as well, we deem our bi-criteria approximation to be of high practical relevance.

We furthermore show that, while the number of polylines in the bundles is not suitable to obtain an FPT-algorithm, the problem of polyline bundle simplification is indeed fixed-parameter tractable in the number of bend points that are shared among the polylines.

2 Formal Problem Definition

An instance of the *polyline bundle simplification* problem (from now on abbreviated by PBS) is specified by a triple (B, \mathcal{L}, δ) , where $B = \{b_1, \dots, b_n\}$ is a set of n points (*bends*) in the plane, a *polyline bundle* \mathcal{L} , which is a set $\mathcal{L} = \{L_1, \dots, L_\ell\}$ of ℓ polylines $L_i = (s_i, \dots, t_i)$ represented as lists of points from B , and a distance parameter δ , which specifies a threshold for the the maximum (segment-wise) Fréchet distance between original and simplified polyline bundle. Each polyline L_i ($i \in \{1, \dots, \ell\}$) is simple in the sense that each bend of B appears at most once in its list.

► **Definition 1** (Polyline Bundle Simplification). *Given a triple (B, \mathcal{L}, δ) , the goal is to obtain a minimum size subset $B^* \subseteq B$ of points, such that for each polyline $L_i \in \mathcal{L}$ its induced simplification S_i (which is $L_i \cap B^*$ while preserving the order of points)*

- *contains the start and the end point of L_i , i.e., $s_i, t_i \in S_i$, and*
- *has a segment-wise Fréchet distance of at most δ to L_i , i.e., for each line segment (a, b) of S_i and the corresponding sub-polyline of L_i from a to b , abbreviated by $L_i[a, \dots, b]$, we have $d_{\text{Fréchet}}((a, b), L_i[a, \dots, b]) \leq \delta$.*

For the sake of self-containedness we restate the definition of the Fréchet distance below.

► **Definition 2** (Fréchet Distance). *Between two polylines $L_1 = (b_{1,1}, b_{1,2}, \dots, b_{1,|L_1|})$ and $L_2 = (b_{2,1}, b_{2,2}, \dots, b_{2,|L_2|})$ in the Euclidean plane, the Fréchet distance $d_{\text{Fréchet}}(L_1, L_2)$ is*

$$d_{\text{Fréchet}}(L_1, L_2) := \inf_{\alpha, \beta} \max_{t \in [0,1]} \|c_{L_1}(\alpha(t)) - c_{L_2}(\beta(t))\|,$$

where $\alpha: [0, 1] \rightarrow [1, |L_1|]$ and $\beta: [0, 1] \rightarrow [1, |L_2|]$ are continuous and non-decreasing functions with $\alpha(0) = \beta(0) = 1$, $\alpha(1) = |L_1|$, $\beta(1) = |L_2|$,

and $c_{L_i}: [1, |L_i|] \rightarrow \mathbb{R}^2$ with $c_{L_i}: x \mapsto (\lfloor x \rfloor + 1 - x)b_{i, \lfloor x \rfloor} + (x - \lfloor x \rfloor)b_{i, \lfloor x \rfloor + 1}$.

3 Hardness of Polyline Bundle Simplification

In this section, we describe a polynomial-time reduction from Minimum Independent Dominating Set (MIDS) to PBS to show NP -hardness and hardness of approximation. In the MIDS problem, we are given a graph $G = (V, E)$, where V is the vertex set and E is the edge set of G . We define $\hat{n} = |V|$ and $\hat{m} = |E|$. The goal is to find a set $V^* \subseteq V$ of minimum cardinality that is a dominating set of G as well as an independent set in G . A dominating set contains for each vertex v , v itself or at least one of v 's neighbors. An independent set contains for each edge at most one of its endpoints. Halldórsson [11] has shown that MIDS, which is also referred to as MINIMUM-MAXIMAL-INDEPENDENT-SET, is NP -hard to approximate within a factor of $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$. In his proof, he uses a reduction from SAT to MIDS: from a SAT formula Φ , he constructs a graph such that an algorithm approximating MIDS would decide if Φ is satisfiable. We observe that this reduction is still correct if Φ is a 3-SAT formula. Moreover, we observe that the number of edges in the graph constructed in this reduction by a 3-SAT formula is linear in the number of vertices. Thus, we conclude the following corollary and assume henceforth that we reduce only from sparse graph instances of MIDS, in other words, $\hat{m} \leq c\hat{n}$ for some sufficiently large constant c .

► **Corollary 3.** *MIDS on graphs of \hat{n} vertices and $\mathcal{O}(\hat{n})$ edges, i.e., sparse graphs, is NP -hard to approximate within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$.*

In our reduction, we use three types of gadgets, which are in principle all lengthy zigzag pieces. We use *vertex gadgets* to indicate whether a vertex is in the set V^* or not, *edge gadgets* to enforce the independent set property, and *neighborhood gadgets* to enforce the dominating set property. See Fig. 3 for an overview. We define our gadgets in terms of an arbitrary δ (threshold for the maximum Fréchet distance) and some $\gamma \leq 2\delta/(10\hat{n}^2 + 5)$. Note that our problem setting allows overlaps of different polylines without having a common bend or segment (non-planar input). In our reduction there can also be overlaps, which do not affect the involved polylines locally.

Vertex Gadget. For each vertex, we construct a *vertex gadget* (see Figure 3a), which we arrange vertically next to each other on a horizontal line in arbitrary order and with some distance $x_{\text{spacing}} \geq (2\hat{n}^2 + 2)3\delta$ between one and the next vertex gadget.

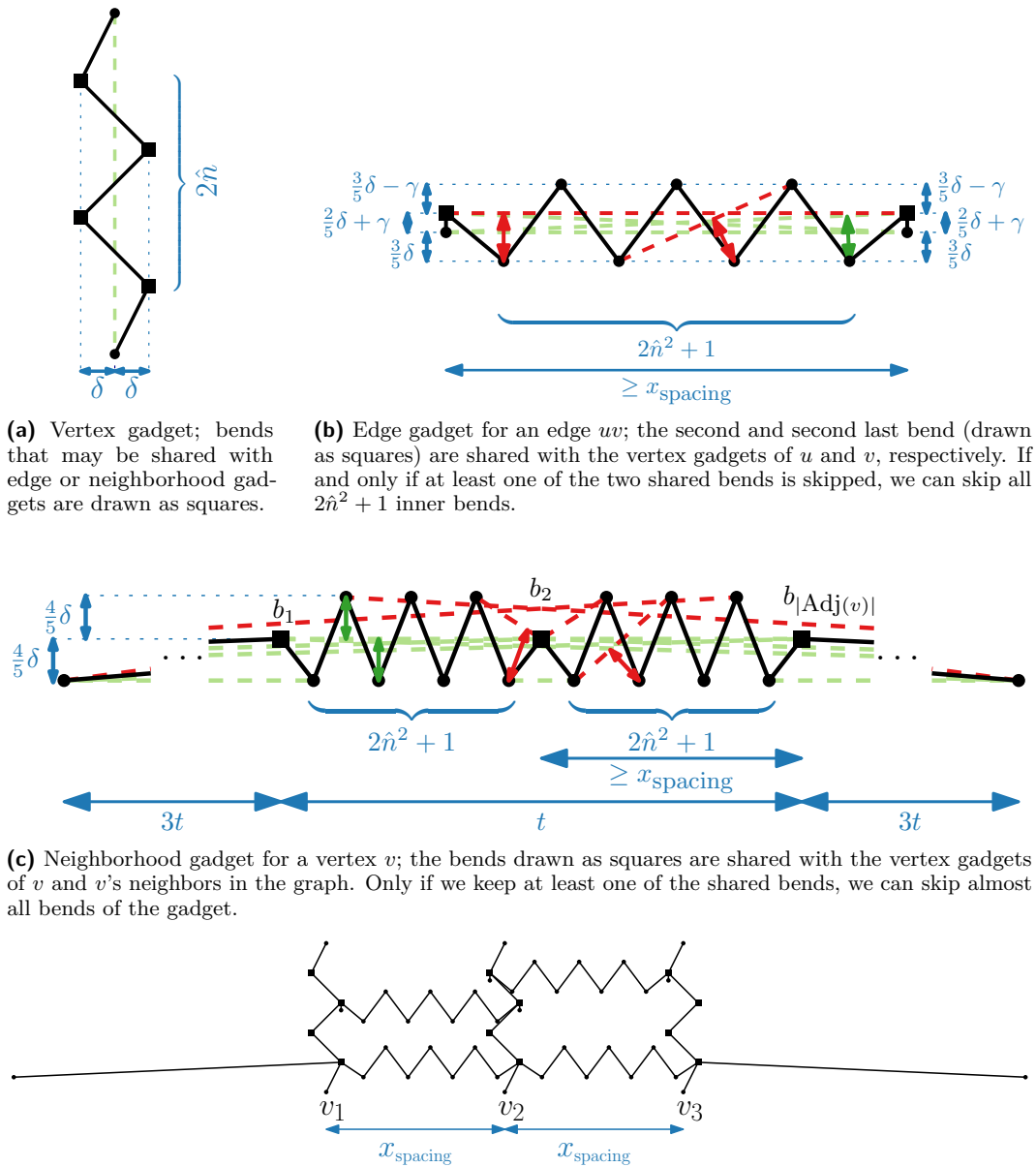
A vertex gadget has $2\hat{n} + 2$ bends arranged in a zigzag course with x-distance 2δ (δ for the first and the last segment) and y-distance 3δ between each two consecutive bends.

► **Claim 4.** In a vertex gadget, there is precisely one shortcut, which starts at the first and ends at the last bend.

Clearly, the line segment from the first to the last bend has Fréchet distance at most δ to the other bends and segments of the vertex gadget. Moreover, observe that there is no shortcut starting or ending at any inner bend. Thus, either none or all inner bends are skipped. We say that the corresponding vertex is in V^* if and only if we do not skip the inner vertices.

Edge Gadget. For each edge $\{u, v\}$, we construct an *edge gadget* (see Figure 3b) being a zigzag course with $2\hat{n}^2 + 5$ bends and sharing its second and second last bend with one of the two corresponding vertex gadgets – the vertex gadgets of u and v . All neighboring bends from the second to the second last are equidistant in x-dimension, while the first and second bend, and the second last and last bend have the same x-coordinate. In y-dimension, the first and the last bend are $2/5\delta + \gamma$ below the second and second last bend, respectively. The other bends are $3/5\delta - \gamma$ above the second bend or $3/5\delta$ below the first bend.

35:6 Simplification of Polyline Bundles



(a) Vertex gadget; bends that may be shared with edge or neighborhood gadgets are drawn as squares. (b) Edge gadget for an edge uv ; the second and second last bend (drawn as squares) are shared with the vertex gadgets of u and v , respectively. If and only if at least one of the two shared bends is skipped, we can skip all $2\hat{n}^2 + 1$ inner bends.

(c) Neighborhood gadget for a vertex v ; the bends drawn as squares are shared with the vertex gadgets of v and v 's neighbors in the graph. Only if we keep at least one of the shared bends, we can skip almost all bends of the gadget.

(d) Combination of three vertex gadgets (for the vertices v_1, v_2, v_3) with two edge gadgets (for the edges v_1v_2 and v_2v_3) and a neighborhood gadget for the vertex v_2 .

■ **Figure 3** Schematization of our reduction from MIDS to PBS. Shortcuts are indicated by dashed green line segments. Dashed red line segments between two bends indicate that there is no shortcut. The vertices in our minimum independent dominating set are precisely the ones for which we do not take the shortcut of the corresponding vertex gadgets.

▷ **Claim 5.** In an edge gadget, there are precisely three long shortcuts. These are (i) from the first to the last bend, (ii) from the first to the second last bend, and (iii) from the second to the last bend. Beside these three shortcuts, there are ≤ 4 more shortcuts, which skip only the second and the second last bend (and possibly also the third and third last bend). There is no shortcut not skipping one of the shared bends, i.e., the second or the second last bend.

In Appendix A, we argue that Claim 5 is correct. It follows that not skipping one of the two shared bends is a relatively expensive choice in terms of retained bends. Remember that not skipping one of the shared bends means not taking the shortcut in the corresponding vertex gadget, which means putting the corresponding vertex into V^* . So, skipping almost all bends in the edge gadget of $\{u, v\}$ implies not having u or v in V^* , which means respecting the independent set property for the edge $\{u, v\}$.

Neighborhood Gadget. For each vertex v , we construct a *neighborhood gadget* (see Figure 3c). This gadget shares a bend with every vertex gadget corresponding to a vertex of $\text{Adj}(v)$, which is v and the vertices being adjacent to v . These shared bends are on the same height. The vertex gadgets of $\text{Adj}(v)$ appear in some horizontal order in our construction. Say the corresponding vertices in order are $u_1, \dots, u_{|\text{Adj}(v)|}$. Let the shared bends with u_1 and $u_{|\text{Adj}(v)|}$ be b_1 and $b_{|\text{Adj}(v)|}$, respectively, and define t as the distance between b_1 and $b_{|\text{Adj}(v)|}$. We place the first bend (the starting point) of the neighborhood gadget $4/5\delta$ below and $3t$ to the left of b_1 , where t is the distance between b_1 and $b_{|\text{Adj}(v)|}$, and let the second bend be b_1 . Symmetrically, we place the last bend (the end point) of the gadget $4/5\delta$ below and $3t$ to the right of $b_{|\text{Adj}(v)|}$ and let the second last bend be $b_{|\text{Adj}(v)|}$. Between each two bends b_i and b_{i+1} shared with the vertex gadgets of u_i and u_{i+1} for each $i \in \{1, \dots, |\text{Adj}(v)| - 1\}$, we add a zigzag with $2\hat{n}^2 + 1$ bends as in Figure 3c.

▷ **Claim 6.** In a neighborhood gadget, the only shortcuts are (i) the shortcuts skipping only b_i for $i \in \{1, \dots, |\text{Adj}(v)|\}$ and (ii) the shortcuts starting at the first bend or b_i with $i \in \{1, \dots, |\text{Adj}(v)|\}$ and ending at the last bend or b_j with $i < j \in \{1, \dots, |\text{Adj}(v)|\}$ – except for the shortcut starting at the first and ending at the last bend.

In Appendix A, we argue that Claim 6 is correct. Consequently, we can skip almost all bends in a neighborhood gadget if we keep at least one bend of $b_1, \dots, b_{|\text{Adj}(v)|}$. If we skip all of them, we can skip no other bend. So, to avoid high costs, we must not take the shortcut of the vertex gadget of at least one vertex of $\text{Adj}(v)$. This means that we must, for each $v \in V$, add a vertex of $\text{Adj}(v)$ to V^* , which enforces the dominating set property.

Observe that all shared bends are shared between only two polylines – by a vertex gadget and either an edge gadget or a neighborhood gadget. With $2\hat{n}$ inner bends, a vertex gadget provides enough bends that are shared with the edge and neighborhood gadgets as a vertex is contained in at most \hat{n} neighborhoods and has at most $\hat{n} - 1$ incident edges. In the following lemma, we analyze the size of the constructed PBS instance.

► **Lemma 7.** *By our reduction, we obtain from an instance $G = (V, E)$ of MIDS an instance of PBS with n bends such that $n \leq 10c\hat{n}^3$, where $\hat{n} = |V| \geq 2$, $|E| \leq c\hat{n}$ ($c \geq 1$ is constant).*

Proof. To count the bends of the vertex, edge, and neighborhood gadgets without double counting, we charge the shared bends to the vertex gadgets. All vertex gadgets together have $\hat{n}(2\hat{n} + 2)$ bends, all edge gadgets have $\hat{m}(2\hat{n}^2 + 3)$ bends without shared bends, and all neighborhood gadgets have $2\hat{m} \cdot (2\hat{n}^2 + 1) + 2\hat{n}$ bends without shared bends. Summing these values up and using $\hat{m} = |E| \leq c\hat{n}$ yields (for $\hat{n} \geq 2$)

$$n = 2\hat{n}^2 + 2\hat{n} + 2\hat{m}\hat{n}^2 + 3\hat{m} + 4\hat{m}\hat{n}^2 + 2\hat{m} + 2\hat{n} \leq 6c\hat{n}^3 + 2\hat{n}^2 + (4 + 5c)\hat{n} \leq 10c\hat{n}^3. \quad (1)$$

◀

We say a simplification of an instance of PBS obtained by this reduction *corresponds* to an independent and dominating set V' and vice versa if we take all “long” shortcuts in the vertex gadgets except for the ones corresponding to V' and we skip all inner unshared bends in all

edge and neighborhood gadgets, which is possible since V' is independent and dominating. Observe that for each independent and dominating set there is precisely one corresponding simplification (which is also valid acc. to δ).

► **Lemma 8.** *Let V' be a solution for an instance $G = (V, E)$ of MIDS. In the instance (B, \mathcal{L}, δ) of PBS obtained by our reduction, the size of the simplification corresponding to V' is $2\hat{n}(|V'| + c + 2)$, where $\hat{n} = |V|$ and $c \geq 1$ is constant.*

Proof. Only for all $v \in V \setminus V'$, we take the shortcuts in the corresponding vertex gadgets in (B, \mathcal{L}, δ) . This gives us $(\hat{n} - |V'|) \cdot 2 + |V'| \cdot (2 + 2\hat{n}) = 2\hat{n}(1 + |V'|)$ remaining bends in all vertex gadgets combined. In the following, we will count shared bends for the vertex gadgets. We take a “long” shortcut in all of the edge gadgets. This gives us two remaining unshared bends in all edges gadgets ($c\hat{n} \cdot 2$ bends in total). Moreover, we skip all inner unshared bends in all of the neighborhood gadgets ($2\hat{n}$ bends remaining). Altogether, this sums up to $2\hat{n}(|V'| + 1 + c + 1)$. ◀

By Lemma 8, we know that for an optimal solution V^* of an instance of MIDS, the corresponding simplification in the instance (B, \mathcal{L}, δ) of PBS obtained by our reduction has size $2\hat{n}(OPT_{\text{MIDS}} + c + 2)$, where $OPT_{\text{MIDS}} = |V^*|$ and which of course is at least the size OPT_{PBS} of the optimal solution of (B, \mathcal{L}, δ) . We formalize this in the following corollary.

