20th Scandinavian Symposium on Algorithm Theory

SWAT 2026, June 17–19, 2026, Copenhagen, Denmark

Edited by

Pierre Fraigniaud

LIPIcs – Vol. 370 – SWAT 2026 www.dagstuhl.de/lipics

Editors

Pierre Fraigniaud
CNRS and Université Paris Cité, France
pierre.fraigniaud@irif.fr

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

ISBN 978-3-95977-421-5

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/978-3-95977-421-5.

Publication date
June, 2026

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume 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 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/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.2026.0

ISBN 978-3-95977-421-5 **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

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE – Chair)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (IT University of Copenhagen, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Pierre Fraigniaud 0:ix

List of Authors
..... 0:xi

Papers

Classifiers in High Dimensional Hilbert Metrics
Aditya Acharya, Auguste H. Gezalyan, and David M. Mount 1:1–1:16

On the Fragile Complexity of Geometric Algorithms
Boris Aronov, Mayank Goswami, John Iacono, and Indu Ramesh 2:1–2:18

Reachability with Restricted Reactions in Inhibitory Chemical Reaction Networks
Divya Bajaj, Bin Fu, Ryan Knobel, Austin Luchsinger, Aiden Massie, Pablo Santos, Ramiro Santos, Robert Schweller, Evan Tomai, and Tim Wylie 3:1–3:18

Improved Bounds for Online TSP on the Half-Line
Júlia Baligács, Yann Disser, and Linda Thelen 4:1–4:17

Arranging Pairwise Disjoint Shapes to Partition Point Sets
Robert D. Barish and Tetsuo Shibuya 5:1–5:18

How Many Slopes Does Polynomial Area Cost?
Michael A. Bekos, Eleni Katsanou, Philipp Kindermann, and Maria Eleni Pavlidi 6:1–6:18

On the Doubling Dimension and the Perimeter of Geodesically Convex Sets in Fat Polygons
Mark de Berg, Prosenjit Bose, and Leonidas Theocharous 7:1–7:16

Submodular Max-Min Allocation Under Identical Valuations
Kimon Boehmer 8:1–8:17

QPTAS for MWIS and Finding Large Sparse Induced Subgraphs in Graphs with Few Independent Long Holes
Édouard Bonnet, Jadwiga Czyżewska, Tomáš Masařík, Marcin Pilipczuk, and Paweł Rzążewski 9:1–9:14

Robotic Arm Rotation: Standing up Is Harder Than You Think
Nicolas Bousquet, Frank Connor, Remy El Sabeh, Louis-Roy Langevin, Amer E. Mouawad, Naomi Nishimura, and Agnes Totschnig 10:1–10:17

Improved Approximation of Two Watchmen's Routes in Simple Polygons
Anna Brötzner, Bengt J. Nilsson, and Christiane Schmidt 11:1–11:16

Strategy Repair in Reachability Games via a Graph Quotientation
Tiziana Calamoneri, Pierre Gaillard, Giacomo Paesani, and Giuseppe Perelli 12:1–12:16

20th Scandinavian Symposium on Algorithm Theory (SWAT 2026).
Editor: Pierre Fraigniaud
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Faster Algorithms for Shortest Unique or Absent Substrings
Panagiotis Charalampopoulos, Manal Mohamed, Solon P. Pissis, Hilde Verbeek, and Wiktor Zuba 13:1–13:19

Orthogonal Strip Partitioning of Polygons: Lattice-Theoretic Algorithms and Lower Bounds
Jaehoon Chung 14:1–14:17

One Color Makes All the Difference in the Tractability of Partial Coloring in Semi-Streaming
Avinandan Das 15:1–15:16

Online Hitting Set for Axis-Aligned Squares
Minati De, Satyam Singh, and Csaba D. Tóth 16:1–16:18

Incremental Strongly Connected Components with Predictions
Ronald Deng, Samuel McCauley, Aidin Niaparast, Helia Niaparast, Bennett Ptak, Shirel Quintanilla, Shikha Singh, and Nathan Vosburg 17:1–17:16

On Fréchet Traveling Salesmen Problems
Omrit Filtser, Tzalik Maimon, and Michal Moiseev 18:1–18:17

Constant-Factor Approximations for Doubly Constrained Fair k-Center, k-Median and k-Means
Nicole Funk, Annika Hennes, Johanna Hillebrand, and Sarah Sturm 19:1–19:19

Near-Linear and Parameterized Approximations for Maximum Cliques in Disk Graphs
Jie Gao, Paweł Gawrychowski, Panos Giannopoulos, Wolfgang Mulzer, Satyam Singh, Frank Staals, and Meirav Zehavi 20:1–20:17

Dynamic MIS Revisited: Incremental, Fault Tolerant and Fully Dynamic
Manoj Gupta, Shahbaz Khan, and Madhu Surendra 21:1–21:11

Search-Space Reduction for Boolean MinCSPs via Essential Constraints
Bart M. P. Jansen and Ruben F. A. Verhaegh 22:1–22:17

Indexing Range Maximum-Sum Segment Queries with Offsets
Seungbum Jo and Dominik Köppl 23:1–23:16

Cutwidth Versus BFS-Width with Applications to Graph Reconstruction from Distance Queries
Chirag Kaudan and Amir Nayyeri 24:1–24:13

Parameterized Critical Node Cut Revisited
Dušan Knop, Nikolaos Melissinos, and Manolis Vasilakis 25:1–25:17

On the Parameterized Complexity of Min-Sum-Radii
Pankaj Kumar, Haiko Müller, Sebastian Ordyniak, and Melanie Schmidt 26:1–26:18

New Results on Three-Sided Skyline Range Counting and Reporting
Suruchi Kushwaha and Yakov Nekrich 27:1–27:14

The Parameterized Complexity of Coloring Mixed Graphs
Antonio Lauerbach, Konstanty Junosza-Szaniawski, Marie Diana Sieper, and Alexander Wolff 28:1–28:18

Bichromatic Classifications of Points Using Strips
Jaegun Lee, Chaeyoon Chung, and Hee-Kap Ahn 29:1–29:17

Parallel Algorithms for Group Isomorphism via Code Equivalence
Michael Levet 30:1–30:19

Improved and Parameterized Algorithms for Online Multi-Level Aggregation
Young-San Lin and Alex Turoczy 31:1–31:17

Semirandom Planted Bipartite Subgraphs
Anand Louis and Kirtan Vora 32:1–32:13

Polychromatic 2-Colorings with Bounded Discrepancy for Triangulations
Alma Arevalo Loyola, Ahmad Biniaz, Prosenjit Bose, and Thomas Shermer 33:1–33:13

Linear-Time Exact Computation of Influence Spread on Bounded-Pathwidth Graphs
Kengo Nakamura and Masaaki Nishino 34:1–34:17

Path-Reporting Distance Oracles for Vertex-Labeled Graphs
Ofer Neiman and Alon Spector 35:1–35:16

Exact Subquadratic Algorithm for Many-To-Many Matching on Planar Point Sets with Integer Coordinates
Seongbin Park and Eunjin Oh 36:1–36:16

Faster Linear-Space Data Structures for Path Frequency Queries
Ovidiu Rața 37:1–37:17

New Algorithms for Girth and Cycle Detection
Liam Roditty and Plia Trabelsi 38:1–38:17

Faster Approximate Linear Matroid Intersection
Tatsuya Terao 39:1–39:19

Maximum Independent Sets in Disk Graphs with Disks in Convex Position
Anastasiia Tkachenko and Haitao Wang 40:1–40:18

Preface

The Scandinavian Symposium 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 20th edition of the SWAT conference series.

A total of 124 papers were submitted to the conference, two of which were withdrawn a few days after the submission. Among the remaining 122 papers, the program committee selected 40 for presentation at the conference. In addition, the conference program featured two invited lectures, given by Alexandra Lassota (Eindhoven University of Technology), and Oren Weimann (University of Haifa, Israel), the latter being shared speaker with the 37th Annual Symposium on Combinatorial Pattern Matching (CPM 2026). Kimon Boehmer (ENS Paris), and Tatsuya Terao (Kyoto University) received the best student-paper award.