► **Corollary 9.** *For an instance $G = (V, E)$ of MIDS and the instance (B, \mathcal{L}, δ) of PBS obtained by our reduction from G , $OPT_{\text{PBS}} \leq 2\hat{n}(OPT_{\text{MIDS}} + c + 2)$.*

► **Theorem 10.** *PBS is NP-hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of bend points in the polyline bundle.*

Proof. Assume that there is an approximation algorithm \mathcal{A} solving any instance of PBS within a factor of $n^{\frac{1}{3}-\varepsilon}$ for some constant $\varepsilon > 0$ relative to the optimal solution. We can transform any instance $G = (V, E)$ of MIDS, where $\hat{n} = |V|$, $\hat{m} = |E|$, and $OPT_{\text{MIDS}} = |V^*|$, this is the size of an optimal solution, to an instance (B, \mathcal{L}, δ) of PBS using the reduction described above in this section, where $|B| = n$ and the size of an optimal solution is OPT_{PBS} .

Employing \mathcal{A} to solve (B, \mathcal{L}, δ) yields a (simplified) polyline bundle $\mathcal{L}_{\mathcal{A}}$. We denote the number of bends in $\mathcal{L}_{\mathcal{A}}$ by $n_{\mathcal{A}}$ and we know that $n_{\mathcal{A}} \leq OPT_{\text{PBS}} \cdot n^{\frac{1}{3}-\varepsilon}$ for some $\varepsilon > 0$. If all $(2\hat{n}^2 + 1)$ -bend-sequences in all edge and neighborhood gadgets are skipped, we can immediately read an independent dominating vertex set $V' \subseteq V$ from the vertex gadgets where the shortcut is not taken. Otherwise, we replace $\mathcal{L}_{\mathcal{A}}$ such that it corresponds to any maximal independent set $V' \subseteq V$ (which is always an independent and dominating set and can be found greedily in polynomial time). Observe that this can only lower the number of bends compared to a solution not skipping all $(2\hat{n}^2 + 1)$ -bend-sequences in the edge and neighborhood gadgets as in all vertex gadgets together we can skip at most $\hat{n} \cdot 2\hat{n}$ bends.

Using Lemma 8 and Corollary 9, we can state that

$$n^{\frac{1}{3}-\varepsilon} \geq \frac{n_{\mathcal{A}}}{OPT_{\text{PBS}}} \geq \frac{2\hat{n}(|V'| + c + 2)}{2\hat{n}(OPT_{\text{MIDS}} + c + 2)} > \frac{|V'|}{OPT_{\text{MIDS}} + c + 2}, \quad (2)$$

which we can reformulate as $|V'| < n^{\frac{1}{3}-\varepsilon}(OPT_{\text{MIDS}} + c + 2)$. We can assume that $OPT_{\text{MIDS}} > c + 2$ as otherwise we could check all subsets of V of size at most $c + 2$ in polynomial time. Similarly, we can assume that \hat{n} is large enough so that $\hat{n}^{2\varepsilon} > 20c$. Beside this, we apply Lemma 7 and obtain

$$|V'| < 2n^{\frac{1}{3}-\varepsilon}OPT_{\text{MIDS}} \leq 2 \cdot (10c\hat{n}^3)^{\frac{1}{3}-\varepsilon}OPT_{\text{MIDS}} \quad (3)$$

$$< 20c \cdot \hat{n}^{1-3\varepsilon}OPT_{\text{MIDS}} < \hat{n}^{2\varepsilon} \cdot \hat{n}^{1-3\varepsilon}OPT_{\text{MIDS}} = \hat{n}^{1-\varepsilon}OPT_{\text{MIDS}}. \quad (4)$$

Since we know that it is NP -hard to approximate MIDS within a factor of $\hat{n}^{1-\varepsilon}$ for any $\varepsilon > 0$, it follows that \mathcal{A} cannot be a polynomial time algorithm, unless $P = NP$. Or in other words, it is NP -hard to approximate PBS within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. ◀

Currently, we use one polyline per gadget. So, our reduction uses $2\hat{n} + \hat{m}$ polylines. We can reduce the number of polylines to two by connecting all vertex gadgets – one after the other – in arbitrary order by two segments, which gives us the first polyline, and by connecting all edge and neighborhood gadgets in arbitrary order by two segments, which gives us the second polyline. The extra bend between each pair of new segments is placed far away from the construction, e.g. at (∞, ∞) . This never creates new shortcuts for skipping a bend in a vertex gadget or in a neighborhood gadget. Yet, we might create new shortcuts that allow for additionally skipping the first and the last bend of an edge gadget. However, we cannot skip any further bend unless the second or second last bend is skipped, which preserves the functionality of our gadget. For the analysis, this gives us an additive constant of at most $2\hat{n} + \hat{m}$ bends that cannot be skipped, which we can include to Inequalities (2)–(4) in Theorem 10 with the same result to obtain the following corollaries.

▶ **Corollary 11.** *Even for instances of two polylines, PBS is NP -hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of bend points in the polyline bundle.*

▶ **Corollary 12.** *PBS is not fixed-parameter tractable in the number of polylines ℓ .*

4 Bi-criteria Approximation for Polyline Bundle Simplification

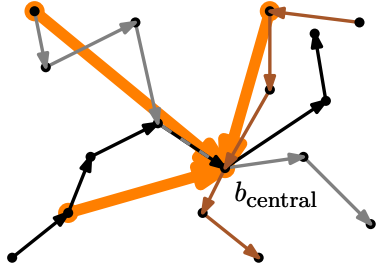
In this section, we describe a bi-criteria approximation algorithm for PBS. Conceptually, a bi-criteria approximation is a generalization of a (classical) approximation where it is allowed to violate a certain constraint by a specific factor. In particular, an algorithm is called a bi-criteria (α, β) -approximation algorithm if it runs in polynomial time and produces a solution of size at most $\alpha \cdot OPT$ while relaxing the constraint by a factor of β .

In our particular problem PBS, we relax the error bound δ . In Section 3, we have shown that there is no bi-criteria $(n^{\frac{1}{3}-\varepsilon}, 1)$ -approximation algorithm for PBS for any $\varepsilon > 0$ unless $P = NP$. This strong inapproximability comes from the high sensitivity towards choices of keeping or discarding single bends, which is modulated by the given value of δ . By making a bad choice we cannot take (helpful) shortcuts that have a distance just a little greater than the given distance threshold δ to the original sub-polyline. This can be overcome by relaxing the constraint slightly. In particular, we show that allowing a constraint violation by a factor of $\beta = 2$, we can design an efficient algorithm with an approximation guarantee of $\alpha \in \mathcal{O}(\log(\ell + n))$. For an overview of our algorithm see Fig. 6.

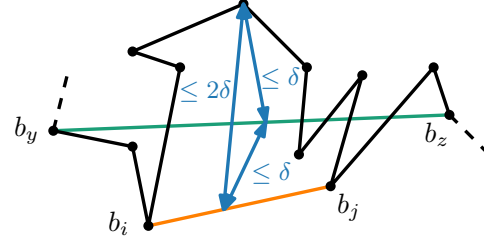
The key building block of our algorithm is a connection between PBS and a certain geometric set cover problem, which we call *star cover problem*. The star cover problem models the aspect of shortcutting polylines by few bend points but does not take into account consistency. We argue, however, that approximate solutions to the star cover problem can be post-processed to form consistent PBS solutions by slightly violating the error threshold δ .

Star Cover Problem

Next, we introduce the *star cover* problem, which is a special type of the set cover problem defined over instances of PBS. Informally spoken, a star is a bend together with some incident shortcut segments. These shortcut segments span sets of original segments of the polylines. To this end, we first direct each polyline $L \in \mathcal{L}$ in a given PBS instance (B, \mathcal{L}, δ) arbitrarily but ensuring that all (shortcut) segments of L are oriented in the same direction. Then a star consists of a set of incoming shortcuts of some bend; see Fig. 4 for an example.



■ **Figure 4** Example of a *star* (in orange) around a bend b_{central} , which lies on three polylines. Each polyline was assigned an arbitrary direction indicated by arrow heads.



■ **Figure 5** Example of the maximum Fréchet distance between a line segment (b_i, b_j) and its corresponding sub-polyline if there is a valid shortcut (b_y, b_z) going over b_i and b_j .

► **Definition 13 (Star).** A star is the combination of a bend $b_{\text{central}} \in B$ and, for each polyline $L \in \mathcal{L}$ that contains b_{central} , one or zero incoming shortcut segments (according to δ).

We say a star s covers a segment–polyline pair (e, L) , if s contains for L a shortcut $(b_{\text{outer}}, b_{\text{central}})$ and e lies on L between b_{outer} and b_{central} . Our goal is to find a small set of stars that cover all segment–polyline pairs. We denote the set of all segment–polyline pairs in the input by \mathcal{U} and the subset of pairs covered by a particular star s by \mathcal{U}_s . Then the *star cover problem* is defined as follows.

► **Definition 14 (Star Cover).** A star cover C is a set of stars, such that $\bigcup_{s \in C} \mathcal{U}_s = \mathcal{U}$, i.e. all segment–polyline pairs are covered. The star cover problem (abbreviated by STCO) asks for a minimum size star cover.

Relationship between Instances of Polyline Bundle Simplification and Star Cover

Next, we investigate the relationship between an instance of STCO and its corresponding instance of PBS. We argue that every (optimal) solution for PBS can be decomposed into a star cover. Hence an optimal STCO yields a lower bound for an optimal PBS solution.

► **Lemma 15.** The size OPT_{STCO} of an optimal solution of any instance of STCO obtained from an instance (B, \mathcal{L}, δ) of PBS is bounded by $OPT_{\text{STCO}} \leq OPT_{\text{PBS}}$, where OPT_{PBS} is the size of an optimal solution of (B, \mathcal{L}, δ) .

Proof. Consider an optimal solution B^* of (B, \mathcal{L}, δ) . From the simplified polyline bundle induced by B^* , we can get a star cover for any instance of STCO obtained from (B, \mathcal{L}, δ) by iteratively adding a star in the following way until there are only isolated bends. Get a star s by taking any connected bend $b_{\text{central}} \subseteq B^*$ as a central bend and the bends that precede b_{central} on each of the simplified polylines as its outer bends. Remove the segment–polyline pairs covered by s from our simplified polyline bundle. Repeat this until there are no more segment–polyline pairs. The obtained star cover has at most $|B^*|$ stars and at least as many stars as a minimum star cover. So, $OPT_{\text{STCO}} \leq OPT_{\text{PBS}}$. ◀

Approximation for the Star Cover Problem

We can compute an approximate solution for STCO by employing the classical greedy algorithm [15] for set cover, which iteratively selects the set with the most uncovered elements until all elements are covered. However, if applied naively, the running time would be exponential in the size of the PBS instance as the number of stars might be in the order

of $n \cdot 2^\ell$. We observe, however, that it suffices to consider only maximal stars (containing on each polyline incident to the central bend the incoming shortcut that covers the largest number of segments). As there are only n maximal stars, this guarantees polynomial running time.

► **Lemma 16.** *We can compute an $\mathcal{O}(\log(t+w))$ -approximation for an instance of STCO obtained from an instance (B, \mathcal{L}, δ) in time $\mathcal{O}(\ell n^3)$, where t is the maximum number of polylines any bend point occurs in and w is the maximum number of segments any valid shortcut (according to δ) can skip.*

Proof. There is a polynomial time greedy algorithm that yields an $\mathcal{O}(\log m)$ approximation for the set cover problem, where m is the size of the largest set in the given collection of subsets of the universe [15]. The greedy algorithm works as follows. While there are uncovered elements from the universe, add the set with the largest number of uncovered elements to the set cover. In an instance of STCO, this m is the maximum number of segment–polyline pairs $\max_{\text{star } s} |\mathcal{U}_s|$ a single star can cover. If the central bend point of a star lies in at most t polylines, the star contains at most t shortcut segments, and each of which covers at most w segments, hence we have $m = tw$. Observe that $\mathcal{O}(\log(tw)) = \mathcal{O}(\log(t+w))$.

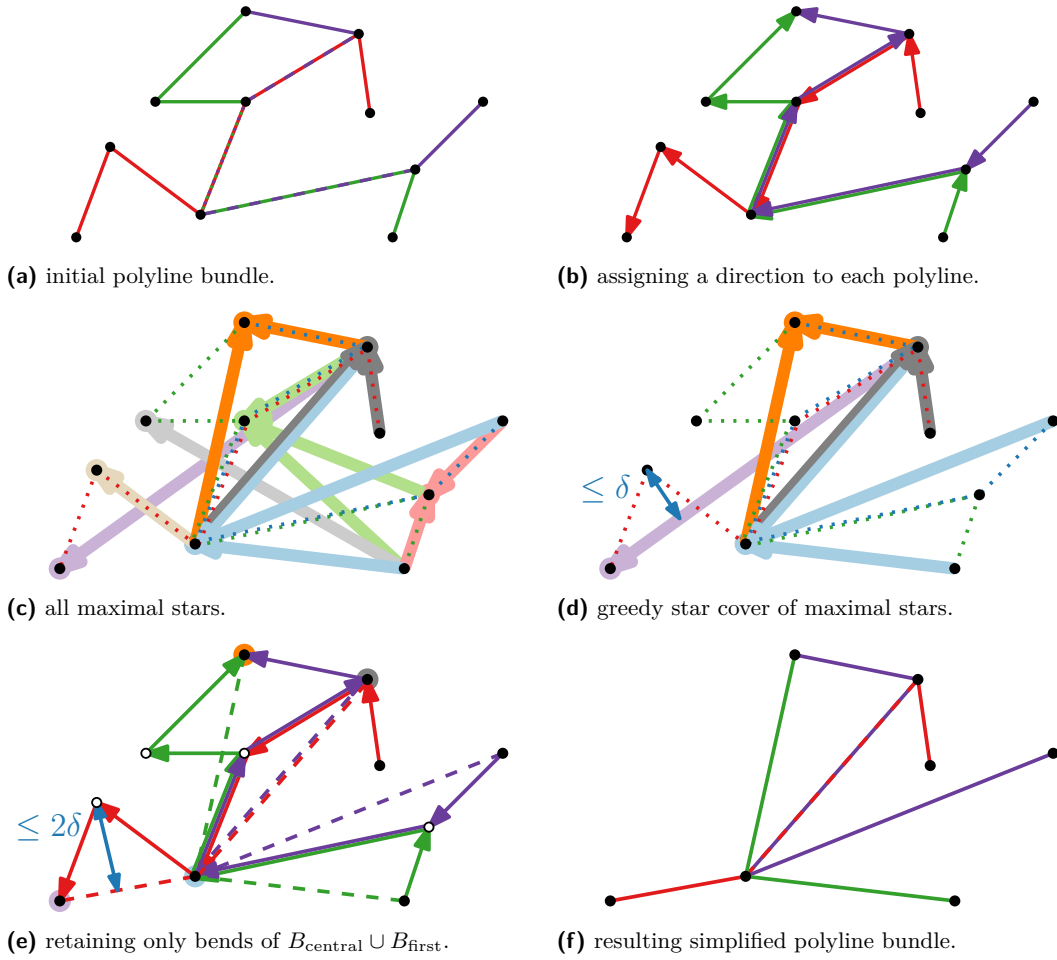
Having settled the $\mathcal{O}(\log(t+w))$ approximation ratio, it remains to prove the polynomial running time. Using the algorithm by Imai and Iri [14] independently for each polyline, we can find all (maximal) shortcuts for every bend on every polyline in time $\mathcal{O}(\ell n^3)$. Combining these shortcuts at every bend gives us all n maximal stars in time $\mathcal{O}(\ell n)$. For each star, we also save the number of segment–polyline pairs it covers and, to each segment–polyline pair, we link all stars it appears in. Both can be done in time $\mathcal{O}(\ell n^2)$. As long as there are uncovered segments, we find the star with the most uncovered segments and then update the number of uncovered segments for the other stars. This can be done in $\mathcal{O}(\ell n^2)$ time in total as well. ◀

Relationship between Star Covers and Solutions of Polyline Bundle Simplification

While a solution for PBS can be directly converted into a star cover as argued above, the converse is more intricate. The shortcuts contained in the selected stars may be overlapping or nested along a polyline, that is, bends skipped by one shortcut may be end points of another shortcut in the set. Moreover, shared parts of different polylines may be shortcut differently. Therefore consistency is not guaranteed. We explain how to derive from a star cover solution a solution for its corresponding instance of PBS. Some of the shortcuts of the STCO solution are replaced by shorter shortcuts in order to integrate some intermediate point to the PBS solution. Lemma 17 states that those newly introduced shortcuts can be at most 2δ away from the original polyline. The situation described there is depicted in Fig. 5. It follows immediately from a lemma by Agarwal et al. ([1], Lemma 3.3).

► **Lemma 17.** *Given a polyline $L = (b_1, b_2, \dots, b_{|L|})$ and a distance threshold δ . If there are $y, z \in \mathbb{N}$ with $1 \leq y < z \leq |L|$ and $d_{\text{Fréchet}}((b_y, b_z), L[b_y, \dots, b_z]) \leq \delta$ (i.e., segment (b_y, b_z) is a valid shortcut), then for any $i, j \in \mathbb{N}$ with $y \leq i < j \leq z$, $d_{\text{Fréchet}}((b_i, b_j), L[b_i, \dots, b_j]) \leq 2\delta$.*

Equipped with this lemma, we now discuss the actual transformation from a STCO solution to a PBS solution. The idea is to keep, beside the starting points of all polylines, only the central bend points of the selected stars while dropping their leaves. This is closely tied with the fact that we minimize the number of stars while ignoring their degree in the algorithm. The main insight here is that the shortcuts induced by this augmented point set still have a small distance to the original polylines.



■ **Figure 6** Example of our bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$ -approximation algorithm for PBS.

► **Lemma 18.** *Let C be a star cover for an instance of STCO obtained from an instance (B, \mathcal{L}, δ) of PBS. If C is an α -approximation for its instance of STCO, a bi-criteria $(\alpha + 1, 2)$ -approximation for (B, \mathcal{L}, δ) can be computed in time $\mathcal{O}(n)$ from C .*

Proof. Let B_{central} be the set of central bends of the stars in C and let B_{first} be the set of first bends of all polylines from \mathcal{L} . We return $B_{\text{central}} \cup B_{\text{first}}$ as the bi-criteria approximate solution. Clearly, we can construct this set in time $\mathcal{O}(n)$. According to Lemma 15, $\text{OPT}_{\text{STCO}} \leq \text{OPT}_{\text{PBS}}$, where OPT_{PBS} is the size of the optimal solution of (B, \mathcal{L}, δ) and OPT_{STCO} is the size of the optimal solution of the instance of STCO where C is an approximation for. We conclude

$$|B_{\text{central}} \cup B_{\text{first}}| \leq \alpha \text{OPT}_{\text{STCO}} + \text{OPT}_{\text{PBS}} \leq (\alpha + 1) \text{OPT}_{\text{PBS}}. \quad (5)$$

Let \mathcal{L}' be the polyline bundle induced by $B_{\text{central}} \cup B_{\text{first}}$. It remains to prove that the Fréchet distance between each induced segment of each polyline in \mathcal{L}' and its corresponding sub-polyline in \mathcal{L} is at most 2δ . Consider any segment (b_i, b_j) of any polyline $L' \in \mathcal{L}'$ corresponding to a polyline $L \in \mathcal{L}$ such that b_i precedes b_j in L . There is a star s in C that covers all segments of $L[b_i, b_j]$. Clearly, all segments of $L[b_i, b_j]$ are covered by the stars of C and if there was no single star s covering all segments of $L[b_i, b_j]$, but multiple stars, there would be another central bend of a star between b_i and b_j on L and, in L' , (b_i, b_j) would

not be a segment. The central bend b_{central} of s succeeds b_j or is equal to b_j as otherwise s would not cover all of $L[b_i, b_j]$. Accordingly, the outer bend b_{outer} of s on L precedes b_i or is equal to b_i as otherwise s would not cover all of $L[b_i, b_j]$. By the definition of a star, we know that $d_{\text{Fréchet}}((b_{\text{outer}}, b_{\text{central}}), L[b_{\text{outer}}, b_{\text{central}}]) \leq \delta$. By Lemma 17, it follows that $d_{\text{Fréchet}}((b_i, b_j), L[b_i, b_j]) \leq 2\delta$. ◀

Bi-criteria Approximation for Polyline Bundle Simplification via Star Cover

Using the previous lemmas, we obtain the main theorem of this section. It is reasonable to assume that the number ℓ of polylines is polynomial in n in practically relevant settings. Hence, we essentially obtain an exponential improvement over the complexity-theoretic lower bound $n^{\frac{1}{3}-\varepsilon}$ if we allow the slight violation of the error bound.

► **Theorem 19.** *There is a bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$ -approximation algorithm for PBS running in time $\mathcal{O}(\ell n^3)$, where ℓ is the number of polylines and n is the number of bend points in the polyline bundle.*

Proof. We describe a (kind of) approximation-preserving reduction from PBS to STCO, which can be realized as a bi-criteria approximation algorithm. Its steps are depicted in Fig. 6. Given an instance (B, \mathcal{L}, δ) of PBS, where we let the size of the optimal solution be OPT_{PBS} , we assign an arbitrary direction to each $L \in \mathcal{L}$. This yields our corresponding instance of STCO. For this corresponding instance of STCO, compute an $\mathcal{O}(\log(t + w))$ approximation star cover C . We can do this in time $\mathcal{O}(\ell n^3)$ according to Lemma 16. According to Lemma 18, we can compute a bi-criteria $(\mathcal{O}(\log(t + w)), 2)$ -approximation for (B, \mathcal{L}, δ) from C in $\mathcal{O}(n)$ time. Since $t \leq \ell$ and $w \leq n$, this is also a bi-criteria $(\mathcal{O}(\log(\ell + n)), 2)$ -approximation. ◀

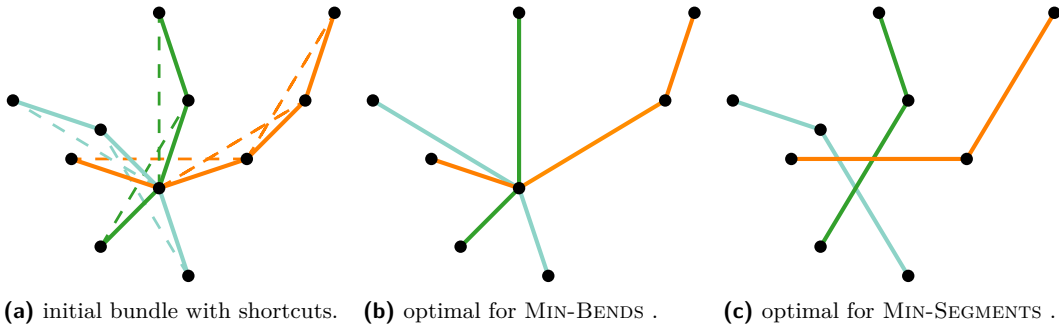
5 Fixed-Parameter Tractability

A brute force approach is checking for every subset of the bend set B in time $\mathcal{O}(\ell \cdot n)$ whether it is a valid simplification and accepting the one with the smallest number of bends or segments. Consequently, the runtime of this approach is $\mathcal{O}(2^n \cdot \ell \cdot n)$. When considering fixed-parameter tractability, investigating parameters of the input is a natural choice. According to Corollary 12, PBS is not fixed-parameter tractable (FPT) in the number of polylines ℓ . However, PBS is FPT in the number of shared bends, i.e., bends contained in more than one polyline. We denote the set of those bends by B_{shared} and we let $k := |B_{\text{shared}}|$.

► **Theorem 20.** *PBS is FPT in the number of shared bends k . There is an algorithm solving PBS in time $\mathcal{O}(2^k \cdot \ell \cdot n^2 + \ell n^3)$.*

Proof. We describe an algorithm that solves PBS in time $\mathcal{O}(2^k \cdot \ell \cdot n^3)$. Given an instance (B, \mathcal{L}, δ) of PBS, the first step is to compute, for each $L \in \mathcal{L}$, its shortcut graph G_L using the algorithm by Imai and Iri [5]. This can be done in time $\mathcal{O}(\ell \cdot n^3)$. For a polyline L and a distance threshold δ , the *shortcut graph* is the directed graph that has the bends of L as its vertices and has an edge from u to v if $d_{\text{Fréchet}}((u, v), L[u, \dots, v]) \leq \delta$, this is, if there is a shortcut from u to v in L . Given the shortcut graph G_L of L , the vertices of a shortest path in G_L from the first bend of L to the last bend of L define an optimal simplification of L .

The second step is to iterate over all subsets $B' \subseteq B_{\text{shared}}$ and check if B' is part of an optimal solution. Before the first iteration, we initialize a variable $n_{\text{min}} = \infty$ and we will save the current best solution by \mathcal{S}_{min} . Then, in each iteration, we temporarily remove from all shortcut graphs G_L all vertices $B_{\text{not-contained}} = B_{\text{shared}} - B'$ and all edges that correspond



■ **Figure 7** Example of three polylines, where the goals MIN-SEGMENTS and MIN-BENDS differ.

to a shortcut skipping a bend in B' . Clearly, removing $B_{\text{not-contained}}$ can be performed in $\mathcal{O}(n^2)$ time for each G_L . For the removal of the edges in G_L , note that we can sort the list of bends $B_{\text{not-contained}}$ and the list of all edges (defined by their endpoints) alphanumerically by the occurrence of the bends within the polyline L . If we traverse both lists simultaneously in ascending order, we remove an edge if and only if its endpoint-bends come before and after the currently considered bend from $B_{\text{not-contained}}$. Therefore, the removal operations can be performed in $\mathcal{O}(n^2)$ time per G_L .

If some shortcut graph becomes disconnected by these removal operations, we continue with the next iteration. Otherwise, we take the bends of a shortest path from the first to the last bend in each reduced version of G_L . Together they define a simplification \mathcal{S} of our PBS instance. If the number $n_{\mathcal{S}}$ of bends in \mathcal{S} is less than n_{\min} , we set $n_{\min} = n_{\mathcal{S}}$ and $\mathcal{S}_{\min} = \mathcal{S}$. After the iteration process, we return \mathcal{S}_{\min} . Since we have 2^k subsets of B_{shared} and each iteration can be performed in $\mathcal{O}(\ell \cdot n^2)$ time, the running time of the algorithm is in $\mathcal{O}(2^k \cdot \ell \cdot n^2 + \ell n^3)$.

It remains to prove that \mathcal{S}_{\min} is in the end an optimal solution of our input instance of PBS. First note that our algorithm always returns some polyline simplification because for $B' = B_{\text{shared}}$, we do not get a disconnected G_L after the removal operations.

The returned solution is valid because the shared bends of B' are taken in all simplified polylines (they cannot be skipped) and the other shared bends are skipped in all simplified polylines. Our algorithm finds the minimum size solution because in one iteration it considers $B' = B^* \cap B_{\text{shared}}$, where B^* is the set of retained bends of an optimal solution. Moreover, an optimal solution cannot have fewer bends occurring in only one polyline L than our algorithm since this would imply a shorter shortest path within the reduced version of G_L . ◀

6 Conclusion and Outlook

We have generalized the well-known problem of polyline simplification from a single polyline to polyline bundles. Although in the case of one polyline, efficient algorithms have long been known, it turned out that simplifying two or more polylines is a problem that is indeed hard to approximate within a factor of $n^{\frac{1}{3}-\varepsilon}$ for any $\varepsilon > 0$. However, if we relax the constraint on the maximum Fréchet distance between original and simplified polyline by a factor of 2, we can overcome this strong inapproximability bound. Moreover, we can find an optimal simplification quickly if we have only a small number of shared bends since the problem of polyline bundle simplification is fixed-parameter tractable (FPT) in this parameter.

Based on our results, there are many possible directions for future research.

- Our current bi-criteria approximation guarantee is logarithmic in the number of polylines ℓ plus the number of bend points n . In most practical application, ℓ is smaller than n or at most polynomial in n . From a theoretical perspective, however, it might be interesting to get rid off the dependency on ℓ in the bi-criteria approximation in order to get improvements for the case where ℓ is significantly larger than n .
- As a distance measure, we employed the Fréchet distance, which we consider to be more natural and intuitive than the Hausdorff distance when comparing polylines. However, the Hausdorff distance is sometimes used in classical polyline simplification as well. Our hardness results also apply to the Hausdorff distance, but our bi-criteria approximation algorithm fails since Lemma 17 is not true for the Hausdorff distance. One might consider PBS using the Hausdorff distance or other (even non-segment-wise) distance measurements.
- In our generalization to bundles of polylines, we aim for a minimizing the number of retained bends (MIN-BENDS). However, minimizing the number of retained segments (MIN-SEGMENTS) is an alternative goal, which also generalizes the classical minimization problem for a single polyline. Optimal simplifications for both goals may differ; see Fig. 7. Our hardness and FPT results also apply for the goal MIN-SEGMENTS. However, it is not clear how to obtain a similar result for the bi-criteria approximability.
- For practical purposes, the scalability of the proposed bi-criteria approximation algorithm, the FPT algorithm, and possibly new heuristics should be investigated on real-world data.

References

- 1 Pankaj K. Agarwal, Sarel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005. doi:10.1007/s00453-005-1165-y.
- 2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.
- 3 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG'14)*, pages 367–376, 2014. doi:10.1145/2582112.2582144.
- 4 Maike Buchin, Bernhard Kilgus, and Andrea Kölsch. Group diagrams for representing trajectories. In *Proceedings of the 11th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 1–10, 2018.
- 5 W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996. doi:10.1142/S0218195996000058.
- 6 Marc de Berg, Marc J. van Kreveld, and Stefan Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(1):243–257, 1998. doi:10.1559/152304098782383007.
- 7 David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.
- 8 Regina Estkowski and Joseph SB Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG'01)*, pages 40–49. ACM, 2001. doi:10.1145/378583.378612.
- 9 Stefan Funke, Thomas Mendel, Alexander Miller, Sabine Storandt, and Maria Wiebe. Map simplification with topology constraints: Exactly and in practice. In *Proceedings of the 19th Workshop on Algorithm Engineering and Experiments (ALENEX'17)*, pages 185–196. SIAM, 2017.

- 10 Michael Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991. doi:10.1007/BFb0020793.
- 11 Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2.
- 12 Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Roadrunner: improving the precision of road network inference from GPS trajectories. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 3–12, 2018.
- 13 John Hershberger and Jack Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling (SDH'92)*, pages 134–143, 1992.
- 14 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In Godfried T. Toussaint, editor, *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.
- 15 David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974. doi:10.1016/S0022-0000(74)80044-9.
- 16 Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972. doi:10.1016/S0146-664X(72)80017-0.
- 17 Marc J. 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. URL: <https://journals.carleton.ca/jocg/index.php/jocg/article/view/415>.

A Omitted Content of Section 3

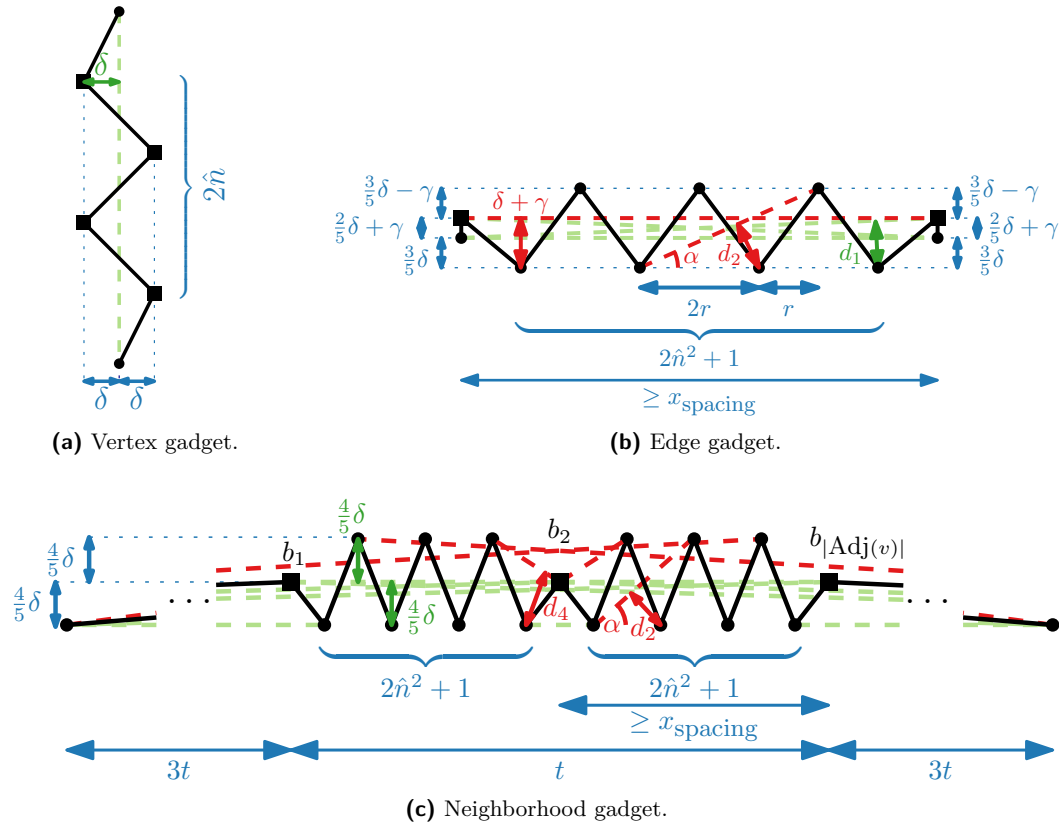
It remains to show the correctness of Claim 5 and Claim 6, which we use in our reduction from MIDS to PBS. Our gadgets are depicted in Fig. 3. For convenience, we provide by Fig. 8 a copy of them with some additional details, to which we will refer in this appendix. For example r is the x-distance between two consecutive (inner) vertices in an edge and a neighborhood gadget (if a gadget is rotated, the distance is measured along the corresponding rotated axis). We know that $r \geq x_{\text{spacing}}/(2\hat{n}^2 + 2)$.

▷ **Claim 5.** In an edge gadget, there are precisely three long shortcuts. These are (i) from the first to the last bend, (ii) from the first to the second last bend, and (iii) from the second to the last bend. Beside these three shortcuts, there are ≤ 4 more shortcuts, which skip only the second and the second last bend (and possibly also the third and third last bend). There is no shortcut not skipping one of the shared bends, i.e., the second or the second last bend.

In (i), both of the shared bends, these are the second and the second last, are skipped and we can take the “long” shortcut from the first to the last bend because the line segment between them is horizontal and has y-distance $3/5\delta$ or $2/5\delta + \gamma$ or δ to all inner bends. In (ii), the most critical part is the distance d_1 between the third last bend and the straight-line segment from the first to the second last bend (see Figure 3b). It is

$$d_1 \leq \frac{3}{5}\delta + \left(\frac{2}{5}\delta + \gamma\right) - \frac{\frac{2}{5}\delta + \gamma}{2\hat{n}^2 + 2} \leq \delta + \gamma - \frac{\frac{2}{5} \cdot \frac{10\hat{n}^2 + 5}{2}\gamma + \gamma}{2\hat{n}^2 + 2} = \delta. \quad (6)$$

Observe that (iii) is the same as (ii) but mirrored. If neither the second nor the second last bend is skipped, i.e., if u and v are in the set V^* , then we cannot cut short anything in this gadget. Clearly, we cannot take a “long” shortcut from the second to the second last bend because the lower row of inner bends has distance $\delta + \gamma$ from the potential shortcut segment.