Each submitted paper was reviewed during five weeks by at least three reviewers who were either members of the program committee, or externals, invited as experts by the committee. The selection was performed under double-anonymity (submissions and reviews were anonymous). All papers were then discussed by the program committee members via `easychair` during two additional weeks, and selected or not based on the content of the reviews and the outcome of these discussions. I would like to thank the program committee and the invited experts for their great effort. The program committee consisted of Cristina Bazgan (Université Paris Dauphine, France), Ioana O. Bercea (KTH Royal Institute of Technology, Sweden), Paola Bonizzoni (Università di Milano-Bicocca, Italy), Prosenjit Bose (Carleton University, Canada), Michael Dinitz (Johns Hopkins University, USA), Michal Feldman (Tel-Aviv University, Israel), Omrit Filtser (The Open University of Israel), Pierre Fraigniaud (CNRS and Université Paris Cité, France), Gramoz Goranci (University of Vienna, Austria), Pinar Heggernes (University of Bergen, Norway), Sophie Huiberts (CNRS and Clermont Auvergne University, France), Evangelos Kipouridis (Max-Planck-Institut, Germany), Christian Konrad (University of Bristol, UK), Tuukka Korhonen (University of Copenhagen, Denmark), Robert Krauthgamer (Weizmann, Israel), François Le Gall (Nagoya University, Japan), Paloma T. de Lima (Norwegian School of Economics, Norway and IT University of Copenhagen, Denmark), Frederik Mallmann-Trenn (King's College London, UK), Klaus Meer (Universität Cottbus, Germany), Pedro Montealegre (Universidad Adolfo Ibáñez, Chile), Alantha Newman (Université Grenoble Alpes, France), Irene Parada (UPC Barcelona, Spain), Mor Perry (Academic College of Tel-Aviv-Yaffo, Israel) Seth Pettie (University of Michigan, USA), Andrea Pietracaprina (University of Padua, Italy), R. Ravi (Carnegie Mellon University, USA), Puck Rombach (University of Vermont, USA), Eva Rotenberg (IT University of Copenhagen, Denmark), David Saulpic (CNRS and Université Paris Cité, France), and Anna Zych-Pawlewicz (University of Warsaw, Poland).

The organizing committee was co-chaired by Philip Bille, and Inge Li Gørtz, both from Technical University of Denmark. The SWAT conference series is guided by a steering committee consisting, as in 2026, of Per Austrin (KTH Royal Institute of Technology, Sweden), Fedor Fomin (University of Bergen, Norway), Inge Li Gørtz (Technical University of Denmark), Magnus Mar Halldorsson (Reykjavik University, Iceland), and Jukka Suomela (Aalto University, Finland).

Pierre Fraigniaud
Program Committee Chair SWAT 2026

20th Scandinavian Symposium on Algorithm Theory (SWAT 2026).
Editor: Pierre Fraigniaud
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Aditya Acharya (1)
Department of Computer Science, University of Maryland, College Park, MD, USA

Hee-Kap Ahn (29)
Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Republic of Korea

Boris Aronov (2)
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Divya Bajaj (3)
University of Texas Rio Grande Valley, TX, USA

Júlia Baligács (4)
University of Warsaw, Poland

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

Michael A. Bekos (6)
University of Ioannina, Greece

Ahmad Biniaz (33)
School of Computer Science, University of Windsor, Canada

Kimon Boehmer (8)
DIENS, École normale supérieure, PSL University, Paris, France;
LIP, ENS Lyon, France

Édouard Bonnet (9)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Prosenjit Bose (7, 33)
School of Computer Science, Carleton University, Ottawa, Canada

Nicolas Bousquet (10)
CNRS, Université de Montréal CRM – CNRS, Canada;
Univ. Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69621, Lyon, France

Anna Brötzner (11)
Faculty of Technology and Society, Malmö University, Sweden

Tiziana Calamoneri (12)
Department of Computer Science, Sapienza University of Rome, Italy

Panagiotis Charalampopoulos (13)
King's College London, UK

Chaeyoon Chung (29)
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Jaehoon Chung (14)
Korea Institute for Advanced Study (KIAS), Seoul, Republic of Korea

Frank Connor (10)
McGill University, Montreal, Canada

Jadwiga Czyżewska (9)
University of Warsaw, Poland

Avinandan Das (15)
Aalto University, Finland

Mark de Berg (7)
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Minati De (16)
Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Ronald Deng (17)
Williams College, Williamstown, MA, USA

Yann Disser (4)
TU Darmstadt, Germany

Remy El Sabeh (10)
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Omrit Filtser (18)
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Bin Fu (3)
University of Texas Rio Grande Valley, TX, USA

Nicole Funk (19)
Department of Computer Science, University of Cologne, Germany

20th Scandinavian Symposium on Algorithm Theory (SWAT 2026).
Editor: Pierre Fraigniaud
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Pierre Gaillard (12)
Department of Computer Science, Sapienza University of Rome, Italy;
LaBRI, Université de Bordeaux, France

Jie Gao (20)
Computer Science Department, Rutgers University, Piscataway, NJ, USA

Paweł Gawrychowski (20)
Institute of Computer Science, University of Wrocław, Poland

Auguste H. Gezalyan (1)
Department of Computer Science, University of Maryland, College Park, MD, USA

Panos Giannopoulos (20)
Department of Computer Science, City St George's, University of London, UK

Mayank Goswami (2)
Department of Computer Science, Queens College CUNY, New York, NY, USA

Manoj Gupta (21)
Department of Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India

Annika Hennes (19)
Faculty of Mathematics and Natural Sciences, Heinrich Heine University Düsseldorf, Germany

Johanna Hillebrand (19)
Faculty of Mathematics and Natural Sciences, Heinrich Heine University Düsseldorf, Germany

John Iacono (2)
Université libre de Bruxelles, Ixelles, Belgium

Bart M. P. Jansen (22)
Eindhoven University of Technology, The Netherlands

Seungbum Jo (23)
Chungnam National University, Daejeon, Republic of Korea

Konstanty Junosza-Szaniawski (28)
Warsaw University of Technology, Poland

Eleni Katsanou (6)
National Technical University of Athens, Greece

Chirag Kaudan (24)
Oregon State University, Corvallis, OR, USA

Shahbaz Khan (21)
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Philipp Kindermann (6)
Trier University, Germany

Ryan Knobel (3)
University of Texas Rio Grande Valley, TX, USA

Dušan Knop (25)
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Pankaj Kumar (26)
University of Birmingham, UK

Suruchi Kushwaha (27)
Michigan Technological University, Houghton, MI, USA

Dominik Köppl (23)
University of Yamanashi, Kofu, Japan

Louis-Roy Langevin (10)
McGill University, Montreal, Canada

Antonio Lauerbach (28)
Julius-Maximilians-Universität Würzburg, Germany

Jaegun Lee (29)
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Michael Levet (30)
Department of Computer Science, College of Charleston, SC, USA

Young-San Lin (31)
Melbourne Business School, University of Melbourne, Australia

Anand Louis (32)
Indian Institute of Science, Bengaluru, India

Alma Arevalo Loyola (33)
School of Computer Science, Carleton University, Ottawa, Canada

Austin Luchsinger (3)
University of Texas Rio Grande Valley, TX, USA

Tzalik Maimon (18)
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Tomáš Masařík (9)
University of Warsaw, Poland

Aiden Massie (3)
University of Texas Rio Grande Valley, TX, USA

Samuel McCauley (17)
Williams College, Williamstown, MA, USA

Nikolaos Melissinos (25)
Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Manal Mohamed (13)
King's College London, UK

Michal Moiseev (18)
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Amer E. Mouawad (10)
American University of Beirut, Lebanon; David R. Cheriton School of Computer Science, University of Waterloo, Canada

David M. Mount (1)
Department of Computer Science, University of Maryland, College Park, MD, USA

Wolfgang Mulzer (20)
Institut für Informatik, Freie Universität Berlin, Germany

Haiko Müller (26)
University of Leeds, UK

Kengo Nakamura (34)
Communication Science Laboratories, NTT, Inc., Kyoto, Japan

Amir Nayyeri (24)
Oregon State University, Corvallis, OR, USA

Ofer Neiman (35)
Faculty of Computer and Information Science, Ben-Gurion University of the Negev

Yakov Nekrich (27)
Michigan Technological University, Houghton, MI, USA

Aidin Niaparast (17)
Carnegie Mellon University, Pittsburgh, PA, USA

Helia Niaparast (17)
Carnegie Mellon University, Pittsburgh, PA, USA

Bengt J. Nilsson (11)
Faculty of Technology and Society, Malmö University, Sweden

Naomi Nishimura (10)
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Masaaki Nishino (34)
Communication Science Laboratories, NTT, Inc., Kyoto, Japan

Eunjin Oh (36)
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Sebastian Ordyniak (26)
University of Leeds, UK

Giacomo Paesani (12)
Department of Computer Science, Sapienza University of Rome, Italy

Seongbin Park (36)
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Maria Eleni Pavlidi (6)
University of Ioannina, Greece

Giuseppe Perelli (12)
Department of Mathematics, Sapienza University of Rome, Italy

Marcin Pilipczuk (9)
University of Warsaw, Poland

Solon P. Pissis (13)
The Cyprus Institute, Nicosia, Cyprus; CWI, Amsterdam, The Netherlands; Vrije Universiteit Amsterdam, The Netherlands

Bennett Ptak (17)
Williams College, Williamstown, MA, USA

Shirel Quintanilla (17)
Williams College, Williamstown, MA, USA

Indu Ramesh (2)
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Ovidiu Rața (37)
University of Copenhagen, Denmark

Liam Roditty (38)
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Paweł Rzążewski (9)
Warsaw University of Technology, Poland; University of Warsaw, Poland

Pablo Santos (3)
University of Texas Rio Grande Valley, TX, USA

Ramiro Santos (3)
University of Texas Rio Grande Valley, TX, USA

Christiane Schmidt (11)
Department of Science and Technology, Linköping University, Sweden

Melanie Schmidt (26)
Heinrich Heine University Düsseldorf, Germany

Robert Schweller (3)
University of Texas Rio Grande Valley, TX, USA

Thomas Shermer (33)
School of Computing Science, Simon Fraser University, Burnaby, Canada

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

Marie Diana Sieper (28)
Julius-Maximilians-Universität Würzburg, Germany

Satyam Singh (16, 20)
Department of Computer Science, Aalto University, Espoo, Finland

Shikha Singh (17)
Williams College, Williamstown, MA, USA

Alon Spector (35)
Faculty of Computer and Information Science, Ben-Gurion University of the Negev

Frank Staals (20)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Sarah Sturm (19)
Department of Computer Science, University of Bonn, Germany

Madhu Surendra (21)
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Tatsuya Terao (39)
Research Institute for Mathematical Sciences, Kyoto University, Japan

Linda Thelen (4)
TU Darmstadt, Germany

Leonidas Theocharous (7)
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Anastasiia Tkachenko (40)
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Evan Tomai (3)
University of Texas Rio Grande Valley, TX, USA

Agnes Totschnig (10)
McGill University, Montreal, Canada

Plia Trabelsi (38)
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Alex Turoczy (31)
Department of Computer Science, University of Oxford, UK

Csaba D. Tóth (16)
Department of Mathematics, California State University Northridge, Los Angeles, CA, USA; Department of Computer Science, Tufts University, Medford, MA, USA

Manolis Vasilakis (25)
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Hilde Verbeek (13)
CWI, Amsterdam, The Netherlands

Ruben F. A. Verhaegh (22)
Eindhoven University of Technology, The Netherlands

Kirtan Vora (32)
Indian Institute of Science, Bengaluru, India

Nathan Vosburg (17)
Williams College, Williamstown, MA, USA

Haitao Wang (40)
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Alexander Wolff (28)
Julius-Maximilians-Universität Würzburg, Germany

Tim Wylie (3)
University of Texas Rio Grande Valley, TX, USA

Meirav Zehavi (20)
Institute for the Theory of Computing, Ben Gurion University of the Negev, Beer-Sheva, Israel

Wiktor Zuba (13)
University of Warsaw, Poland

Classifiers in High Dimensional Hilbert Metrics

Aditya Acharya
Department of Computer Science, University of Maryland, College Park, MD, USA

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

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

Abstract

Classifying points in high-dimensional spaces is a fundamental geometric problem in machine learning. In this paper, we address the problem of classifying points in the d-dimensional Hilbert polygonal metric. The Hilbert metric is a generalization of the Cayley-Klein hyperbolic distance to arbitrary convex bodies and has a diverse range of applications in machine learning and convex geometry. We first present an efficient LP-based algorithm in the metric for the large-margin SVM problem. Our algorithm runs in time polynomial in the number of points, the number of bounding facets, and the dimension. This is a significant improvement over previous work, which either provides no theoretical guarantees on runtime or suffers from exponential runtime. We also consider the closely related Funk metric. Finally, we present efficient algorithms for the soft-margin SVM problem and nearest-neighbor-based classification in the Hilbert metric.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Support vector machines, Hilbert geometry, classification, machine learning

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.1

1 Introduction

In machine learning, *binary classification* is a supervised learning task where an algorithm learns to map input data (feature vectors in $\mathbb{R}^d$) to one of two mutually exclusive classes, often denoted as *positive* and *negative*. The primary objective is to train a model that can accurately predict the class of a new, unseen data point. Binary classification is one of the most fundamental geometric problems in machine learning [5, 11, 37, 39]. Support vector machines (SVMs) were introduced by Vapnik and Chervonenkis [41] as a robust binary classification algorithm [10]. Given two sets of *training points* P^+ and P^-, SVM computes a hyperplane that separates the two sets such that the minimum distance from each point set to the separating hyperplane, called the *margin*, is maximized.

SVMs are widely used across diverse fields such as computer vision [26, 40], natural language processing [13, 22], and computational biology [20]. Traditionally, these models are built upon Euclidean geometry. However, there has been a massive recent surge of interest in non-Euclidean spaces, particularly hyperbolic geometries. This interest stems from the unique exponential volume growth property of hyperbolic space, which allows it to embed trees and complex hierarchical structures with minimal distortion. Consequently, hyperbolic representations have become highly effective for modeling word hypernymy and linguistic structures in natural language processing [14, 31], capturing complex user-item interactions and implicit hierarchies in recommendation systems [8], and mapping structural connectivity in brain networks in neuroscience [2]. SVMs have been successfully adapted to these hyperbolic settings to respect this intrinsic geometry [9].

While unbounded spaces (like the hyperbolic Poincaré disk) excel at representing hierarchical data, a vast number of critical domains in modern machine learning are intrinsically constrained within *bounded* convex polytopes. In this paper, we study the SVM problem from

© Aditya Acharya, Auguste H. Gezalyan, and David M. Mount;
licensed under Creative Commons License CC-BY 4.0

the perspective of the Hilbert geometry and the closely related Funk geometry. The Hilbert metric (defined formally in Section 2) generalizes the Cayley-Klein model of hyperbolic geometry to arbitrary bounded convex sets. It boasts several desirable properties, such as straight-line geodesics and invariance under projective transformations [34]. In bounded domains, distances must be acutely sensitive to the proximity of a point to the boundary of the feasible region. The Hilbert metric naturally imposes this penalty, diverging to infinity as points approach the boundary.

The motivation for developing classification algorithms specifically tailored to bounded polytopal boundaries under the Hilbert metric is driven by several pivotal paradigms in contemporary machine learning:

- **Compositional Data and the Probability Simplex:** Compositional data, such as market shares, genomic relative abundances, and topic distributions, consist of nonnegative vectors that sum to one. These vectors naturally reside within a bounded polytope known as the *probability simplex*. Euclidean distances are notoriously ill-suited here because they fail to capture relative variations near the boundaries (i.e., when probabilities approach 0 or 1). The Hilbert metric implicitly respects these compositional constraints and boundary sensitivities, serving as a powerful intrinsic geometry for clustering and classification in the simplex [32, 33].
- **Optimal Transport and the Birkhoff Polytope:** In optimal transport, matching, and assignment problems, the set of feasible transport plans (doubly stochastic matrices) forms a bounded convex domain known as the Birkhoff polytope [29]. Solving machine learning problems over permutations or transport plans frequently requires navigating this highly structured polytope [6,38]. An SVM formulated in the Hilbert metric provides a theoretically sound mechanism for large-margin classification of transport matrices by explicitly respecting the linear constraints that define the Birkhoff polytope.
- **Deep Learning on Symmetric Spaces and Siegel Networks:** There is a growing interest in representation learning on symmetric spaces, such as the bounded Siegel disk or the cone of symmetric positive-definite matrices. These spaces are foundational in advanced architectures such as Siegel networks and are crucial for applications in quantum information theory, medical imaging, and deep learning [21, 24, 27, 36]. The Hilbert metric naturally models the intrinsic geometry of these bounded domains, particularly when they are approximated by high-dimensional bounding facets.
- **Combinatorial Reasoning and Tropical Geometry:** Recent advances at the intersection of machine learning and combinatorial optimization utilize tropical geometry to map discrete algorithms into continuous, piecewise-linear, polyhedral structures. In tropical geometry, feasible solutions and decision boundaries trace the faces of polyhedral complexes [18,43]. Classifying or routing information within these spaces requires a metric that aligns with polyhedral boundaries. A Hilbert SVM provides an exact, geometrically grounded decision boundary for neural algorithmic reasoning over such combinatorial domains.