■ **Figure 8** Some additional details to Fig. 3.

Moreover, we cannot take a “short” shortcut from a bend of the lower row to a bend of the upper row or the other way around. If we would aim to skip two inner bends, the distance d_2 (see Figure 3b) from an inner bend to the shortcut segment would have to be at most δ . However, it is

$$d_2 = \sin \alpha \cdot 2r = \sin \arctan \frac{\frac{8}{5}\delta}{3r} \cdot 2r = \frac{\frac{8\delta}{15r}}{\sqrt{\left(\frac{8\delta}{15r}\right)^2 + 1}} \cdot 2r = \frac{16\delta r}{\sqrt{(8\delta)^2 + (15r)^2}}, \quad (7)$$

where

$$r \geq \frac{x_{\text{spacing}}}{2\hat{n}^2 + 2} \geq \frac{(2\hat{n}^2 + 2)3\delta}{2\hat{n}^2 + 2} = 3\delta, \quad (8)$$

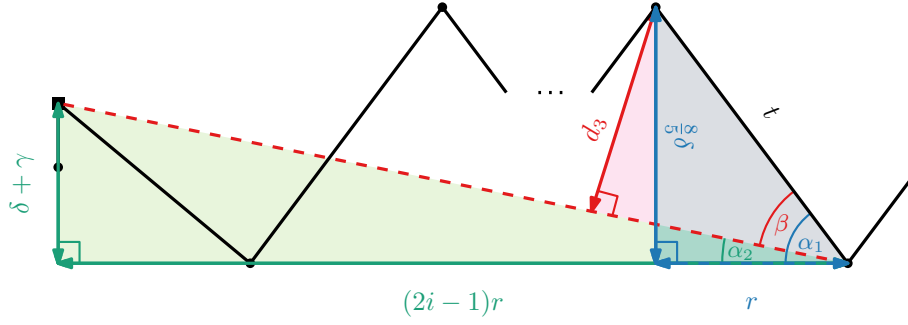
and hence,

$$d_2 \geq \frac{48\delta^2}{\sqrt{64\delta^2 + 2025\delta^2}} = \frac{48}{\sqrt{2089}}\delta = 1.0502 \dots \delta. \quad (9)$$

Observe that this becomes even greater if we aim for skipping four or more bends or if we start or end at one of the two shared bends. To make this clearer, we explicitly consider the latter case where a potential shortcut would start at the second bend and end at the $(2i + 1)$ -th bend. This situation is depicted in Fig. 9. If it was a valid shortcut, d_3 would be less than or equal to δ . Since d_3 is inside a rectangular triangle, its length is

$$d_3 = t \cdot \sin \beta, \quad (10)$$

35:18 Simplification of Polyline Bundles



■ **Figure 9** The potential shortcut segment in an edge gadget from the second bend to an inner bend (the $(2i + 1)$ -th bend) is dashed in red. However, $d_3 > \delta$ makes it no valid shortcut segment.

where t is inside another rectangular triangle with legs of length r and $8/5\delta$, so

$$t = \sqrt{r^2 + \left(\frac{8}{5}\delta\right)^2}. \quad (11)$$

We can determine β via the angles α_1 and α_2 as

$$\beta = \alpha_1 - \alpha_2 = \arctan \frac{\frac{8}{5}\delta}{r} - \arctan \frac{\delta + \gamma}{(2i - 1)r}. \quad (12)$$

In the arctan-functions, all parameters are positive, so they live in the range $[0, \pi/2)$. Hence, β lives in the range $(-\pi/2, \pi/2)$. In this range, the sin-function is monotonously increasing. Therefore, to give a lower bound on d_3 , we can use a lower bound on $\sin \beta$ by specifying a lower bound on β . Since $i \geq 2$, $\gamma < \delta/(5\hat{n}^2)$ and $\hat{n} \geq 1$, we state that

$$\beta = \alpha_1 - \alpha_2 > \arctan \frac{\frac{8}{5}\delta}{r} - \arctan \frac{\frac{6}{5}\delta}{3r} = \arctan \frac{8}{5r'} - \arctan \frac{2}{5r'}, \quad (13)$$

where $r' = r/\delta$. A lower bound on t is

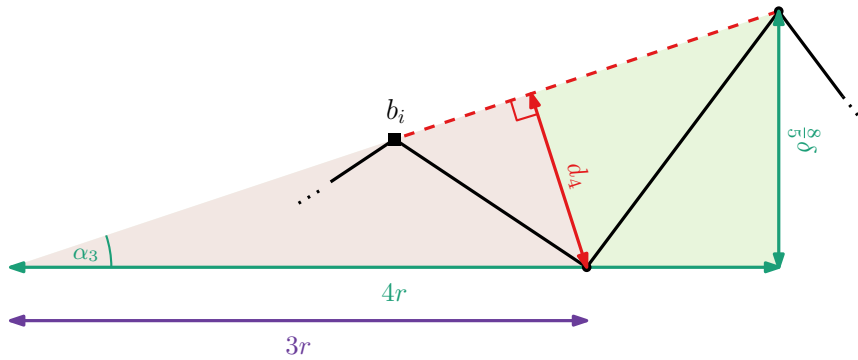
$$t = \sqrt{(r'\delta)^2 + \left(\frac{8}{5}\delta\right)^2} > r'\delta. \quad (14)$$

So, we can get a lower bound on d_3 by

$$d_3 = t \cdot \sin \beta > r' \underbrace{\sin \left(\arctan \frac{8}{5r'} - \arctan \frac{2}{5r'} \right)}_{c(r')} \cdot \delta. \quad (15)$$

To prove that d_3 is always greater than δ , it suffices to show that the prefactor $c(r')$ is equal to or greater than 1 for all possible values of r' . We reformulate $c(r')$ using well-known trigonometric identities:

$$\begin{aligned} c(r') &= r' \sin \left(\arctan \frac{8}{5r'} - \arctan \frac{2}{5r'} \right) \\ &= \frac{6}{\sqrt{25 + \frac{68}{r'^2} + \frac{256}{25r'^4}}} \end{aligned} \quad (16)$$



■ **Figure 10** The potential shortcut segment in a neighborhood gadget from a bend b_i to an inner bend is dashed in red. However, $d_4 > \delta$ makes it no valid shortcut segment.

For $r' = 3$, this is $c(r') = 1.0495\dots$ and, from Equation (16), it is easy to see that $c(r')$ is even greater for $r' > 3$. Thus, we conclude that $d_3 > \delta$ always holds.

It remains to consider potential shortcuts starting or ending at the first or the last bend. Clearly, skipping only the second or second last bend is always possible. Skipping the second and the third bend or skipping the second last and the third last bend may sometimes be possible depending on how much the edge gadget is stretched horizontally. However, according to the previous analysis, skipping more bends is not possible since the distance between the potential shortcut segment and the bend before the end point of the potential shortcut is at least d_3 .

▷ **Claim 6.** In a neighborhood gadget, the only shortcuts are (i) the shortcuts skipping only b_i for $i \in \{1, \dots, |\text{Adj}(v)|\}$ and (ii) the shortcuts starting at the first bend or b_i with $i \in \{1, \dots, |\text{Adj}(v)|\}$ and ending at the last bend or b_j with $i < j \in \{1, \dots, |\text{Adj}(v)|\}$ – except for the shortcut starting at the first and ending at the last bend.

Clearly, the shortcuts (i) for skipping any b_i (or exactly one neighbor of b_i) are valid and there is no shortcut from the first to the last bend since the potential shortcut segment has distance $8/5\delta$ to the upper row of bends. In (ii), there clearly is a shortcut if we start at any b_i and end at any b_j . If we start at some b_i and end at the last bend, observe that, in the most extreme case, the segment from b_1 to the last bend has a y-distance to the upper row of

$$\frac{4}{5}\delta + \frac{t}{4t} \cdot \frac{4}{5}\delta = \delta \tag{17}$$

when it passes $b_{|\text{Adj}(v)|}$ in x-dimension. Thus, this shortcut is valid and the same holds for the shortcuts from the first bend to some b_j .

It remains to argue that there are no more shortcuts. A shortcut starting and ending at a bend on the upper or lower row is not possible because it would either be a horizontal segment, which has distance $8/5\delta$ to the other row, or the distance to some bend in between would be at least d_2 , which we have shown to be greater than δ in Equations (7)–(9). It is easy to see that there is no shortcut starting at the first bend and ending at some inner bend of the upper or lower row. The same holds true for shortcuts starting at some inner bend of the upper or lower row and ending at the last bend.

Moreover, a shortcut segment starting (ending) at some b_i for $i \in \{1, \dots, |\text{Adj}(v)|\}$ and skipping one bend would have a distance of d_4 to this bend as depicted in Fig. 10. Since d_4

35:20 Simplification of Polyline Bundles

is inside a rectangular triangle, we can determine d_4 by

$$d_4 = 3r \cdot \sin \alpha_3, \quad (18)$$

where α_3 is in another rectangular triangle and thus can be determined by

$$\alpha_3 = \arctan \frac{\frac{8}{5}\delta}{4r} = \arctan \frac{2}{5r'}. \quad (19)$$

Putting them together, we get

$$d_4 = 3r'\delta \cdot \sin \arctan \frac{2}{5r'} = 3r'\delta \frac{\frac{2}{5r'}}{\sqrt{1 - \frac{4}{25r'^2}}} = \frac{6}{\sqrt{25 - \frac{4}{r'^2}}}\delta. \quad (20)$$

For $r' = 3$, this is $1.2108 \dots \delta$ and again, for $r' > 3$, d_4 is even greater.

If we skip more than one inner bend, the distance to the last skipped bend becomes only greater. Hence, we conclude that Claim 6 is correct.

Quantum Algorithm for Finding the Optimal Variable Ordering for Binary Decision Diagrams

Seiichiro Tani 

NTT Communication Science Laboratories, NTT Corporation, Atsugi, Japan
seiichiro.tani.cs@hco.ntt.co.jp

Abstract

An ordered binary decision diagram (OBDD) is a directed acyclic graph that represents a Boolean function. Since OBDDs have many nice properties as data structures, they have been extensively studied for decades in both theoretical and practical fields, such as VLSI (Very Large Scale Integration) design, formal verification, machine learning, and combinatorial problems. Arguably, the most crucial problem in using OBDDs is that they may vary exponentially in size depending on their variable ordering (i.e., the order in which the variables are to be read) when they represent the same function. Indeed, it is NP hard to find an optimal variable ordering that minimizes an OBDD for a given function. Friedman and Supowit provided a clever deterministic algorithm with time/space complexity $O^*(3^n)$, where n is the number of variables of the function, which is much better than the trivial brute-force bound $O^*(n!2^n)$. This paper shows that a further speedup is possible with quantum computers by presenting a quantum algorithm that produces a minimum OBDD together with the corresponding variable ordering in $O^*(2.77286^n)$ time and space with an exponentially small error probability. Moreover, this algorithm can be adapted to constructing other minimum decision diagrams such as zero-suppressed BDDs.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Data structures design and analysis

Keywords and phrases Binary Decision Diagram, Variable Ordering, Quantum Algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.36

Related Version A full version of the paper is available at <https://arxiv.org/abs/1909.12658>.

1 Introduction

1.1 Background

Ordered binary decision diagrams

The ordered binary decision diagram (OBDD) is one of the data structures that have been most often used for decades to represent Boolean functions in practical situations, such as VLSI design, formal verification, optimization of combinatorial problems, and machine learning, and it has been extensively studied from both theoretical and practical standpoints (see standard textbooks and surveys, e.g., Refs. [8, 12, 7]). Moreover, many variants of OBDDs have been invented to more efficiently represent data with properties observed frequently in specific applications. More technically speaking, OBDDs are directed acyclic graphs that represent Boolean functions and also known as special cases of oblivious read-once branching programs in the field of complexity theory. The reason for OBDDs' popularity lies in their nice properties – they can be uniquely determined up to isomorphism for each function once *variable ordering* (i.e., the order in which to read the variables) is fixed and, thanks to this property, the equivalence of functions can be checked by just testing the isomorphism between the OBDDs representing the functions. In addition, binary operations such as AND and OR between two functions can be performed efficiently over the OBDDs representing those functions [2]. Since these properties are essential in many applications, OBDDs



© Seiichiro Tani;

licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 36; pp. 36:1–36:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have gathered much attention from various research fields. To enjoy these nice properties, however, we actually need to address a crucial problem, which is that OBDDs may vary exponentially in size depending on their variable ordering. For instance, a Boolean function $f(x_1, \dots, x_{2n}) = x_1x_2 + x_3x_4 + \dots + x_{2n-1}x_{2n}$ has a $(2n + 2)$ -sized OBDD for the ordering (x_1, \dots, x_{2n}) and a 2^{n+1} -sized OBDD for the ordering $(x_1, x_3, \dots, x_{2n-1}, x_2, x_4, \dots, x_{2n})$ [8, Sec. 8.1] (see Figure 1 for the case where $n = 6$). This is not a rare phenomenon; it could happen in many concrete functions that one encounters. Thus, since the early stages of OBDD research, one of the most central problems has been how to find an optimal variable ordering, i.e., one that minimizes OBDDs. Since there are $n!$ permutations over n variables, the brute-force search requires at least $n! = 2^{\Omega(n \log n)}$ time to find an optimal variable ordering. Indeed, finding an optimal variable ordering for a given function is an NP hard problem (see a short survey in the full paper [11]).

To tackle this high complexity, many heuristics have been proposed to find an optimal variable ordering or a relatively good one. These heuristics work well for Boolean functions appearing in specific applications since they are based on very insightful observations, but they do not guarantee a worst-case time complexity lower than that achievable with the brute-force search. The only algorithm with a much lower worst-case time complexity bound, $O^*(3^n)$ time ($O^*(\cdot)$ hides a polynomial factor), than the brute-force bound $O^*(n!2^n)$ for *all* Boolean functions with n variables was provided by Friedman and Supowit [5], and that was almost thirty years ago!

1.2 Our Results

In this paper, we show that quantum speedup is possible for the problem of finding an optimal variable ordering of the OBDD for a given function. This is the first quantum speedup for the OBDD-related problems. Our algorithms assume the quantum random access memory (QRAM) model [6], which is commonly used in the literature concerned with quantum algorithms. In the model, one can read contents from or write them into quantum memory in a superposition. We provide our main result in the following theorem.

► **Theorem 1.** *There exists a quantum algorithm that, for a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ given as its truth table, produces a minimum OBDD representing f together with the corresponding variable ordering in $O^*(\gamma^n)$ time and space with an exponentially small error probability with respect to n , where the constant γ is at most 2.77286. Moreover, the OBDD produced by our algorithm is always a valid one for f , although it is not minimum with an exponentially small probability.*

This improves upon the classical best bound $O^*(3^n)$ [5] on time/space complexity. The classical algorithm achieving this bound is a deterministic one. However, there are no randomized algorithms that compute an optimal variable ordering in asymptotically less time complexity as far as we know.

It may seem somewhat restricted to assume that the function f is given as its truth table, since there are other common representations of Boolean functions such as DNFs, CNFs, Boolean circuits and OBDDs. However, this is not the case. Our algorithm actually works in more general settings where the input function f is given as any representation such that the value of f on any specified assignment can be computed over the representation in polynomial time in n , such as polynomial-size DNFs/CNFs/circuits and OBDDs of any size. This is because, in such cases, the truth table of f can be prepared in $O^*(2^n)$ time and the minimum OBDD is computable from that truth table with our algorithm. We restate Theorem 1 in a more general form as follows.

► **Corollary 2.** *Let $R(f)$ be any representation of a Boolean function f with n variables such that the value of $f(x)$ on any given assignment $x \in \{0,1\}^n$ can be computed on $R(f)$ in polynomial time with respect to n . Then, there exists a quantum algorithm that, for a function $f: \{0,1\}^n \rightarrow \{0,1\}$ given as $R(f)$, produces a minimum OBDD representing f together with the corresponding variable ordering in $O^*(\gamma^n)$ time and space with an exponentially small error probability with respect to n , where the constant γ is at most 2.77286. Possible representations as $R(f)$ are polynomial-size DNFs/CNFs/circuits and OBDDs of any size for function f .*

There are many variants of OBDDs, among which the zero-suppressed BDDs (ZDDs or ZBDDs) introduced by Minato [9] have been shown to be very powerful in dealing with combinatorial problems (see Knuth's famous book [7] for how to apply ZDDs to such problems). With slight modifications, our algorithm can construct a minimum ZDD with the same time/space complexity. We believe that similar speedups are possible for many other variants of OBDDs (adapting our algorithm to multiterminal BDDs (MTBDDs) [8] is almost trivial).

Technical Contribution

Recently, Ambainis et al. [1] has introduced break-through quantum techniques to speed up classical dynamic programming approaches. Inspired by their technique, our quantum algorithm speeds up the classical one (called FS) discovered by Friedman and Supowit [5]. Ambainis et al.'s results depend on the property that a large problem can be divided into subproblems that can be regarded as a scale-down version of the original problem and can be solved with the same algorithm, as is often the case with graph problems. In our case, firstly, it is unclear whether the problem can be divided into subproblems. Secondly, subproblems would be to optimize the ordering of variables starting from the middle variable or even from the opposite end, i.e., from the variable to be read *first*, toward the one to be read *last*. Such subproblems cannot be solved with the algorithm FS, and, in particular, optimizing in the latter case essentially requires the equivalence check of subfunctions of f , which is very costly. Our technical contribution is to find, by carefully observing the unique properties of OBDDs, that it is actually possible to even recursively divide the original problem into not the same but somewhat similar kinds of subproblems, to generalize the algorithm FS so that it can solve the subproblems, and to use the quantum minimum finding algorithm to efficiently select the subproblems that essentially contribute to the optimal variable ordering. In the full paper [11], we provide the technical outline of our algorithm, which would help readers understand the structure of our algorithm.

2 Preliminaries

2.1 Basic Terminology

Let \mathbb{N} , \mathbb{Z} and \mathbb{R} be the sets of natural numbers, integers, and real numbers, respectively. For each $n \in \mathbb{N}$, let $[n]$ be the set $\{1, \dots, n\}$, and \mathcal{S}_n be the permutation group over $[n]$. We may denote a singleton set $\{k\}$ by k for notational simplicity if it is clear from the context; for instance, $I \setminus \{k\}$ may be denoted by $I \setminus k$, if we know I is a set. For any subset $I \subseteq [n]$, let $\Pi_n(I)$ be the set of $\pi \in \mathcal{S}_n$ such that the first $|I|$ members $\{\pi[1], \dots, \pi[|I|]\}$ constitutes I , i.e.,

$$\Pi_n(I) := \{\pi \in \mathcal{S}_n : \{\pi[1], \dots, \pi[|I|]\} = I\} \subseteq \mathcal{S}_n.$$

For simplicity, we omit the subscript n and write $\Pi(I)$. More generally, for any two disjoint subsets $I, J \subseteq [n]$, let

$$\Pi_n(\langle I, J \rangle) := \{\pi \in \mathcal{S}_n : \{\pi[1], \dots, \pi[|I|]\} = I, \{\pi[|I| + 1], \dots, \pi[|I| + |J|]\} = J\} \subseteq \mathcal{S}_n.$$

For any disjoint subsets $I_1, \dots, I_m \subseteq [n]$ for $m \in [n]$, $\Pi_n(\langle I_1, \dots, I_m \rangle)$ is defined similarly. For simplicity, we may denote $\langle I \rangle$ by I , if it is clear from the context.

We denote the union operation over *disjoint* sets by \sqcup (instead of \cup) when we emphasize the disjointness of the sets.

For n Boolean variables x_1, \dots, x_n , any set $I \subseteq [n]$, and any vector $b = (b_1, \dots, b_{|I|}) \in \{0, 1\}^{|I|}$, x_I denotes the ordered set $(x_{j_1}, \dots, x_{j_{|I|}})$, where $\{j_1, \dots, j_{|I|}\} = I$ and $j_1 < \dots < j_{|I|}$, and $x_I = b$ denotes $x_{j_i} = b_i$ for each $i = [|I|]$. For any Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ with variables x_1, \dots, x_n , we denote by $f|_{x_I=b}$ the function obtained by restricting f with $x_I = b$. If I is a singleton set, say, $I = \{i\}$, we may write x_i and $f|_{x_i=b}$ to mean $x_{\{i\}}$ and $f|_{x_{\{i\}}=b}$, respectively, for notational simplicity. We say that g is a *subfunction* of f if g is equivalent to the function $f|_{x_I=b}$ for some $I \subseteq [n]$ and $b \in \{0, 1\}^{|I|}$.

For any function $g(n)$ in n , we use the notation $O^*(g(n))$ to hide a polynomial factor in n . We further denote $X = O^*(Y)$ by $X \approx Y$.

We use the following upper bound many times in this paper. For $n \in \mathbb{N}$ and $k \in [n] \cup \{0\}$, it holds that $\binom{n}{k} \leq 2^{n \mathbf{H}(k/n)}$, where $\mathbf{H}(\cdot)$ represents the binary entropy function $\mathbf{H}(\delta) := -\delta \log_2 \delta - (1 - \delta) \log_2 (1 - \delta)$.

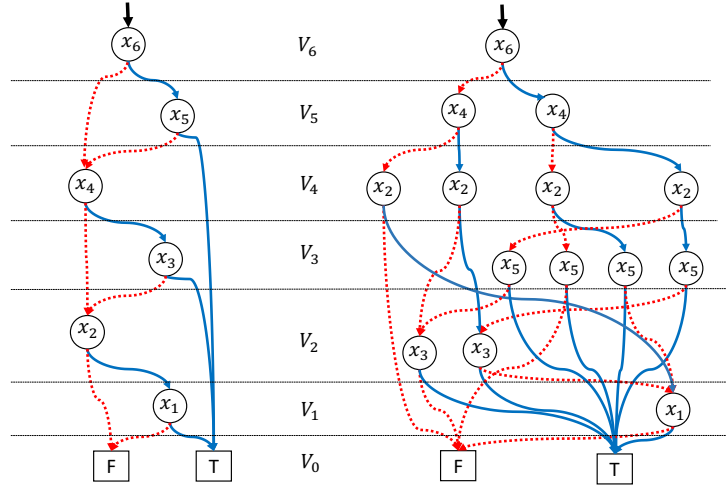
2.2 Ordered Binary Decision Diagrams

We provide a quick review of OBDDs. For more details, consult standard textbooks (e.g., Refs. [8, 12]).

For any Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ over variables x_1, \dots, x_n and any permutation $\pi \in \mathcal{S}_n$ (called a *variable ordering*), an OBDD $\mathcal{B}(f, \pi)$ is a single-rooted directed acyclic graph $G(V, E)$ that is unique up to isomorphism, defined as follows (examples are shown in Figure 1).

1. The node set V is the union of two disjoint sets N and T of *non-terminal* nodes with out-degree two and *terminal* nodes with out-degree zero, respectively, where T contains exactly two nodes: $T = \{\mathbf{f}, \mathbf{t}\}$. The set N contains a unique source node r , called the *root*.
2. $\mathcal{B}(f, \pi)$ is a leveled graph with $n + 1$ levels. Namely, the node set can be partitioned into n subsets: $V := V_0 \sqcup V_1 \sqcup \dots \sqcup V_n$, where $V_n = \{r\}$ and $V_0 = T = \{\mathbf{t}, \mathbf{f}\}$, such that each directed edge $(u, v) \in E$ is in $V_i \times V_j$ for a pair $(i, j) \in [n] \times (\{0\} \sqcup [n - 1])$ with $i > j$. For each $i \in [n]$, subset V_i (called the *level i*) is associated with the variable $x_{\pi[i]}$, or alternatively, each node in V_i is labeled with $x_{\pi[i]}$.¹ For convenience, we define a map $\text{var}: N \rightarrow [n]$ such that if $v \in V_i$ then $\text{var} = \pi[i]$.
3. The two edges emanating from every non-terminal node v are called the *0-edge* and the *1-edge*, which are labeled with 0 and 1, respectively. For every $u \in N$, let u_0 and u_1 be the destinations of the 0-edge and 1-edge of u , respectively.
4. Let $\mathcal{F}(f)$ be the set of all subfunctions of f . Define a bijective map $F: V \rightarrow \mathcal{F}(f)$ as follows: (a) $F(r) = f$ for $r \in V_n$; (b) $F(\mathbf{t}) = \text{true}$ and $F(\mathbf{f}) = \text{false}$ for $\mathbf{t}, \mathbf{f} \in V_0$; (c) For every $u \in N$ and $b \in \{0, 1\}$, $F(u_b)$ is the subfunction obtained from $F(u)$ by substituting $x_{\text{var}(u)}$ with b , i.e., $F(u_b) = F(u)|_{x_{\text{var}(u)}=b}$.

¹ In the standard definition, V_i is associated with the variable $x_{\pi[n-i]}$. Our definition follows the one given in [5] to avoid complicated subscripts of variables.



■ **Figure 1** The OBDDs represent the function $f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ under two variable orderings: $(x_1, x_2, x_3, x_4, x_5, x_6)$ (left) and $(x_1, x_3, x_5, x_2, x_4, x_6)$ (right), where the solid and dotted arcs express 1-edges and 0-edges, respectively, and the terminal nodes for true and false are labeled with T and F, respectively. For each $n \in \mathbb{N}$, the function $f(x_1, \dots, x_{2n}) = x_1x_2 + x_3x_4 + \dots + x_{2n-1}x_{2n}$ has a $(2n + 2)$ -sized OBDD for the ordering (x_1, \dots, x_{2n}) and a 2^{n+1} -sized OBDD for the ordering $(x_1, x_3, \dots, x_{2n-1}, x_2, x_4, \dots, x_{2n})$.

5. $\mathcal{B}(f, \pi)$ must be minimal in the sense that the following reduction rules cannot be applied. In other words, $\mathcal{B}(f, \pi)$ is obtained by maximally applying the following rules:
- if there exists a *redundant* node $u \in N$, then remove u and its outgoing edges, and redirect all the incoming edges of u to u_0 , where a node u is redundant if u_0 is the same node as u_1 .
 - if there exist *equivalent nodes* $\{u, v\} \subset N$, then remove v (i.e., any one of them) and its outgoing edges, and redirect all incoming edges of v to u , where u and v are equivalent if (1) $\text{var}(u)$ is equal to $\text{var}(v)$, and (2) u_0 and u_1 are the same nodes as v_0 and v_1 , respectively.

For each $j \in [n]$, $\text{Cost}_j(f, \pi)$ denotes the width at the level associated with the variable x_j , namely, the number of nodes in the level $\pi^{-1}[j]$ (see Figure 2 in Appendix). For $I \subseteq [n]$, let π_I be a permutation π in $\Pi(I)$ that minimizes the number of nodes in level 1 to level $|I|$:

$$\pi_I := \arg \min \left\{ \sum_{j=1}^{|I|} \text{Cost}_{\pi[j]}(f, \pi) : \pi \in \Pi(I) \right\}. \quad (1)$$

Note that $\sum_{j=1}^{|I|} \text{Cost}_{\pi[j]}(f, \pi) = \sum_{i \in I} \text{Cost}_i(f, \pi)$ for $\pi \in \Pi(I)$. More generally, for disjoint subsets $I_1, \dots, I_m \subseteq [n]$, $\pi_{\langle I_1, \dots, I_m \rangle}$ is a permutation in $\Pi(\langle I_1, \dots, I_m \rangle)$ that minimizes the number of the nodes in level 1 to level $|I_1| + \dots + |I_m|$ over all $\pi \in \Pi(\langle I_1, \dots, I_m \rangle)$:

$$\pi_{\langle I_1, \dots, I_m \rangle} := \arg \min \left\{ \sum_{j=1}^{|I_1| + \dots + |I_m|} \text{Cost}_{\pi[j]}(f, \pi) : \pi \in \Pi(\langle I_1, \dots, I_m \rangle) \right\}. \quad (2)$$

Note that $\min \sum_{j=1}^{|I_1| + \dots + |I_m|} \text{Cost}_{\pi[j]}(f, \pi) = \sum_{i \in I_1 \sqcup \dots \sqcup I_m} \text{Cost}_i(f, \pi)$ for any $\pi \in \Pi(\langle I_1, \dots, I_m \rangle)$. The following well-known lemma captures the essential property of OBDDs. It states that the number of nodes at level $i \in [n]$ is constant over all π , provided that the two sets $\{\pi[1], \dots, \pi[i-1]\}$ and $\{\pi[i+1], \dots, \pi[n]\}$ are fixed (see Figure 3 in Appendix).

► **Lemma 3** ([5]). *For any non-empty subset $I \subseteq [n]$ and any $i \in I$, there exists a constant c_f such that, for each $\pi \in \Pi(\langle I \setminus \{i\}, \{i\} \rangle)$, $\text{Cost}_{\pi[|I|]}(f, \pi) \equiv \text{Cost}_i(f, \pi) = c_f$.*

For convenience, we define shorthand for the minimums of the sums in Eqs. (1) and (2). For $I' \subseteq I \subseteq [n]$, $\text{MINCOST}_I[I']$ is defined as the number of nodes in the levels associated with variables indexed by elements in I' under permutation π_I , namely, $\text{MINCOST}_I[I'] := \sum_{i \in I'} \text{Cost}_i(f, \pi_I)$. More generally, for disjoint subsets $I_1, \dots, I_m \subseteq [n]$ and $I' \subseteq I_1 \sqcup \dots \sqcup I_m$,

$$\text{MINCOST}_{\langle I_1, \dots, I_m \rangle}[I'] := \sum_{i \in I'} \text{Cost}_i(f, \pi_{\langle I_1, \dots, I_m \rangle}).$$

As a special case, we denote $\text{MINCOST}_{\langle I_1, \dots, I_m \rangle}[I_1 \sqcup \dots \sqcup I_m]$ by $\text{MINCOST}_{\langle I_1, \dots, I_m \rangle}$. We define MINCOST_\emptyset as 0.

2.3 The Algorithm by Friedman and Supowit

This subsection reviews the algorithm by Friedman and Supowit [5]. We will generalize their idea later and heavily use the generalized form in our quantum algorithm. Hereafter, we call their algorithm FS.

2.3.1 Key Lemma and Data Structures

The following lemma is the basis of the dynamic programming approach used in FS.

► **Lemma 4.** *For any non-empty subset $I \subseteq [n]$ and any Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$, the following holds:*

$$\text{MINCOST}_I = \min_{k \in I} (\text{MINCOST}_{I \setminus k} + \text{Cost}_k(f, \pi_{\langle I \setminus k, k \rangle})) = \min_{k \in I} (\text{MINCOST}_{\langle I \setminus k, k \rangle}).$$

The proof is given in the full paper [11].

Before sketching algorithm FS, we provide several definitions. For any $I \subseteq [n]$, TABLE_I is an array with $2^{n-|I|}$ cells each of which stores a non-negative integer. Intuitively, for $b \in \{0, 1\}^{n-|I|}$, the cell $\text{TABLE}_I[b]$ stores (the pointer to) the unique node of $\mathcal{B}(f, \pi_I)$ associated via F with function $f|_{x_{[n] \setminus I} = b}$. Hence, we may write $\text{TABLE}_I[x_{[n] \setminus I} = b]$ instead of $\text{TABLE}_I[b]$ to clearly indicate the value assigned to each variable x_j for $j \in [n] \setminus I$. The purpose of TABLE_I is to relate all subfunctions $f|_{x_{[n] \setminus I} = b}$ ($b \in \{0, 1\}^{n-|I|}$) to the corresponding nodes of $\mathcal{B}(f, \pi_I)$. We assume without loss of generality that the pointers to nodes of $\mathcal{B}(f, \pi_I)$ are non-negative integers and, in particular, those to the two terminal nodes corresponding to false and true are the integers 0 and 1, respectively. Thus, TABLE_\emptyset is merely the truth table of f .

Algorithm FS computes TABLE_I together with π_I , MINCOST_I , and another data structure, NODE_I for all $I \subseteq [n]$, starting from TABLE_\emptyset via dynamic programming. NODE_I is the set of all triples of (the pointers to) nodes, $(u, u_0, u_1) \in N \times (N \sqcup T) \times (N \sqcup T)$, in $\mathcal{B}(f, \pi_I)$, where $\text{var}(u) = \pi_I[|I|]$, and (u, u_0) and (u, u_1) are the 0-edge and 1-edge of u , respectively. Thus, NODE_I contains the structure of the subgraph of $\mathcal{B}(f, \pi_I)$ induced by $V_{|I|}$. The purpose of the NODE_I is to prevent the algorithm from duplicating existing nodes, i.e., creating nodes associated with the same subfunctions as those with which the existing nodes are associated. By the definition, NODE_\emptyset is the empty set. We assume that NODE_I is implemented with an appropriate data structure, such as a balanced tree, so that the time complexity required for membership testing and insertion is the order of logarithm in the number of triples stored in NODE_I . An example of TABLE_I and NODE_I is shown in Figure 4 in Appendix.

More generally, for disjoint subset $I_1, \dots, I_m \subseteq [n]$, $\text{TABLE}_{\langle I_1, \dots, I_m \rangle}$ is an array with $2^{n-|I_1 \sqcup \dots \sqcup I_m|}$ cells such that, for $b \in \{0, 1\}^{n-|I_1 \sqcup \dots \sqcup I_m|}$, $\text{TABLE}_{\langle I_1, \dots, I_m \rangle}[b]$ stores the nodes of $\mathcal{B}(f, \pi_{\langle I_1, \dots, I_m \rangle})$ associated with the function $f|_{x_{[n] \setminus I_1 \sqcup \dots \sqcup I_m} = b}$. $\text{NODE}_{\langle I_1, \dots, I_m \rangle}$ is defined similarly for $\mathcal{B}(f, \pi_{\langle I_1, \dots, I_m \rangle})$. For simplicity, we hereafter denote by $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ the quadruplet $(\pi_{\langle I_1, \dots, I_m \rangle}, \text{MINCOST}_{\langle I_1, \dots, I_m \rangle}, \text{TABLE}_{\langle I_1, \dots, I_m \rangle}, \text{NODE}_{\langle I_1, \dots, I_m \rangle})$.

2.3.2 Sketch of Algorithm FS

Algorithm FS performs the following operations for $k = 1, \dots, n$ in this order. For each k -element subset $I \subseteq [n]$, compute $\mathcal{FS}(\langle I \setminus i, i \rangle)$ from $\mathcal{FS}(\langle I \setminus i \rangle)$ for each $i \in I$ in the manner described later (note that, since the cardinality of the set $I \setminus i$ is $k - 1$, $\mathcal{FS}(\langle I \setminus i \rangle)$ has already been computed). Then set $\mathcal{FS}(I) \leftarrow \mathcal{FS}(\langle I \setminus i^*, i^* \rangle)$, where i^* is the index $i \in I$ that minimizes $\text{MINCOST}_{\langle I \setminus i, i \rangle}$, implying that $\pi_I = \pi_{\langle I \setminus i^*, i^* \rangle}$. This is justified by Lemma 4. A schematic view of the algorithm is shown in Figure 5 in Appendix.