1.1 Related Work

In the Euclidean metric, SVMs are typically solved by formulating them as convex optimization problems, specifically quadratic programming, which efficiently maximizes the margin between classes [3, 7]. This formulation frequently exploits Lagrangian duality, allowing the problem to be expressed purely in terms of inner products, which can then be elegantly generalized via the kernel trick [19]. However, generalizing this dual formulation to non-Euclidean spaces, such as hyperbolic or Hilbert geometries, is highly non-trivial.

Substituting the Euclidean inner product or distance with a non-Euclidean metric generally destroys the positive semi-definiteness of the kernel or the convexity of the objective function. Consequently, the standard trick of solving the dual problem yields a nonconvex optimization problem when applied naively to these geometries.

Because of this inherent non-convexity, previous attempts to formulate SVMs and linear classifiers in hyperbolic spaces have faced significant computational hurdles. Cho *et al.* [9] were among the first to consider large-margin classification in hyperbolic space. While they demonstrated that their approach significantly outperforms Euclidean classifiers on hierarchical benchmarks by respecting the data's intrinsic geometry, their method directly optimized a non-convex margin objective using Riemannian gradient descent. Without careful initialization, such approaches are highly susceptible to local minima and provide neither a guarantee of a global optimum nor a worst-case polynomial running time. Other continuous optimization techniques for representation learning in hyperbolic spaces similarly rely on iterative local methods lacking rigorous theoretical convergence guarantees [15, 35]. Furthermore, as highlighted by Mishne *et al.* [30], training hyperbolic learning models via Riemannian optimization frequently suffers from severe numerical instability and convergence issues, leading to catastrophic floating-point failures. By formulating our Hilbert SVM as a series of exact LPs, our approach completely obviates these Riemannian convergence and numerical stability bottlenecks.

Another potential avenue for solving SVMs in the Hilbert metric is to exploit isometric embeddings. Extending results of de la Harpe [12], Vernicos [42] presented an elegant isometry mapping the polytopal Hilbert geometry into a finite-dimensional normed vector space. While one might be tempted to map the input points into this embedding space to solve a standard SVM, this approach is fundamentally flawed for linear classification. Because the mapping is highly non-linear, a simple linear classifier (a hyperplane) computed in the embedding space does not translate back to a linear hyperplane in the original input space. Furthermore, computing the complex intersection of the resulting decision boundary with the polytope's feasible image is analytically and computationally impractical.

Recently, Acharya *et al.* [1] addressed the exact polytopal SVM problem in the Hilbert metric without relying on non-convex optimization or non-linear embeddings. They showed that the Hilbert SVM could be modeled precisely as an LP-type problem. While their algorithmic framework achieves a running time linear in the number of training points, its complexity grows polynomially with the total number of vertices of the polytope defining the metric balls. Unfortunately, the vertex complexity of a polytope defined by m facets generally grows exponentially with the dimension d. As such, their method suffers from the curse of dimensionality and remains unsuitable for high-dimensional spaces, where the vast majority of machine learning applications lie.

1.2 Our Contributions

In this paper, we explore an alternative algebraic viewpoint of Hilbert geometry to develop an efficient, globally approximate algorithm for the Hilbert SVM problem. The defining feature of our approach is that the running time scales only *polynomially* with the dimension of the space, successfully circumventing the exponential bottleneck that plagues prior vertex-based methods while avoiding the local-minima pitfalls of Riemannian optimization.

Rather than relying on non-convex gradient descent or explicit vertex enumeration, our algorithm leverages Birkhoff's formulation of the Hilbert metric to reduce the maximum-margin problem to a sequence of linear programming (LP) feasibility tests. Because we rely on numerical LP solvers, we assume the input coordinates and bounding hyperplane coefficients are represented as bounded B-bit rational numbers, and we compute the maximum margin to within a user-supplied additive approximation error ε.

The following theorem formalizes our main result:

▶ **Theorem 1.** *Consider a polytope Ω in $\mathbb{R}^d$ defined by m hyperplanes, and two linearly separable point sets P^+ and P^- of total size n contained within Ω's interior. Assume further that all of the defining coordinates and coefficients are representable as B-bit rational numbers. Then there exists an algorithm running in time $O(poly(n, m, d, B, \log(1/\varepsilon))$ that computes a separating hyperplane whose margin in the Hilbert metric defined by Ω is within an additive error ε of the optimum.*

This polynomial dependence on the ambient dimension d is a crucial advancement for modern machine learning applications. Furthermore, our framework is highly extensible:

Asymmetric Metrics: We generalize our LP-feasibility results to the closely related asymmetric Funk geometry, which is useful for directed spaces and specific optimal transport problems (Section 2.1).

Soft-Margin Classification: To handle real-world datasets that contain noise or are not perfectly linearly separable, we provide a robust soft-margin formulation for the Hilbert SVM that scales penalties based on boundary proximity (Section 4.1).

Nearest-Neighbor Classifiers: While isometric embeddings fail for hyperplane SVMs, we leverage Vernicos's mapping to design an efficient approximate nearest-neighbor-based classifier for non-linear decision boundaries (Section 4.2).

2 Mathematical Preliminaries

In this section, we provide a number of definitions and facts, which will be used throughout. We will use different fonts, specifically p and $\mathbf{p}$, to distinguish between a point $p \in \mathbb{R}^d$ and its associated d-dimensional column vector $\mathbf{p}$. We represent a $(d-1)$-dimensional flat, that is a hyperplane, L in $\mathbb{R}^d$ as a pair $(\mathbf{w}, c) \in \mathbb{R}^d \times \mathbb{R}$ represented by the function $f_L(x) = \mathbf{w}^\top \mathbf{x} + c$. A hyperplane partitions space into three sets, L^-, L, L^+, where a point $\mathbf{x} \in \mathbb{R}^d$ belongs to each of these sets depending on whether $f_L(x)$ is < 0, $= 0$, or > 0, respectively.

Given a set of m hyperplanes, $\mathcal{L} = \{L_i : (\mathbf{w}_i, c_i)\}_{i=1}^m$ let $\Omega = \Omega(\mathcal{L})$ denote the closure of the space bounded by the positive half-spaces L_i^+, that is, $\Omega = \mathrm{cl}\big(\bigcap_{i=1}^m L_i^+\big)$. Let $\partial\Omega$ denote the boundary of Ω. Throughout, we assume that Ω is bounded and full-dimensional. Given a nonnegative integer m, let $[m]$ denote the index set $\{1, \ldots, m\}$.

Given two points $p, q \in \mathrm{int}(\Omega)$, let $\overline{pq}$ denote the *chord* of Ω passing through these points, that is, the intersection of Ω with the line through p and q. Let $\overleftrightarrow{pq}$ denote the line that passes through them, and $\overrightarrow{pq}$ denote the ray starting from p and passing through q.

Given $p, q \in \mathbb{R}^d$, let $d(p, q)$ denote their Euclidean distance. Given a hyperplane L, let $d(p, L)$ denote the minimum Euclidean (perpendicular) distance from p to L.