To compute $\mathcal{FS}(\langle I \setminus i, i \rangle)$ from $\mathcal{FS}(\langle I \setminus i \rangle)$, do the following. First set $\text{NODE}_{\langle I \setminus i, i \rangle} \leftarrow \emptyset$ and $\text{MINCOST}_{\langle I \setminus i, i \rangle} \leftarrow \text{MINCOST}_{I \setminus i}$ as their initial values. Then, for each $b \in \{0, 1\}^{n-|I|}$, set

$$u_0 \leftarrow \text{TABLE}_{I \setminus i}[x_{[n] \setminus I} = b, x_i = 0], \quad u_1 \leftarrow \text{TABLE}_{I \setminus i}[x_{[n] \setminus I} = b, x_i = 1].$$

If $u_0 = u_1$, then store u_0 in $\text{TABLE}_{\langle I \setminus i, i \rangle}[b]$. Otherwise, test whether (u, u_0, u_1) for some u is a member of $\text{NODE}_{I \setminus i}$. If it is, store u in the $\text{TABLE}_{\langle I \setminus i, i \rangle}[b]$; otherwise create a new triple (u', u_0, u_1) , insert it to $\text{NODE}_{\langle I \setminus i, i \rangle}$ and increment $\text{MINCOST}_{\langle I \setminus i, i \rangle}$. Since u' is the pointer to the new node, u' must be different from any pointer already included in $\text{NODE}_{\langle I \setminus i, i \rangle}$ and from any pointer to a node in $V_1 \sqcup \dots \sqcup V_{k-1}$ in $\mathcal{B}(f, \pi_{\langle I \setminus i \rangle})$, where $k = |I|$. Such u' can be easily chosen by setting u' to two plus the value of $\text{MINCOST}_{\langle I \setminus i, i \rangle}$ before the increment, since the $\text{MINCOST}_{\langle I \setminus i, i \rangle}$ is exactly the number of triples in $\text{NODE}_{\langle I \setminus i, i \rangle}$ plus $|V_1 \sqcup \dots \sqcup V_{k-1}|$, and the numbers 0 and 1 are reserved for the terminal nodes. We call the above procedure *table folding* with respect to x_i , because it halves the size of $\text{TABLE}_{\langle I \setminus i \rangle}$. We also mean it by “folding $\text{TABLE}_{\langle I \setminus i \rangle}$ with respect to x_i ”.

The complexity analysis is fairly simple. For each k , we need to compute $\mathcal{FS}(I)$ for $\binom{n}{k}$ possible I 's with $|I| = k$. For each I , it takes $O^*(2^{n-k})$ time since the size of $\text{TABLE}_{I \setminus i}$ is 2^{n-k+1} and each operation to $\text{NODE}_{I \setminus i}$ takes a polynomial time in n . Thus, the total time is $\sum_{k=0}^n 2^{n-k+1} \binom{n}{k} = 2 \cdot 3^n$ up to a polynomial factor. The point is that computing each $\mathcal{FS}(I)$ takes time linear to the size of $\text{TABLE}_{I \setminus i}$ up to a polynomial factor. The space required by Algorithm FS during the process for k is dominated by that for TABLE_I , $\text{TABLE}_{I \setminus i}$ and NODE_I for all I and $i \in I$, which is $O^*(2^{n-k} \binom{n}{k})$. The space complexity is thus $O^*(\max_{k \in \{0\} \cup [n]} 2^{n-k} \binom{n}{k}) = O^*(3^n)$.

► **Theorem 5** (Friedman and Supowit [5]). *Suppose that the truth table of $f: \{0, 1\}^n \rightarrow \{0, 1\}$ is given as input. Algorithm FS produces $\mathcal{FS}([n])$ in $O^*(3^n)$ time and space.*

2.4 Quantum Computation

We assume that readers have a basic knowledge of quantum computing (e.g., Ref. [10]). We provide only a lemma used to obtain our results.

► **Lemma 6** (Quantum Minimum Finding [4, 3]). *For every $\varepsilon > 0$ there exists a quantum algorithm that, for a function $f: [N] \rightarrow \mathbb{Z}$ given as an oracle, finds an element $x \in [N]$ at which $f(x)$ achieves the minimum, with error probability at most ε by making $O(\sqrt{N \log(1/\varepsilon)})$ queries.*

In this paper, the search space N is exponentially large in n and we are interested in exponential complexities, ignoring polynomial factors in them. We can thus safely assume $\varepsilon = 1/2^{p(n)}$ for a polynomial $p(n)$, so that the overhead is polynomially bounded. Since our algorithms use Lemma 6 a constant number of times, their overall error probabilities are exponentially small for a sufficiently large $p(n)$. In the following proofs, we thus assume that ε is exponentially small whenever we use Lemma 6, and do not explicitly analyze the error probability for simplicity.

Our algorithms assume the quantum random access memory (QRAM) model [6], which is commonly used in the literature when considering quantum algorithms. In the model, one can read contents from or write them into quantum memory in a superposition.

3 Quantum Algorithm with Divide-and-Conquer

We generalize Lemma 4 and Theorem 5 and use them in our quantum algorithm.

► **Lemma 7.** *For any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$ and any Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$, the following holds:*

$$\begin{aligned} \text{MINCOST}_{\langle I_1, \dots, I_m, J \rangle} &= \min_{k \in J} \left(\text{MINCOST}_{\langle I_1, \dots, I_m, J \setminus \{k\} \rangle} + \text{Cost}_k(f, \pi_{\langle I_1, \dots, I_m, J \setminus \{k\}, \{k\} \rangle}) \right) \\ &= \min_{k \in J} \left(\text{MINCOST}_{\langle I_1, \dots, I_m, J \setminus \{k\}, \{k\} \rangle} \right). \end{aligned}$$

The proof of this lemma is very similar to that of Lemma 4 and given in the full paper [11]. Based on Lemma 7, we generalize Theorem 5 to obtain algorithm FS^* (its pseudo code is given below, and a schematic view of FS^* is shown in Figure 6 in Appendix).

► **Lemma 8 (Classical Composition Lemma).** *For disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, there exists a deterministic algorithm FS^* that produces $\mathcal{FS}(\langle I_1, \dots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ for an underlying function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ in $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot 3^{|J|})$ time and space. More generally, for each $k \in [|J|]$, the algorithm produces the set $\{\mathcal{FS}(\langle I_1, \dots, I_m, K \rangle) : K \subseteq J, |K| = k\}$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ in $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \sum_{j=0}^k 2^{|J|-j} \binom{|J|}{j})$ time and space.*

Note that if $I_1 \sqcup \dots \sqcup I_m = \emptyset$ and $J = [n]$, then we obtain Theorem 5.

Proof. We focus on the simplest case of $m = 1$, for which our goal is to show an algorithm that produces $\mathcal{FS}(\langle I, J \rangle)$ from $\mathcal{FS}(I)$. It is straightforward to generalize the proof to the case of $m \geq 2$. Starting from $\mathcal{FS}(I)$, the algorithm first folds TABLE_I with respect to each variable in $\{x_j : j \in J\}$ to obtain $\mathcal{FS}(\langle I, j \rangle)$ for every $j \in J$, then fold $\text{TABLE}_{\langle I, j_1 \rangle}$ with respect to x_{j_2} and $\text{TABLE}_{\langle I, j_2 \rangle}$ with respect to x_{j_1} to obtain $\mathcal{FS}(\langle I, \{j_1, j_2\} \rangle)$ by taking the minimum of $\text{MINCOST}_{\langle I, j_1, j_2 \rangle}$ and $\text{MINCOST}_{\langle I, j_2, j_1 \rangle}$ for every $j_1, j_2 \in J$, and repeat this to finally obtain $\mathcal{FS}(\langle I, J \rangle)$. This algorithm is justified by Lemma 7. For each $j \in [|J|]$, $K \subseteq J$ with $|K| = j$, and $h \in K$, the time complexity of computing $\mathcal{FS}(\langle I, K \rangle)$ from $\mathcal{FS}(\langle I, K - h \rangle)$ is linear to the size of $\text{TABLE}_{\langle I, K \rangle}$, i.e., $2^{n-|I|-j}$ up to a polynomial factor. The total time is thus, up to a polynomial factor,

$$\sum_{j=1}^{|J|} 2^{n-|I|-j} \binom{|J|}{j} < 2^{n-|I|-|J|} \sum_{j=0}^{|J|} 2^{|J|-j} \binom{|J|}{j} = 2^{n-|I \sqcup J|} \cdot 3^{|J|}.$$

If we stop the algorithm at $j = k$, then the algorithm produces the set $\{\mathcal{FS}(\langle I, K \rangle) : K \subseteq J, |K| = k\}$. The time complexity in this case is at most $2^{n-|I|-|J|} \sum_{j=0}^k 2^{|J|-j} \binom{|J|}{j}$, up to a polynomial factor.

Since the space complexity is trivially upper-bounded by the time complexity, we complete the proof. \blacktriangleleft

► **Remark 9.** It is not difficult to see that the algorithm FS^* works even when the function f has a multivalued function: $f: \{0, 1\}^n \rightarrow \mathbb{Z}$. The only difference from the Boolean case is that the truth table maps each Boolean assignment to a value in \mathbb{Z} . In this case, the algorithm produces a variant of an OBDD (called a multi-terminal BDD, MTBDD) of minimum size. In addition, our algorithm with slight modifications to the table folding rule in FS^* can construct a minimum zero-suppressed BDD (ZDD) [9] for a given Boolean function. The details are described in the full paper [11]. These modifications are also possible for the quantum algorithms described later, since they perform table folding by running FS^* as a subroutine.

■ **Algorithm FS^*** Composable variant of algorithm FS. “ $A \leftarrow B$ ” means that B is substituted for A .

Input: disjoint subsets $I, J \in [n]$ and $\mathcal{FS}(I)$

Output: $\mathcal{FS}(\langle I, J \rangle)$

```

1 Function Main()
2   for  $\ell := 1$  to  $|J|$  do
3     for each  $\ell$ -element subset  $K \subseteq J$  do
4        $\text{MINCOST}_{\langle I, K \rangle} \leftarrow +\infty$ ; // init.
5       for each  $k \in K$  do
6          $\mathcal{FS}(\langle I, K \setminus k, k \rangle) \leftarrow \text{FOLD}(I, K, k, \mathcal{FS}(\langle I, K \setminus k \rangle))$ ;
7         if  $\text{MINCOST}_{\langle I, K \rangle} > \text{MINCOST}_{\langle I, K \setminus k, k \rangle}$  then
8            $\mathcal{FS}(\langle I, K \rangle) \leftarrow \mathcal{FS}(\langle I, K \setminus k, k \rangle)$ ;
9         end
10      end
11    end
12  end
13  return  $\mathcal{FS}(\langle I, J \rangle)$ 
14 end
15 Function FOLD( $I, K, k, \mathcal{FS}(\langle I, K \setminus k \rangle)$ ) // produce  $\mathcal{FS}(\langle I, K \setminus k, k \rangle)$  from  $\mathcal{FS}(\langle I, K \setminus k \rangle)$ 
16   $\pi_{\langle I, K \setminus k, k \rangle} \in \{\pi \in \Pi(\langle I, K \setminus k, k \rangle) : \pi[i] = \pi_{\langle I, K \setminus k \rangle}[i] \ (i = 1, \dots, |I \sqcup K| - 1)\}$ ; // init.
17   $\text{MINCOST}_{\langle I, K \setminus k, k \rangle} \leftarrow \text{MINCOST}_{\langle I, K \setminus k \rangle}$ ; // init.
18   $\text{NODE}_{\langle I, K \setminus k, k \rangle} \leftarrow \emptyset$ ; // init.
19  for  $b \in \{0, 1\}^{n - |I| - |K|}$  do
20     $u_0 \leftarrow \text{TABLE}_{\langle I, K \setminus k \rangle}[x_{[n] \setminus (I \sqcup K)} = b, x_k = 0]$ ;
21     $u_1 \leftarrow \text{TABLE}_{\langle I, K \setminus k \rangle}[x_{[n] \setminus (I \sqcup K)} = b, x_k = 1]$ ;
22    if  $u_0 = u_1$  then
23       $\text{TABLE}_{\langle I, K \setminus k, k \rangle}[x_{[n] \setminus (I \sqcup K)} = b] \leftarrow u_0$ 
24    else if  $\exists u \ (u, u_0, u_1) \in \text{NODE}_{\langle I, K \setminus k, k \rangle}$  then
25       $\text{TABLE}_{\langle I, K \setminus k, k \rangle}[x_{[n] \setminus (I \sqcup K)} = b] \leftarrow u$ 
26    else // create a new node
27       $u \leftarrow \text{MINCOST}_{\langle I, K \setminus k, k \rangle} + 2$ ;
28       $\text{TABLE}_{\langle I, K \setminus k, k \rangle}[x_{[n] \setminus (I \sqcup K)} = b] \leftarrow u$ ;
29       $\text{MINCOST}_{\langle I, K \setminus k, k \rangle} \leftarrow \text{MINCOST}_{\langle I, K \setminus k, k \rangle} + 1$ ;
30      insert  $(u, u_0, u_1)$  into  $\text{NODE}_{\langle I, K \setminus k, k \rangle}$ 
31    end
32  end
33  return  $\mathcal{FS}(\langle I, K \setminus k, k \rangle)$ 
34 end

```

36:10 Quantum Algorithm for Finding the Optimal Variable Ordering for BDDs

The following theorem is the basis of our quantum algorithms.

► **Lemma 10** (Divide-and-Conquer). *For any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$ and any $k \in [|J|]$, it holds that $\text{MINCOST}_{\langle I_1, \dots, I_m, J \rangle}[J]$ is equal to*

$$\min_{K: K \subseteq J, |K|=k} (\text{MINCOST}_{\langle I_1, \dots, I_m, K \rangle}[K] + \text{MINCOST}_{\langle I_1, \dots, I_m, K, J \setminus K \rangle}[J \setminus K]). \quad (3)$$

In particular, when $I_1 \sqcup \dots \sqcup I_m = \emptyset$ and $J = [n]$, it holds that

$$\text{MINCOST}_{[n]} = \min_{K \subseteq [n], |K|=k} (\text{MINCOST}_K + \text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K]). \quad (4)$$

Proof. We first prove the special case of $I_1 \sqcup \dots \sqcup I_m = \emptyset$ and $J = [n]$. By the definition, we have

$$\text{MINCOST}_{[n]} = \sum_{j=1}^n \text{Cost}_{\pi[j]}(f, \pi) = \sum_{j=1}^k \text{Cost}_{\pi[j]}(f, \pi) + \sum_{j=k+1}^n \text{Cost}_{\pi[j]}(f, \pi)$$

for the optimal permutation $\pi = \pi_{[n]}$. Let $K = \{\pi[1], \dots, \pi[k]\}$. By Lemma 3, the first sum is independent of how π maps $\{k+1, \dots, n\}$ to $[n] \setminus K$. Thus, it is equal to the minimum of $\sum_{j=1}^k \text{Cost}_{\pi_1[j]}(f, \pi)$ over all $\pi_1 \in \Pi(K)$, i.e., MINCOST_K . Similarly, the second sum is independent of how π maps $[k]$ to K . Thus, it is equal to the minimum of $\sum_{j=k+1}^n \text{Cost}_{\pi_2[j]}(f, \pi)$ over all $\pi_2 \in \Pi(\langle K, [n] \setminus K \rangle)$, i.e., $\text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K]$. This completes the proof of Eq. (4).

We can generalize this in a straightforward manner. Let $\pi = \pi_{\langle I_1, \dots, I_m, J \rangle}$ and $\ell = |I_1 \sqcup \dots \sqcup I_m|$. Then, we have

$$\text{MINCOST}_{\langle I_1, \dots, I_m, J \rangle}[J] = \sum_{j=1}^k \text{Cost}_{\pi[\ell+j]}(f, \pi) + \sum_{j=k+1}^{|J|} \text{Cost}_{\pi[\ell+j]}(f, \pi).$$

By defining $K := \{\pi[\ell+1], \dots, \pi[\ell+k]\}$, the same argument as the special case of $\ell = 0$ implies that the first and second sums are $\text{MINCOST}_{\langle I_1, \dots, I_m, K \rangle}[K]$ and $\text{MINCOST}_{\langle I_1, \dots, I_m, K, J \setminus K \rangle}[J \setminus K]$, respectively. This completes the proof of Eq. (3). ◀

A schematic view of the above lemma is shown in Figure 7 in Appendix.

3.1 Simple Cases

We provide simple quantum algorithms on the basis of Lemma 10. The lemma states that, for any $k \in [n]$, $\text{MINCOST}_{[n]}$ is the minimum of $\text{MINCOST}_K + \text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K]$ over all $K \subseteq [n]$ with $|K| = k$. To find K from among $\binom{n}{k}$ possibilities that minimizes this amount, we use the quantum minimum finding (Lemma 6). To compute $\text{MINCOST}_K + \text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K] = \text{MINCOST}_{\langle K, [n] \setminus K \rangle}$, it suffices to first compute $\mathcal{FS}(K)$ (including MINCOST_K), and then $\mathcal{FS}(\langle K, [n] \setminus K \rangle)$ (including $\text{MINCOST}_{\langle K, [n] \setminus K \rangle}$) from $\mathcal{FS}(K)$. The time complexity for computing $\mathcal{FS}(K)$ from $\mathcal{FS}(\emptyset)$ is $O^*(2^{n-k}3^k)$ by Lemma 8 with $I_1 \sqcup \dots \sqcup I_m = \emptyset$ and $J = K$, while that for computing $\mathcal{FS}(\langle K, [n] \setminus K \rangle)$ from $\mathcal{FS}(K)$ is $O^*(3^{n-k})$ by Lemma 8 with $m = 1$, $I_1 = K$, and $J = [n] \setminus K$. Thus, the time complexity for computing $\mathcal{FS}(\langle K, [n] \setminus K \rangle)$ from $\mathcal{FS}(\emptyset)$ is $O^*(2^{n-k}3^k + 3^{n-k})$. Thus, for $k = \alpha n$ with $\alpha \in [0, 1]$ fixed later, the total time complexity up to a polynomial factor is

$$T(n) = \sqrt{\binom{n}{\alpha n}} \left(2^{(1-\alpha)n} 3^{\alpha n} + 3^{(1-\alpha)n} \right) \leq 2^{\frac{1}{2}\mathbf{H}(\alpha)n} \left\{ 2^{[(1-\alpha)+\alpha \log_2 3]n} + 2^{[(1-\alpha) \log_2 3]n} \right\}.$$

To balance the both terms, we set $(1 - \alpha) + \alpha \log_2 3 = (1 - \alpha) \log_2 3$ and obtain $\alpha = \alpha^*$, where $\alpha^* = \frac{\log_2 3 - 1}{2 \log_2 3 - 1} \approx 0.269577$. We have

$$\min_{\alpha \in [0,1]} T(n) = O\left(2^{\frac{1}{2}\mathbf{H}(\alpha^*)n + (1-\alpha^*)n + \alpha^*(\log_2 3)n}\right) = O(\gamma_0^n),$$

where $\gamma_0 = 2.98581\dots$ ² This slightly improves the classical best bound $O^*(3^n)$ on the time complexity. To improve the bound further, we introduce a preprocess that classically computes $\mathcal{FS}(K)$ for every K with $|K| = \alpha n$ ($\alpha \in (0, 1)$) by using Algorithm FS*. By Lemma 8, the preprocessing time is then

$$\sum_{j=1}^{\alpha n} 2^{n-j} \cdot \binom{n}{j} \leq \alpha n \cdot \max_{j \in [\alpha n]} 2^{n-j} \binom{n}{j} \approx \begin{cases} 2^{(1-\alpha)n + \mathbf{H}(\alpha)n} & (\alpha < 1/3) \\ 2^{\frac{2}{3}n + \mathbf{H}(1/3)n} & (\alpha \geq 1/3), \end{cases} \quad (5)$$

since $2^{n-j} \binom{n}{j}$ increases when $j < n/3$ and decreases otherwise. Note that once this preprocess is completed, we can use $\mathcal{FS}(K)$ for free and assume that the cost for accessing $\mathcal{FS}(K)$ is polynomially bounded for all $K \subseteq [n]$ with $|K| = \alpha n$.

Then, assuming that $\alpha < 1/3$, the total time complexity up to a polynomial factor is

$$T(n) = \sum_{j=1}^{\alpha n} 2^{n-j} \cdot \binom{n}{j} + \sqrt{\binom{n}{\alpha n}} \left(n^{O(1)} + 3^{(1-\alpha)n}\right) \lesssim 2^{[(1-\alpha) + \mathbf{H}(\alpha)]n} + 2^{[\frac{1}{2}\mathbf{H}(\alpha) + (1-\alpha)\log_2 3]n}.$$

To balance the both terms, we set $(1 - \alpha) + \mathbf{H}(\alpha) = \frac{1}{2}\mathbf{H}(\alpha) + (1 - \alpha) \log_2 3$ and obtain the solution $\alpha = \alpha^*$, where $\alpha^* := 0.274863\dots$, which is less than $1/3$ as we assumed. At $\alpha = \alpha^*$, we have $T(n) \lesssim 2^{[(1-\alpha^*) + \mathbf{H}(\alpha^*)]n} = O^*(\gamma_1^n)$, where γ_1 is at most 2.97625 ($< \gamma_0$). Thus, introducing the preprocess improves the complexity bound. A schematic view of the above algorithm is shown in Figure 8 in Appendix.

3.2 General Case

We can improve this bound further by applying Lemma 10 k times. We denote the resulting algorithm with constant parameters $k \in \mathbb{N}$ and $\boldsymbol{\alpha} := (\alpha_1, \dots, \alpha_k)$ by $\text{OptOBDD}(k, \boldsymbol{\alpha})$ where $0 < \alpha_1 < \dots < \alpha_k < 1$. Its pseudo code is given below. In addition, we assume $\alpha_1 < 1/3$ and $\alpha_{k+1} = 1$ in the following complexity analysis.

To simplify notations, define two function as follows: for $x, y \in (0, 1)$ such that $x < y$, $f(x, y) := \frac{1}{2}y \cdot \mathbf{H}(x/y) + g(x, y)$ and $g(x, y) := (1 - y) + (y - x) \log_2(3)$.

By Lemma 8, the time required for the preprocess is $\sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell}$ up to a polynomial factor. Thus, the total time complexity can be described as the following recurrence:

$$T(n) = \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell} + L_{k+1}(n), \quad (6)$$

$$\begin{aligned} L_{j+1}(n) &= \sqrt{\binom{\alpha_{j+1}n}{\alpha_j n}} \left(L_j(n) + 2^{(1-\alpha_{j+1})n} 3^{(\alpha_{j+1}-\alpha_j)n}\right) \\ &= \sqrt{\binom{\alpha_{j+1}n}{\alpha_j n}} \left(L_j(n) + 2^{g(\alpha_j, \alpha_{j+1})n}\right), \end{aligned} \quad (7)$$

² More precisely, α^* must be rounded so that $\alpha^* n$ is an integer. We assume hereafter for simplicity of analysis that n is sufficiently large so that the rounding error is negligible compared to the approximation error in the optimum parameter values, such as α^* .

36:12 Quantum Algorithm for Finding the Optimal Variable Ordering for BDDs

■ **Algorithm OptOBDD**(k, α) Quantum OBDD-minimization algorithm with constant parameters $k \in \mathbb{N}$ and $\alpha := (\alpha_1, \dots, \alpha_k) \in [0, 1]^k$ satisfying $0 < \alpha_1 < \dots < \alpha_k < 1$, where the quantum minimum finding algorithm is used in line 8, and FS^* is used in lines 2 and 15. “ $A \leftarrow B$ ” means that B is substituted for A .

Input: $\mathcal{FS}(\emptyset) := \{ \text{TABLE}_\emptyset, \pi_\emptyset, \text{MINCOST}_\emptyset, \text{NODE}_\emptyset \}$ (accessible from all Functions)

Output: $\mathcal{FS}([n])$

```

1 Function Main()
2   compute the set  $\{\mathcal{FS}(K) : K \subseteq [n], |K| = \lfloor \alpha_1 n \rfloor\}$  by algorithm FS (or  $\text{FS}^*$ );
3   make the set of these  $\mathcal{FS}(K)$  global (i.e., accessible from all Functions);
4   return DivideAndConquer( $[n], k + 1$ )
5 end
6 Function DivideAndConquer( $L, t$ )                                // Compute  $\mathcal{FS}(L)$  with  $\alpha_1, \dots, \alpha_t (= \lfloor L/n \rfloor)$ 
7   if  $t = 1$  then return  $\mathcal{FS}(L)$ ;                                //  $\mathcal{FS}(L)$  has been precomputed.
8   Find  $K (\subset L)$  of cardinality  $\lfloor \alpha_{t-1} n \rfloor$ , with Lemma 6, that minimizes  $\text{MINCOST}_{\langle K, L \setminus K \rangle}$ ,
9   which is computed as a component of  $\mathcal{FS}(\langle K, L \setminus K \rangle)$  by calling ComputeFS( $K, L \setminus K, t$ );
10  let  $K^*$  be the set that achieves the minimum;
11  return  $\mathcal{FS}(\langle K^*, L \setminus K^* \rangle)$ 
12 end
13 Function ComputeFS( $K, M, t$ )                                // Compute  $\mathcal{FS}(\langle K, M \rangle)$  with  $\alpha_1, \dots, \alpha_t$ 
14   $\mathcal{FS}(K) \leftarrow \text{DivideAndConquer}(K, t - 1)$ ;
15   $\mathcal{FS}(\langle K, M \rangle) \leftarrow \text{FS}^*(K, M, \mathcal{FS}(K))$ ;
16  return  $\mathcal{FS}(\langle K, M \rangle)$ 
17 end

```

where $j \in [k]$ and $L_1(n) = O^*(1)$. Intuitively, $L_j(n)$ is the time required for producing $\mathcal{FS}(\langle K_1, K_2 \setminus K_1, \dots, K_j \setminus K_{j-1} \rangle)$ such that $\text{MINCOST}_{\langle K_1, K_2 \setminus K_1, \dots, K_j \setminus K_{j-1} \rangle}$ is minimum over all K_1, \dots, K_{j-1} satisfying $|K_\ell| = \alpha_\ell n$ for every $\ell \in [k + 1]$ and $K_\ell \subset K_{\ell+1}$ for every $\ell \in [k]$.

Since $L_1(n) = O^*(1)$, we have $L_2(n) \lesssim \sqrt{\frac{\alpha_2 n}{\alpha_1 n}} \cdot 2^{g(\alpha_1, \alpha_2)n} \lesssim 2^{f(\alpha_1, \alpha_2)n}$. By setting $f(\alpha_1, \alpha_2) = g(\alpha_2, \alpha_3)$, we have $L_3(n) = \sqrt{\frac{\alpha_3 n}{\alpha_2 n}} \cdot (L_2(n) + 2^{g(\alpha_2, \alpha_3)n}) \lesssim \sqrt{\frac{\alpha_3 n}{\alpha_2 n}} \cdot 2^{g(\alpha_2, \alpha_3)n} \lesssim 2^{f(\alpha_2, \alpha_3)n}$. In general, for $j = 2, \dots, k$, setting $f(\alpha_{j-1}, \alpha_j) = g(\alpha_j, \alpha_{j+1})$ yields

$$L_{j+1}(n) \lesssim 2^{f(\alpha_j, \alpha_{j+1})n}.$$

Therefore, the total complexity [Eq. (6)] is

$$T(n) \lesssim \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell} + 2^{f(\alpha_k, \alpha_{k+1})n} \lesssim 2^{(1-\alpha_1)n + \mathbf{H}(\alpha_1)n} + 2^{f(\alpha_k, 1)n},$$

where we use $\alpha_1 < 1/3$, $\alpha_{k+1} = 1$, and Eq. (5). To optimize the right-hand side, we set parameters so that $1 - \alpha_1 + \mathbf{H}(\alpha_1) = f(\alpha_k, 1)$.

In summary, we need to find the values of parameters $\alpha_1, \dots, \alpha_k$ that satisfy the following system of equations and $\alpha_1 < 1/3$:

$$1 - \alpha_1 + \mathbf{H}(\alpha_1) = f(\alpha_k, 1), \tag{8}$$

$$f(\alpha_{j-1}, \alpha_j) = g(\alpha_j, \alpha_{j+1}) \quad (j = 2, \dots, k). \tag{9}$$

By numerically solving this system of equations, we obtain $T(n) = O(\gamma_k^n)$, where γ_k is at most 2.83728 for $k = 6$. The value of γ_k becomes smaller as k increases. However, incrementing k beyond 6 provides only negligible improvement of γ_k . Since the space complexity is trivially upper-bounded by the time complexity, we have the following theorem. Note that the values

of α_i 's are not symmetric with respect to $1/2$. This reflects the fact that optimizing cost is not symmetric with respect to $1/2$, contrasting with many other combinatorial problems.

► **Theorem 11.** *There exists a quantum algorithm that, for the truth table of $f: \{0,1\}^n \rightarrow \{0,1\}$ given as input, produces $\mathcal{FS}([n])$ with probability $1 - \exp(-\Omega(n))$ in $O^*(\gamma^n)$ time and space, where the constant γ is at most 2.83728, which is achieved by $\text{OptOBDD}(k, \alpha)$ with $k = 6$ and $\alpha = (0.183791, 0.183802, 0.183974, 0.186131, 0.206480, 0.343573)$.*

4 Quantum Algorithm with Composition

4.1 Quantum Composition Lemma

By generalizing the quantum algorithm given in Theorem 11, we now provide a quantum version of Lemma 8, called the *quantum composition lemma*.

► **Lemma 12 (Quantum Composition: Base Part).** *For any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, there exists a quantum algorithm that, with probability $1 - \exp(-\Omega(n))$, produces $\mathcal{FS}(\langle I_1, \dots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ for an underlying function $f: \{0,1\}^n \rightarrow \{0,1\}$ in $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|})$ time and space, where γ is the constant defined in Theorem 11.*

The proof is given in the full paper [11]. The proof idea is similar to that used in the proof of Lemma 8. A pseudo code of the algorithm provided in Lemma 12 is shown below as $\text{OptOBDD}_\Gamma^*(k, \alpha)$, where the subroutine Γ appearing in line 17 is set to the deterministic algorithm FS^* , and k and α are set to the values specified in Theorem 11.

■ **Algorithm $\text{OptOBDD}_\Gamma^*(k, \alpha)$** Composable Quantum OBDD-minimization algorithm with subroutine Γ and constant parameters $k \in \mathbb{N}$ and $\alpha = (\alpha_1, \dots, \alpha_k) \in [0, 1]^k$ satisfying $0 < \alpha_1 < \dots < \alpha_k < 1$, where the quantum minimum finding algorithm is used in line 9, and subroutine Γ is used in line 17. “ $A \leftarrow B$ ” means that B is substituted for A . $\Gamma(I_1, I_2, J, \mathcal{FS}(I_1, I_2))$ produces $\mathcal{FS}(\langle I_1, I_2, J \rangle)$ from $\mathcal{FS}(\langle I_1, I_2 \rangle)$.

Input: $I \subseteq [n], J \subseteq [n], \mathcal{FS}(I)$. (accessible from all Functions)

Output: $\mathcal{FS}(\langle I, J \rangle)$

```

1 Function Main()
2    $n' \leftarrow |J|$ ; // init.
3   compute the set  $\{\mathcal{FS}(\langle I, K \rangle) : K \subseteq J, |K| = \lfloor \alpha_1 n' \rfloor\}$  by algorithm  $\text{FS}^*$ ;
4   make  $n'$  and the above set of  $\mathcal{FS}(\langle I, K \rangle)$  global (i.e., accessible from all Functions);
5   return DivideAndConquer( $J, k + 1$ )
6 end
7 Function DivideAndConquer( $L, t$ ) // Compute  $\mathcal{FS}(\langle I, L \rangle)$  with  $\alpha_1, \dots, \alpha_t$ 
8   if  $t = 1$  then return  $\mathcal{FS}(I, L)$ ; //  $\mathcal{FS}(I, L)$  has been precomputed.
9   Find  $K(\subset L)$  of cardinality  $\lfloor \alpha_{t-1} n' \rfloor$ , with Lemma 6, that minimizes  $\text{MINCOST}_{\langle I, K, L \setminus K \rangle}$ 
10  which is computed as a component of  $\mathcal{FS}(\langle I, K, L \setminus K \rangle)$ 
11  by calling  $\text{ComputeFS}(I, K, L \setminus K, t)$ ;
12  let  $K^*$  be the set that achieves the minimum;
13  return  $\mathcal{FS}(\langle I, K^*, L \setminus K^* \rangle)$ 
14 end
15 Function ComputeFS( $I, K, M, t$ ) // Compute  $\mathcal{FS}(\langle I, K, M \rangle)$  with  $\alpha_1, \dots, \alpha_t$ 
16   $\mathcal{FS}(I, K) \leftarrow \text{DivideAndConquer}(K, t - 1)$ ;
17   $\mathcal{FS}(\langle I, K, M \rangle) \leftarrow \Gamma(I, K, M, \mathcal{FS}(I, K))$ ;
18  return  $\mathcal{FS}(\langle I, K, M \rangle)$ 
19 end

```

► **Lemma 13** (Quantum Composition: Induction Part). *Suppose that Γ is a quantum algorithm that, for any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, produces $\mathcal{FS}(\langle I_1, \dots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ with probability $1 - \exp(-\Omega(n))$ in $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|})$ time and space for an underlying function $f: \{0, 1\}^n \rightarrow \{0, 1\}$. Then, for any constant parameters $k \in \mathbb{N}$ and $\alpha = (\alpha_1, \dots, \alpha_k) \in [0, 1]^k$ with $\alpha_1 < \dots < \alpha_k$ and for any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, $\text{OptOBDD}_\Gamma^*(k, \alpha)$ produces $\mathcal{FS}(\langle I_1, \dots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ with probability $1 - \exp(-\Omega(n))$ in $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot \beta_k^{|J|})$ time and space for the function f , where β_k^n upper-bounds, up to a polynomial factor, the time complexity required for $\text{OptOBDD}_\Gamma^*(k, \alpha)$ to compute $\mathcal{FS}([n])$ from $\mathcal{FS}(\emptyset)$, that is, $T(n) = O^*(\beta_k^n)$ for $T(n)$ that satisfies the following recurrence:*

$$T(n) = \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \binom{n}{\ell} + L_{k+1}, \quad (10)$$

$$L_{j+1} = \sqrt{\binom{\alpha_{j+1} n}{\alpha_j n}} \left(L_j + 2^{(1-\alpha_{j+1})n} \gamma^{(\alpha_{j+1}-\alpha_j)n} \right) = \sqrt{\binom{\alpha_{j+1} n}{\alpha_j n}} \left(L_j + 2^{g_\gamma(\alpha_j, \alpha_{j+1})} \right), \quad (11)$$

where $j \in [k]$, $L_1 = O^*(1)$ and $g_\gamma(x, y) := (1 - y) + (y - x) \log_2 \gamma$.

Proof. Recall that algorithm FS^* is used as a subroutine in $\text{OptOBDD}(k, \alpha)$ provided in Theorem 11. Since the input and output of Γ assumed in the statement are the same as those of algorithm FS^* , one can use Γ instead of algorithm FS^* in $\text{OptOBDD}(k, \alpha)$ (compromising on an exponentially small error probability). Let $\text{OptOBDD}_\Gamma(k, \alpha)$ be the resulting algorithm. Then, one can see that the time complexity $T(n)$ of $\text{OptOBDD}_\Gamma(k, \alpha)$ satisfies the recurrence: Eqs. (10)-(11), which are obtained by just replacing $g(x, y)$ with $g_\gamma(x, y)$ in Eqs. (6)-(7). Suppose that $T(n) = O^*(\beta_k^n)$ follows from the recurrence.