2.1 Hilbert Geometry

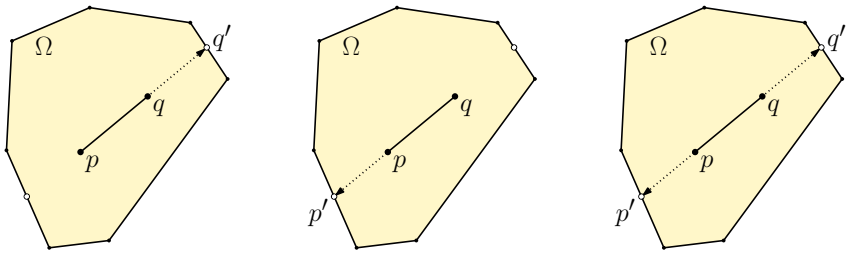
In this section, we introduce the Funk and Hilbert distances. Given a convex body Ω and any two points $p, q \in \mathrm{int}(\Omega)$, let p' and q' denote the endpoints of the chord $\overline{pq}$ ordered as $\langle p', p, q, q' \rangle$ (see Figure 1). If $p = q$ all the distances are defined to be zero. Otherwise, the Funk and Hilbert distances are defined as:

$$\text{Funk:} \quad d_\Omega^F(p, q) := \ln \frac{\|p - q'\|}{\|q - q'\|} \tag{1}$$

$$\text{Reverse Funk:} \quad d_\Omega^{rF}(p, q) := d_\Omega^F(q, p) = \ln \frac{\|q - p'\|}{\|p - p'\|} \tag{2}$$

$$\text{Hilbert:} \quad d_\Omega^H(p, q) := \frac{d_\Omega^F(p, q) + d_\Omega^{rF}(p, q)}{2} = \frac{1}{2} \ln \frac{\|q - p'\|}{\|p - p'\|} \frac{\|p - q'\|}{\|q - q'\|}. \tag{3}$$

Intuitively, $d_\Omega^F(p,q)$ is a measure of how far q is from p relative to the boundary that lies beyond q, and $d_\Omega^{rF}(p,q)$ is just the reverse of this. The Hilbert distance $d_\Omega^H(p,q)$ symmetrizes these by taking their arithmetic mean.



(a) (b) (c)

Figure 1 The (a) forward Funk, (b) reverse Funk, (c) and Hilbert distances between p and q in Ω.

These distance functions are all nonnegative and satisfy the triangle inequality [34]. The Hilbert distance defines a *metric* over $\text{int}(\Omega)$. The Funk and reverse Funk are asymmetric and hence define *weak metrics* over $\text{int}(\Omega)$. Observe that as either point approaches the boundary of Ω, the Hilbert distance approaches ∞. The Hilbert distance is invariant under invertible projective transformations.

2.2 Birkhoff's Formulation and Metric Balls

Birkhoff [4] provided an alternative, equivalent definition of these metrics that is better suited to our approach.

▶ **Proposition 2** (Birkhoff's Formulation). *Given a set of m hyperplanes, $\mathcal{L} = \{L_i : (\mathbf{w}_i, c_i)\}_{i=1}^m$, then for any pair of points $p, q \in \text{int}(\Omega(\mathcal{L}))$:*

$$d_\Omega^F(p,q) = \max_{i\in[m]} \log \frac{d(p,L_i)}{d(q,L_i)} = \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i}$$

$$d_\Omega^{rF}(p,q) = \max_{i\in[m]} \log \frac{d(q,L_i)}{d(p,L_i)} = \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{q} + c_i}{\mathbf{w}_i^\top \mathbf{p} + c_i}$$

$$d_\Omega^H(p,q) = \frac{1}{2} \max_{j,k\in[m]} \log \frac{d(p,L_j)}{d(q,L_j)} \frac{d(q,L_k)}{d(p,L_k)} = \frac{1}{2} \max_{j,k\in[m]} \log \left(\frac{\mathbf{w}_j^\top \mathbf{p} + c_j}{\mathbf{w}_j^\top \mathbf{q} + c_j} \frac{\mathbf{w}_k^\top \mathbf{q} + c_k}{\mathbf{w}_k^\top \mathbf{p} + c_k} \right).$$

This follows from the fact that Ω is convex, and observing that if $\overrightarrow{pq}$ intersects $\partial\Omega$ in q', on the bounding hyperplane L_k, then by similar triangles

$$\frac{d(p,q')}{d(q,q')} = \frac{d(p,L_i)}{d(q,L_i)}.$$

Intuitively, this provides a way to define the metrics without explicitly calculating where the line $\overleftrightarrow{pq}$ intersects $\partial\Omega$. One consequence of the proposition is that the metrics can be reduced to a sup norm over $\mathbb{R}^t$ for some finite dimension t, using a non-linear mapping function [28]. Another useful consequence is its application to constructing metric balls in high dimensions.

Letting d_Ω^* denote any of the above metrics and given a point $p \in \text{int}(\Omega)$ and scalar $r \geq 0$, the associated metric balls are defined in the standard way.

$$B_\Omega^*(p,r) := \{q \in \Omega : d_\Omega^*(p,q) \leq r\}.$$

It is well known that the Funk ball about a point p is a scaling of Ω about p by an appropriate factor that depends on the radius. This can be derived directly from Proposition 2:

$$\begin{aligned} B^F_\Omega(p,r) &= \left\{q \in \Omega \,:\, d^F_\Omega(p,q) \leq r\right\} \\ &= \left\{q \in \Omega \,:\, \max_{i\in[m]} \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i} \leq r\right\} \\ &= \left\{q \in \Omega \,:\, \log \frac{\mathbf{w}_i^\top \mathbf{p} + c_i}{\mathbf{w}_i^\top \mathbf{q} + c_i} \leq r,\ \forall i \in [m]\right\} \\ &= \left\{q \in \Omega \,:\, 0 \leq \mathbf{w}_i^\top \mathbf{q} + c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i),\ \forall i \in [m]\right\} \\ &= \bigcap\nolimits_{i\in[m]} \left\{q \in \Omega \,:\, 0 \leq \mathbf{w}_i^\top \mathbf{q} + c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i)\right\}. \end{aligned} \tag{4}$$

Fixing p and r, let $c^*_i := c_i - e^{-r}(\mathbf{w}_i^\top \mathbf{p} + c_i)$, and let S^F_i be the hyperplane represented by $(\mathbf{w}_i, c^*_i)$ (See Figure 2(a)). Observe that S^F_i is parallel to L_i. We have the following:

▶ **Lemma 3** (Funk Balls). *Given a set of m hyperplanes, $\mathcal{L}$ as in Proposition 2, the Funk ball $B^F_{\Omega(\mathcal{L})}(p,r)$ is a polytope bounded by the m hyperplanes, $S^F_i : (\mathbf{w}_i, c^*_i)$, for $i \in [m]$.*

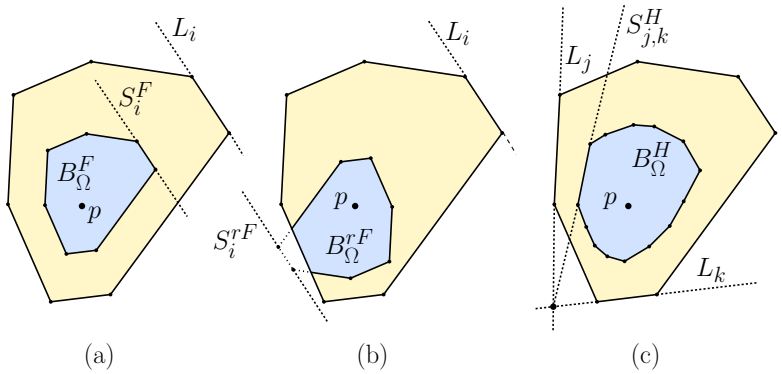


Figure 2 Defining balls and their bounding hyperplanes for (a) Funk, (b) Reverse Funk, and (c) Hilbert.

A similar result can be proved for reverse-Funk balls (see Figure 2(b)). We next use Proposition 2 to demonstrate the Hilbert balls are convex polytopes, but the number of bounding facets is larger by the square. Following the same reasoning as in Eq. (4), we have

$$\begin{aligned} B^H_\Omega(p,r) &= \{q \in \Omega \,:\, d^H_\Omega(p,q) \leq r\} \\ &= \left\{q \in \Omega \,:\, \frac{1}{2} \max_{j,k\in[m]} \log \left(\frac{\mathbf{w}_j^\top \mathbf{p} + c_j}{\mathbf{w}_j^\top \mathbf{q} + c_j} \frac{\mathbf{w}_k^\top \mathbf{q} + c_k}{\mathbf{w}_k^\top \mathbf{p} + c_k}\right) \leq r\right\} \\ &= \bigcap_{j,k\in[m]} \left\{q \in \Omega \,:\, \left(\mathbf{w}_j^\top \mathbf{p} + c_j\right)\left(\mathbf{w}_k^\top \mathbf{q} + c_k\right) \leq e^{2r}\left(\mathbf{w}_k^\top \mathbf{p} + c_k\right)\left(\mathbf{w}_j^\top \mathbf{q} + c_j\right)\right\}. \end{aligned}$$

Fixing p and r, let $\alpha_{j,k} := e^{2r}\left(\mathbf{w}_k^\top \mathbf{p} + c_k\right)$ and $\alpha'_{j,k} := \left(\mathbf{w}_j^\top \mathbf{p} + c_j\right)$. We have

$$B^H_\Omega(p,r) = \bigcap\nolimits_{j,k\in[m]} \left\{q \in \Omega \,:\, 0 \;\leq\; \left(\alpha_{j,k}\mathbf{w}_j - \alpha'_{j,k}\mathbf{w}_k\right)^\top \mathbf{q} + \left(\alpha_{j,k}c_j - \alpha'_{j,k}c_k\right)\right\}. \tag{5}$$

Let $\mathbf{w}^*_{j,k} = \left(\alpha_{j,k}\mathbf{w}_j - \alpha'_{j,k}\mathbf{w}_k\right)$, and $c^*_{j,k} = \left(\alpha_{j,k}c_j - \alpha'_{j,k}c_k\right)$. Let $S^H_{j,k}$ be the hyperplane represented by $(\mathbf{w}^*_{j,k}, c^*_{j,k})$. Observe that $S^H_{j,k}$ is a linear combination of L_j, and L_k, and therefore $S^H_{j,k}$ passes through the intersection of L_j and L_k (see Figure 2(c)). In summary, we have the following:

▶ **Lemma 4** (Hilbert Balls). *Given a set of m hyperplanes, $\mathcal{L}$ as in Proposition 2, the Hilbert ball $B^H_{\Omega(\mathcal{L})}(p,r)$ is a polytope bounded by the at most $m(m-1)$ hyperplanes $S^H_{j,k}$, for $j,k \in [m]$.*

3 Hilbert SVM

In this section, we present our solution to the SVM problem in the Hilbert geometry. Recall that our objective is to compute a maximum-margin hyperplane (with respect to the Hilbert distance) that separates two point sets P^+ and P^- of total size n.

We are given an index set $I = \{1, \ldots, n\}$, and a partition of I into I^+, and I^-, that is $I^+ \cup I^- = I$, $I^+ \cap I^- = \emptyset$, along with the associated d-dimensional point sets: $P^{\pm} = \{p_i : i \in I\}$, $P^+ = \{p_i : i \in I^+\}$, $P^- = \{p_i : i \in I^-\}$. Throughout this section, we assume P^+ and P^- are linearly separable, that is, there is a hyperplane K such that $P^+ \subset K^+$, and $P^- \subset K^-$. (In Section 4.1, we will consider the general case.)

Define the Hilbert distance of a point $p \in \Omega$ to a hyperplane L to be $d^H_\Omega(L, p) = \min_{x \in L} d^H_\Omega(x, p)$. We define the *margin* of P^+ and P^- with respect to a separating L as $\min_{p \in P^{\pm}} d^H_\Omega(L, p)$, the minimum distance of any point to L. Finding the maximum margin of any separating hyperplane can be expressed formally as

$$\max \left\{ \gamma : \exists K \text{ such that } P^+ \subset K^+, P^- \subset K^-, \text{ and } d^H_\Omega(K, p) \geq \gamma, \ \forall p \in P^{\pm} \right\}. \tag{6}$$

Our algorithm will yield an absolute approximation to the optimal solution. In particular, given an *approximation parameter* $\varepsilon > 0$, our algorithm will produce a separating hyperplane that achieves a margin of at least $\gamma - \varepsilon$, where γ is the optimal margin.

Assuming K is represented by the pair $(\mathbf{w}, c)$, this can be expressed equivalently as the following optimization problem:

$$\begin{aligned} \text{maximize:} \quad & \gamma \\ \text{subject to:} \quad & \mathbf{w}^\top \mathbf{p} + c > 0, \quad \forall p \in P^+ \\ & \mathbf{w}^\top \mathbf{p} + c < 0, \quad \forall p \in P^- \\ & d^H_\Omega(K, p) \geq \gamma, \quad \forall p \in P^{\pm}, \quad K : (\mathbf{w}, c). \end{aligned} \tag{7}$$

It is not immediately straightforward as to how to solve Opt (7). One difficulty lies in finding a reasonable representation for $d^H_\Omega(K, p)$, the Hilbert distance of a point to a hyperplane. This was done in an earlier work on the SVM in the polytopal Hilbert geometries [1], but the size of the representation given there depended on the number of vertices in the Hilbert ball, which grows exponentially with the dimension. Other ways of formulating the SVM problem in hyperbolic spaces usually mirror that in the Euclidean setting, framing it as a non-convex optimization problem, with no theoretical guarantees on the runtime [9]. However, by exploiting the geometry of the polytopal Hilbert metric and using the alternative viewpoints of the metrics as presented in Proposition 2, we can significantly improve upon prior attempts.

3.1 Overall Algorithm

We assume every input parameter is given to us as a rational number, expressed as a fraction of two nonzero B-bit integers. We call this representation a *B-bit rational number*. Each of the n points $p \in P^{\pm}$ is represented as a d-vector $\mathbf{p}$ whose coordinates are B-bit rational numbers. The domain Ω is defined by a set $\mathcal{L}$ of m hyperplanes, each of whose $(\mathbf{w}, c)$ representation is given as a sequence of $d + 1$ B-bit rational numbers. We say that the resulting SVM problem instance has *size parameters* (d, n, m, B). The total *bit complexity* of such an instance is $O(d(n + m)B)$.

Before describing the overall algorithm, we first establish an upper bound on γ, the maximum margin in terms of the bit size of the input. Let K be the corresponding separating hyperplane for which γ is achieved. Observe that for an arbitrary $p_+ \in P^+$, and $p_- \in P^-$,

we have $d^H_\Omega(p_-, p_+) \geq d^H_\Omega(p_-, K) + d^H_\Omega(p_+, K) \geq 2\gamma$. Now let the line joining p_- and p_+ intersect $\partial\Omega$ in bounding hyperplanes $L_- : (\mathbf{w}_-, c_-)$, and $L_+ : (\mathbf{w}_+, c_+)$. Therefore,

$$d^H_\Omega(p_-, p_+) \;=\; \frac{1}{2}\log\frac{d\,(p_+, L_-)\,d\,(p_-, L_+)}{d\,(p_+, L_+)\,d\,(p_-, L_-)} \;=\; \frac{1}{2}\log\frac{\left(\mathbf{w}_-^\top \mathbf{p}_+ + c_-\right)\left(\mathbf{w}_+^\top \mathbf{p}_- + c_+\right)}{\left(\mathbf{w}_-^\top \mathbf{p}_- + c_-\right)\left(\mathbf{w}_+^\top \mathbf{p}_+ + c_+\right)}.$$

Consider any one of the terms in the numerator, say, $(\mathbf{w}_-^\top \mathbf{p}_+ + c_-)$. Now, $\mathbf{w}_-$, and p_+ are given by d B-bit rational numbers, therefore their inner product can have an absolute value of at most $d \cdot 2^{2B}$. Hence, the entire term $\mathbf{w}_-^\top \mathbf{p}_+ + c_-$ has an absolute value at most $d \cdot 2^{2B} + 2^B$. Recalling that the Hilbert distance is the logarithm of ratios, up to constant factors, $d^H_\Omega(p_-, p_+)$ is at most $(8B + 4\log d)\log 2 = O(B + \log d)$, implying that $\gamma \leq M$ for some $M \in O(B + \log d)$. Thus, we have:

▶ **Lemma 5.** *Given an SVM instance with size parameters (d, n, m, B), the separation margin γ for any hyperplane is at most $M(d, n, m, B) = O(B + \log d)$.*

Our overall algorithm is as follows. Since we are interested in the optimum margin up to an accuracy of ε, we check if P^+ and P^- are separable with a margin of at least r, where r is an element from the range: $\Gamma = [\varepsilon, 2\varepsilon, 3\varepsilon, \ldots, M]$, where $M = M(d, n, m, B)$. To check separation for a particular value r, we will conduct an (approximate) LP-based feasibility test. We employ this test as part of a parametric binary search over the range Γ. This implies that we only need to solve $O(\log B + \log\log d + \log(1/\varepsilon))$ many feasibility tests. We next describe this LP-based subroutine.