Next, we generalize $\text{OptOBDD}_\Gamma(k, \alpha)$ so that it produces $\mathcal{FS}(\langle I_1, \dots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \dots, I_m \rangle)$ for any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$. The proof is very similar to that of Lemma 12. The only difference is that the time complexity of Γ is $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|})$, instead of $O^*(2^{n-|I_1 \sqcup \dots \sqcup I_m \sqcup J|} \cdot 3^{|J|})$. Namely, when $m = 1$ and $n' = |J|$, the time complexity of $\text{OptOBDD}_\Gamma^*(k, \alpha)$ satisfies the following recurrence: for each $j \in [n]$,

$$T'(n, n') = 2^{n-|I|-n'} \sum_{\ell=1}^{\alpha_1 n'} 2^{n'-\ell} \binom{n'}{\ell} + L'_{k+1}(n, n'),$$

$$L'_{j+1}(n, n') = \sqrt{\binom{\alpha_{j+1} n'}{\alpha_j n'}} \left(L'_j(n, n') + 2^{n-|I|-\alpha_{j+1} n'} \gamma^{(\alpha_{j+1}-\alpha_j) n'} \right) \quad [j \in [n]],$$

$$L'_1(n, n') = O^*(1),$$

from which it follows that $T'(n, n') = 2^{n-|I|-n'} T(n') = O^*(2^{n-|I \sqcup J|} \cdot \beta_k^{|J|})$. It is straightforward to generalize to the case of $m \geq 2$.

The total error probability is exponentially small by union bound, if the error probabilities of Γ and the quantum minimum finding (Lemma 6) are made sufficiently small. ◀

4.2 The Final Algorithm

Lemmas 12 and 13 naturally lead to the following algorithm. We first define Γ_1 as $\text{OptOBDD}_{\text{FS}^*}^*(k^{(0)}, \alpha^{(0)})$ for some $k^{(0)} \in \mathbb{N}$ and $\alpha^{(0)} \in [0, 1]^{k^{(0)}}$. Then, we define Γ_2 as

$\text{OptOBDD}_{\Gamma_1}^*(k^{(1)}, \alpha^{(1)})$ for some $k^{(1)} \in \mathbb{N}$ and $\alpha^{(1)} \in [0, 1]^{k^{(1)}}$. In this way, we can define Γ_{i+1} as $\text{OptOBDD}_{\Gamma_i}^*(k^{(i)}, \alpha^{(i)})$ for some $k^{(i)} \in \mathbb{N}$ and $\alpha^{(i)} \in [0, 1]^{k^{(i)}}$.

Fix $k^{(i)} = 6$ for every i . Note that, in the proof of Lemmas 12 and 13, parameter $\alpha^{(i)} = (\alpha_1^{(i)}, \dots, \alpha_6^{(i)}) \in [0, 1]^6$ is set for each i so that it satisfies the system of equations, a natural generalization of Eqs. (8)-(9),

$$1 - \alpha_1^{(i)} + \mathbf{H}(\alpha_1^{(i)}) = f_\gamma(\alpha_6^{(i)}, 1), \quad (12)$$

$$f_\gamma(\alpha_{j-1}^{(i)}, \alpha_j^{(i)}) = g_\gamma(\alpha_j^{(i)}, \alpha_{j+1}^{(i)}) \quad (j = 2, \dots, 6), \quad (13)$$

where $f_\gamma(x, y) := \frac{1}{2}y \cdot \mathbf{H}(x/y) + g_\gamma(x, y)$ and $g_\gamma(x, y) := (1 - y) + (y - x) \log_2 \gamma$.

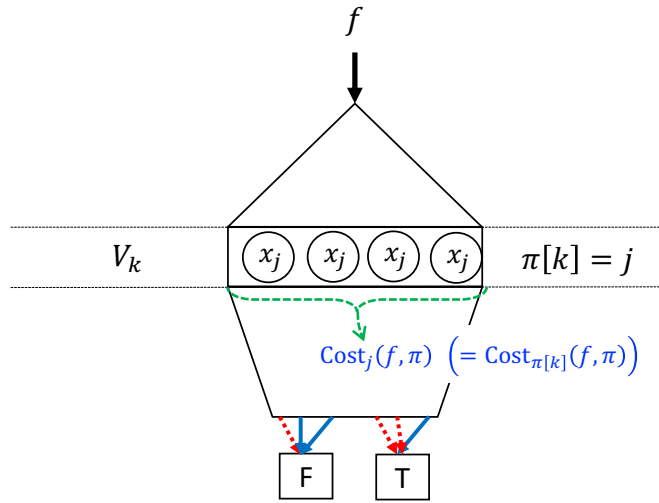
By numerically solving this system of equations for $\gamma = 3$, we have $\beta_6 < 2.83728$ as shown in Theorem 11. Then, numerically solving the system of equations with $\gamma = 2.83728$, we have $\beta_6 < 2.79364$. In this way, we obtain a certain γ less than 2.77286 at the tenth composition. We therefore obtain the following theorem.

► **Theorem 14.** *There exists a quantum algorithm that, for the truth table of $f: \{0, 1\}^n \rightarrow \{0, 1\}$ given as input, produces $\mathcal{FS}([n])$ in $O^*(\gamma^n)$ time and space with probability $1 - \exp(-\Omega(n))$, where the constant γ is at most 2.77286.*

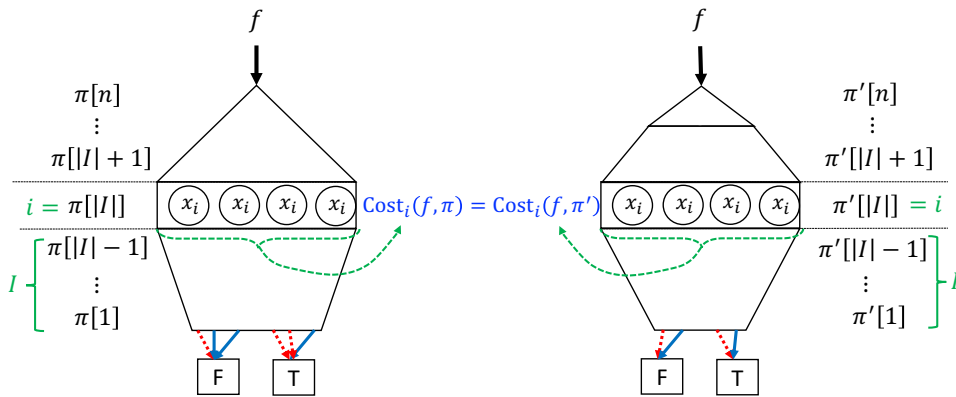
References

- 1 Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsīs, and Jevgēnijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1783–1793, 2019. doi:10.1137/1.9781611975482.107.
- 2 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
- 3 Harry Buhrman, Richard Cleve, Ronald de Wolf, and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science, FOCS*, pages 358–368, 1999.
- 4 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *arXiv.org e-Print archive*, quant-ph/9607014, 1996. URL: <http://arxiv.org/abs/quant-ph/9607014>.
- 5 Steven J. Friedman and Kenneth J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Trans. Comput.*, 39(5):710–713, 1990.
- 6 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, 2008.
- 7 Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 1 edition, 2009.
- 8 Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.
- 9 Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. 30th ACM/IEEE Design Automation Conference*, pages 272–277, 1993.
- 10 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- 11 Seiichiro Tani. Quantum algorithm for finding the optimal variable ordering for binary decision diagrams. *arXiv.org e-Print archive*, arXiv:1909.12658, 2019. URL: <https://arxiv.org/abs/1909.12658>.
- 12 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000.

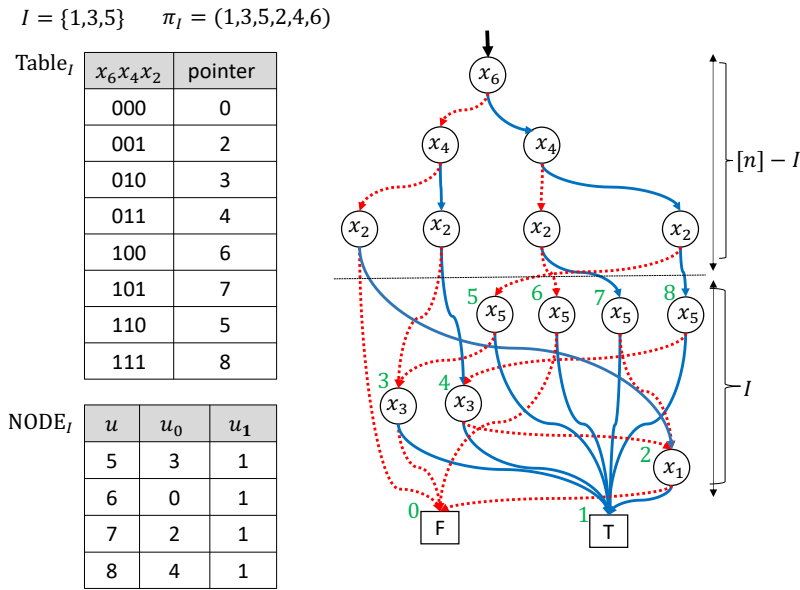
A Appendix



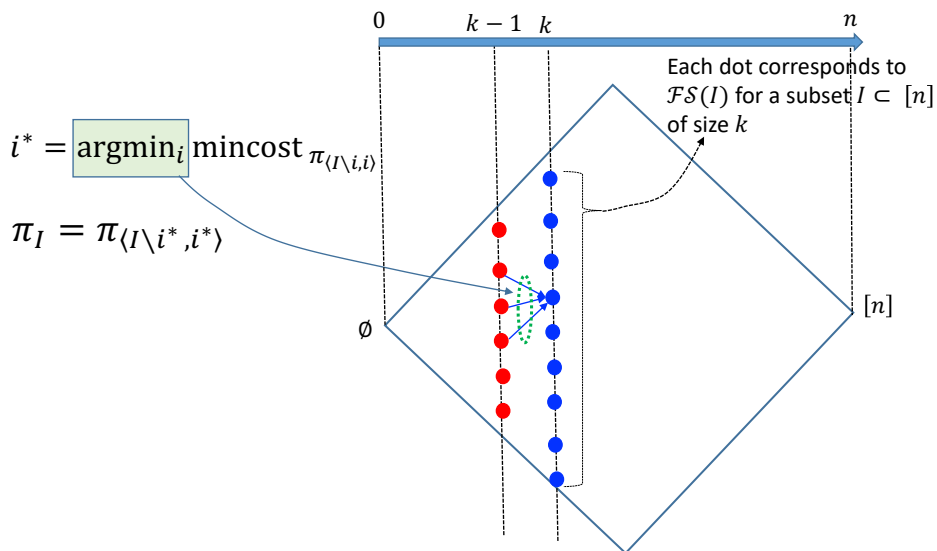
■ **Figure 2** Schematic expression of $\text{Cost}_j(f, \pi)$.



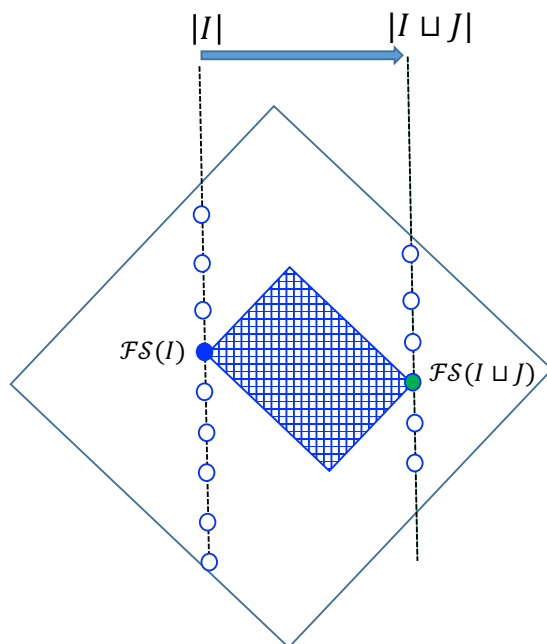
■ **Figure 3** Schematic expression of Lemma 3: For any two permutations $\pi, \pi' \in \mathcal{S}_n$ such that $\{\pi[1], \dots, \pi[|I| - 1]\} = \{\pi'[1], \dots, \pi'[|I| - 1]\}$ and $\pi[|I|] = \pi'[|I|]$, it holds that the number of nodes labeled with x_i in $\mathcal{B}(f, \pi)$ is equal to that of nodes labeled with x_i in $\mathcal{B}(f, \pi')$.



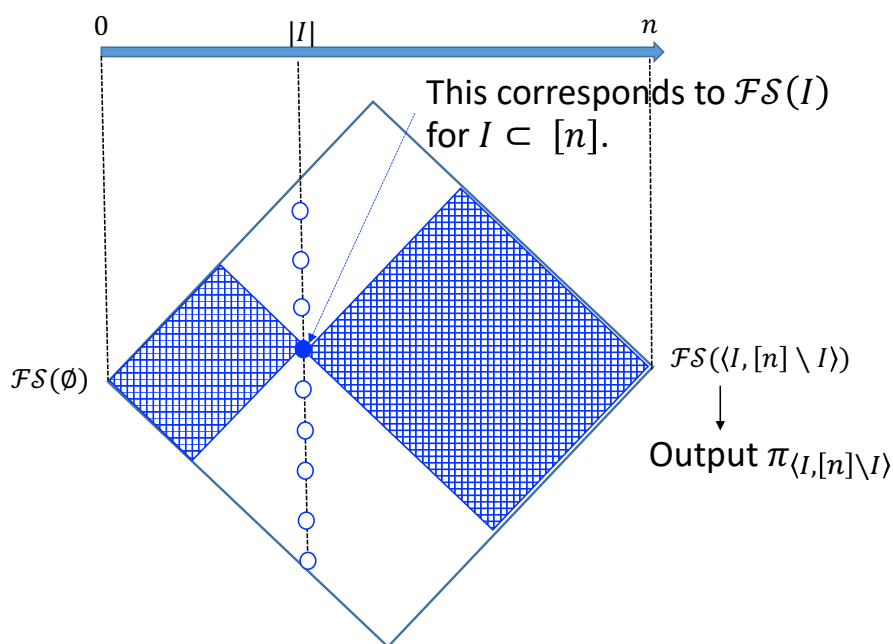
■ **Figure 4** Examples of data structures used in Algorithm FS: TABLE_I and NODE_I with $I = \{1, 3, 5\}$ for the OBDD (rhs) representing $f(x_1, \dots, x_6) = x_1x_2 + x_3x_4 + \dots + x_5x_6$ for the variable ordering $(x_1, x_3, x_5, x_2, x_4, x_6)$. The pointers (integers) to the nodes labeled with x_1, x_3, x_5 are each shown at the top-left positions of the nodes.



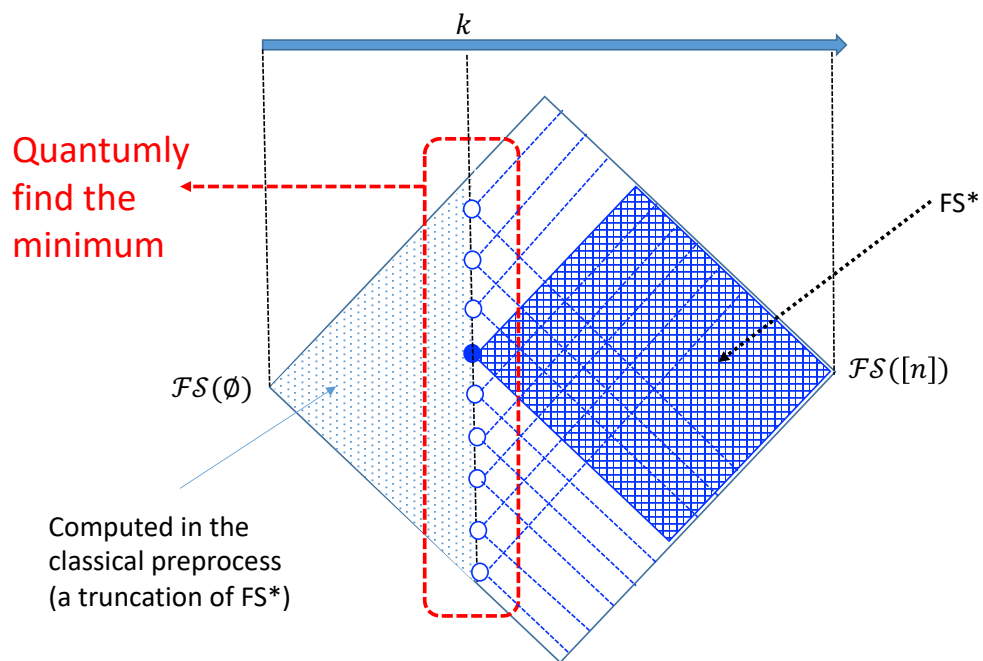
■ **Figure 5** Schematic view of Friedman-Supowit Algorithm. The algorithm goes from the left to the right. On the vertical line indicated by k , there are $\binom{n}{k}$ dots, each of which corresponds to $\mathcal{FS}(I)$ for a subset $I \subseteq [n]$ of size k . $\mathcal{FS}(I)$ is computed from $\mathcal{FS}(\langle I \setminus i \rangle)$ for all $i \in I$, which are arranged as dots on the line indicated by $k - 1$ and have already been computed.



■ **Figure 6** Schematic view of FS^* . This view corresponds to the case where $m = 1$ and $J \subset [n] \setminus I$ in Lemma 8. The shaded area is the one that FS^* sweeps to produce $FS(\langle I, J \rangle)$.



■ **Figure 7** Schematic view of Eq. (4) in Lemma 10. Intuitively, the lemma says that it is possible to decompose FS^* into the parts each of which consists of the two shaded rectangles that share the dot corresponding to $FS(I)$ on the line indicated by $|I|$ for a subset $I \subseteq [n]$ of some fixed size. The optimal variable ordering is induced by one of the parts.



■ **Figure 8** Schematic view of our algorithm in the simplest case (one-parameter case). The dotted area is computed in the classical preprocess, which is realized by truncating the process of FS^* as stated in Lemma 8. The shaded area is computed by using FS^* . The actual algorithm runs the quantum minimum finding, which calls FS^* to coherently compute the shaded area corresponding to every dot on the vertical line indicated by k .