3.2 LP-feasibility

We consider the problem of determining whether it is possible to achieve a given margin γ, up to an additive error of ε. For a particular margin r, recall that $B^H_\Omega(p, r)$ is the Hilbert ball of radius r centered at p. By Lemma 4, $B^H_\Omega(p, r)$ is a polytope bounded by $m(m-1)$ hyperplanes. (In Section 3.3, we will show how to construct them efficiently.) Let $\mathcal{B}(r)^+ = \{B^H_\Omega(p^+, r) \mid p^+ \in P^+\}$, the collection of the Hilbert balls of radius r around the positively labeled points, and define $\mathcal{B}(r)^-$ analogously for P^-.

The feasibility of γ reduces to determining whether the collections of balls $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ are linearly separable. For $i \in [n]$, let $\mathbf{A}(r, i)$ be an $m(m-1) \times d$ matrix and let $\mathbf{b}(r, i)$ be an $m(m-1) \times 1$ vector such that

$$\mathbf{x} \in B^H_\Omega(p_i, r) \iff \mathbf{A}(r, i)\mathbf{x} + \mathbf{b}(r, i) \geq 0.$$

We can construct $\mathbf{A}(r, i)$ and $\mathbf{b}(r, i)$ from the hyperplanes $S^H_{j,k} : (\mathbf{w}^*_{j,k}, c^*_{j,k})$ described in Lemma 4. In particular, for $t \in [m(m-1)]$, if $(\mathbf{w}^*_t, c^*_t)$ denotes the t-th bounding hyperplane of $B^H_\Omega(p_i, r)$, then $\mathbf{w}^*_t$ is the t-th row of $\mathbf{A}(r, i)$ and c^*_t is the t-th element of $\mathbf{b}(r, i)$.

Therefore, finding whether $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ are linearly separable is equivalent to the following decision problem:

$$\begin{aligned}
&\exists K : (\mathbf{w}, c), \quad \text{such that} \\
&\qquad \forall p_i \in P^+,\ \mathbf{w}^\top\mathbf{x} + c \geq 0, \quad \forall \mathbf{x} : \mathbf{A}(r, i)\mathbf{x} + \mathbf{b}(r, i) \geq 0, && (8)\\
&\qquad \forall p_i \in P^-,\ \mathbf{w}^\top\mathbf{x} + c \leq 0, \quad \forall \mathbf{x} : \mathbf{A}(r, i)\mathbf{x} + \mathbf{b}(r, i) \geq 0. && (9)
\end{aligned}$$

For fixed r and i, let $\mathbf{A} = \mathbf{A}(r, i)$ and $\mathbf{b} = \mathbf{b}(r, i)$. Eq. (8) can be rewritten as:

$$\min_{\{\mathbf{x}:\ \mathbf{Ax}+\mathbf{b}\geq 0\}} \mathbf{w}^\top\mathbf{x} + c \;\geq\; 0.$$

By standard LP duality (see, e.g., [16, pp. 81–84]), we have

$$\min_{\{\mathbf{x}:\ \mathbf{Ax}+\mathbf{b}\geq 0\}} \mathbf{w}^\top \mathbf{x} \ = \ \max_{\{\mathbf{y}:\ \mathbf{A}^\top \mathbf{y}=\mathbf{w},\ \mathbf{y}\geq \mathbf{0}\}} -\mathbf{b}^\top \mathbf{y}.$$

Therefore we can simplify Eq. (8) as follows:

$$\begin{aligned}
&\mathbf{w}^\top \mathbf{x} + c \ \geq \ 0, \quad \forall \mathbf{x} : \mathbf{Ax} + \mathbf{b} \geq 0 \\
&\qquad \iff \left(\min_{\{\mathbf{x}:\ \mathbf{Ax}+\mathbf{b}\geq 0\}} \mathbf{w}^\top \mathbf{x} + c\right) \ \geq \ 0 \\
&\qquad \iff \left(\max_{\{\mathbf{y}:\ \mathbf{A}^\top \mathbf{y}=\mathbf{w},\ \mathbf{y}\geq \mathbf{0}\}} -\mathbf{b}^\top \mathbf{y} + c\right) \ \geq \ 0 \\
&\qquad \iff \exists\, \mathbf{y}(i) : \ \mathbf{A}(r,i)^\top \mathbf{y}(i) \ = \ \mathbf{w}, \quad \mathbf{y}(i) \geq \mathbf{0}, \quad \mathbf{b}(r,i)^\top \mathbf{y}(i) \leq c.
\end{aligned}$$

Analogously, Eq. (9) can be simplified to

$$\exists\, \mathbf{y}(i) : \ \mathbf{A}(r,i)^\top \mathbf{y}(i) \ = \ \mathbf{w}, \quad \mathbf{y}(i) \geq \mathbf{0}, \quad \mathbf{b}(r,i)^\top \mathbf{y}(i) \geq c.$$

Hence, to determine whether the collections $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ are linearly separable we solve the following LP-feasibility problem in the parameters $\mathbf{w}_{d\times 1}$, c, and $\mathbf{y}(i)_{m(m-1)\times 1}$, for $i \in [n]$.

$$\begin{aligned}
&\text{Minimize:} && 0 && \\
&\text{Subject to:} && \mathbf{b}(r,i)^\top \mathbf{y}(i) \ \leq \ c, && \forall i \in I^+ \\
& && \mathbf{b}(r,i)^\top \mathbf{y}(i) \ \geq \ c, && \forall i \in I^- \\
& && \mathbf{A}(r,i)^\top \mathbf{y}(i) \ = \ \mathbf{w}, && \forall i \in I \\
& && \mathbf{y}(i) \ \geq \ \mathbf{0}, && \forall i \in I \\
& && \mathbf{1}^\top \mathbf{w} \ = \ 1. && \qquad (10)
\end{aligned}$$

We add the final constraint to avoid the trivial solution in which all the parameters are zero.

▶ Remark 6 (Intuitive Description). Opt. (10) can be understood intuitively by observing the following. $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$ being linearly separable implies there exists a hyperplane $K : (\mathbf{w}, c)$, such that for every $B^+ \in \mathcal{B}(r)^+$, there is a hyperplane K_{B^+} that is parallel to and above K, and is tangent to some ball in B^+. Similarly, for every $B^- \in \mathcal{B}(r)^-$, there is a hyperplane K_{B^-} that is parallel to and below K, and is tangent to some ball in B^- (see Figure 3).

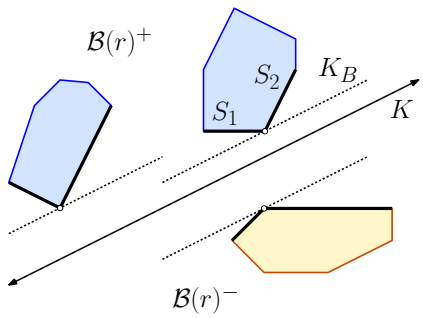


Figure 3 Intuitive description of the dual: K separates $\mathcal{B}(r)^+$ and $\mathcal{B}(r)^-$. K_B is a conical combination of S_1 and S_2 while lying strictly above K.

A hyperplane L that is tangent to a polytope Ω, with bounding hyperplanes $L_1, \ldots, L_m$, can be written as a positive linear combination (often referred to as the conical combination) of the bounding hyperplanes.

The vector $\mathbf{y}(i)$ in Opt. (10) corresponds to the factors for this conical combination. The third constraint in Opt. (10) guarantees a hyperplane K_B parallel to K, along with being a conical combination of the supporting hyperplanes for each $B \in \mathcal{B}(r)^+ \cup \mathcal{B}(r)^-$. The first two constraints ensure K_B is above or below K, depending on whether $B \in \mathcal{B}(r)^+$ or $\mathcal{B}(r)^-$.

▶ Remark 7 (LP for Funk). The LP feasibility formulation for the Funk metric is similar to Opt. (10). We briefly mention the required modifications. By the construction given in Lemma 3, for $i \in [n]$, let $\mathbf{A}(r,i)$ be an $m \times d$ matrix and let $\mathbf{b}(r,i)$ be an $m \times 1$ vector such that

$$\mathbf{x} \in B^F_\Omega(p_i, r) \iff \mathbf{A}(r,i)\mathbf{x} + \mathbf{b}(r,i) \geq 0.$$

For this case, $\mathbf{y}(i)$ will be an $m \times 1$ vector. The remainder of the analysis and formulation is exactly the same as in the case of the Hilbert metric.

3.3 Constructing Hilbert Balls

In this section, we present a procedure for constructing the bounding hyperplanes for a Hilbert ball, $B^H_\Omega(p_i, r)$. Recall that $\mathcal{L} = \{L_i\}_{i=1}^m$ denotes the set of hyperplanes bounding Ω. First, we take two bounding hyperplanes of Ω: $L_j : (\mathbf{w}_j, c_j)$ and $L_k : (\mathbf{w}_k, c_k)$, $j \neq k$, $j, k \in [m]$. For $p_i \in P^\pm$, we find $S(i,r)_{j,k} : (\mathbf{u}(i,r)_{(j,k)}, t(i,r)_{j,k})$, the hyperplane at a Hilbert-distance of r from p_i with respect to the boundary defined by L_j and L_k, such that there exists a line ℓ that intersects L_j, $S(i,r)_{j,k}$, p_i, and L_k in that order (see Figure 4).

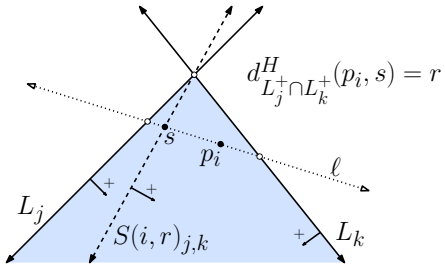


Figure 4 Constructing $B^H_\Omega(p_i, r)$.

It is clear that $\exists\ \alpha : 0 \leq \alpha \leq 1$, such that

$$S(i,r)_{j,k} \;=\; \alpha L_j - (1-\alpha)L_k. \tag{11}$$

For any point s on $S(i,r)_{j,k}$, $s \in L_j^+ \cap L_k^+$, it is easy to see that

$$\frac{d(s, L_k)}{d(s, L_j)} \;=\; \frac{\alpha}{1-\alpha}.$$

Solving for α using the following:

$$\exp(2 \cdot d^H_\Omega(p_i, S(i,r)_{j,k})) \;=\; \frac{\mathbf{w}_j^\top \mathbf{p}_i + c_j}{\mathbf{w}_k^\top \mathbf{p}_i + c_k} \cdot \frac{\alpha}{1-\alpha} \;=\; \exp(2r). \tag{12}$$

For $p_i \in \text{int}(\Omega)$, r is finite and positive: $0 < r < \infty$. Therefore, Eq. (12) can be equivalently written as a linear equation in α. Since for $j, k \in [m]$, $j \neq k$, $i \in [n]$, $S(i,r)_{j,k}$ is a bounding hyperplane for $B^H_\Omega(p_i, r)$, we can use Eqs. (11), and (12) to construct $B^H_\Omega(p_i, r)$, for all $p_i \in P^\pm$ in $O(ndm^2)$ time.

▶ Remark 8 (Funk Balls). We can use a similar approach to construct $B^F_\Omega(p_i, r)$, the Funk ball of radius r around p_i. Recall that the Funk ball around p_i is a scaled copy of Ω around p_i. Therefore for a bounding hyperplane of Ω, $L_j : (\mathbf{w}_j, c_j)$, $j \in [m]$, we have $S(i,r)_j = (\mathbf{w}_j, c^*_j(i,r))$ as a bounding hyperplane of $B^F_\Omega(p_i, r)$ for some constant $c^*_j(i,r)$. We solve for $c^*_j(i,r)$ using the following equation:

$$\exp(d^F_\Omega(p_i, S(i,r)_j)) \;=\; \frac{\mathbf{w}_j^\top \mathbf{p}_i + c_j}{\mathbf{w}_j^\top \mathbf{p}_i + c^*_j(i,r)} \;=\; \exp(r). \tag{13}$$

As before, for any $r \geq 0$, this can be expressed as a linear function in $c^*(i,r)$. Therefore, we can construct $B^F_\Omega(p_i, r)$, for all $p_i \in P^\pm$ in $O(nmd)$ time.

3.4 Complexity Analysis

Recall that to obtain γ within an additive error of ε, we solve $O(\log B + \log\log d + \log(1/\varepsilon))$ LPs of the form Opt. (10). We can use any standard interior-point method, such as Khachiyan's Ellipsoid method [25] or Karmarkar's method [23] to solve Opt. (10) in time that is polynomial in the number of variables, number of inequalities, and bit-length of each variable. We first establish the bit-length required to encode a parameter in our LP-feasibility problem (Opt. (10)).

Recall that we are searching for r over the range $\Gamma = [\varepsilon, 2\varepsilon, \ldots, M]$, with $M = M(d, n, m, B) = O(B + \log d)$. Therefore, the quantity e^{2r} can take on values in the range $\Gamma_e = [e^{2\varepsilon}, e^{4\varepsilon}, \ldots, e^{2M}]$. Instead of fixing r's value exactly over Γ, it is sufficient for our purposes if we search for r over $[r_1, r_2, \ldots, r_{M/\varepsilon}]$, where $(i-1)\varepsilon \leq r_i < i\varepsilon$.

The minimum difference between two elements in Γ_e is at least $e^{4\varepsilon} - e^{2\varepsilon} > \varepsilon$. (This can be verified using Taylor's series for e^x, at $x = 0$.) Therefore, to represent Γ_e, it is sufficient for our purposes to find a bit-representation which allows for a maximum value e^{2M} up to a resolution of ε. This can be done by using $O(M)$ bits for the integer part, and $O(\log(1/\varepsilon))$ bits for the fractional part. Hence, it is sufficient to represent the elements of Γ_e as $O(M + \log(1/\varepsilon)) = O(B + \log d + \log(1/\varepsilon))$-bit rational numbers.

Using Eq. (12), we solve for and represent α, using $O(B + \log d + \log(1/\varepsilon))$-bit rational numbers. Therefore, using Eq. (11) every parameter of $S(i,r)_{j,k}$, which acts as a bounding hyperplane for $B^H_\Omega(p_i, r)$, can also be represented using $O(B + \log d + \log(1/\varepsilon))$-bit rational numbers.

Now, the number of variables in Opt. (10) is $O((n+d)m^2)$, with the number of inequalities being $O(n(m^2 + d))$. Hence, we can determine if Opt. (10) is feasible in $O(\text{poly}(n, m, d)(B + \log d + \log(1/\varepsilon))$ time, for a fixed r. And since we solve $O(\log B + \log\log d + \log(1/\varepsilon))$ such LPs, our overall running time is $O(\text{poly}(n, m, d, B, \log(1/\varepsilon))$. This proves our main result in Theorem 1. Recall that the input bit-length is $d(n+m)B$, implying the overall running time is also polynomial in the input bit-length and the log of the desired accuracy.

4 Extensions

In this section, we will explore two extensions to binary classification and SVM problem in the Hilbert geometry. The first is a soft-margin classifier, which is applicable when the point sets are not linearly separable, and the second is a binary classifier based on nearest neighbors.

4.1 A Soft-Margin Classifier

Up to this point, we assumed P^+ and P^- are linearly separable. If they are not, a common approach is to use a hyperplane classifier with appropriate penalties for misclassifying points in the training set. In this section, we propose a formulation for such a "soft" classifier.

We modify our LP-feasibility problem: Opt (10) to introduce penalties in the form of slack variables. Reiterating the observation from Remark 6, we note that the first two constraints of Opt (10) enforce the condition that, for $p \in P^+$, $B^H_\Omega(p, r)$ is completely above the separator $K : (\mathbf{w}, c)$. Instead, we only require that $B^H_\Omega(p, r)$ is above $K' : (\mathbf{w}, c + \xi)$ for some positive ξ. Here ξ works as a penalty for the improper separation of $B^H_\Omega(p, r)$. We similarly set individual penalties for each point.

Next, we want the following desirable property: points that are closer to the boundary of Ω should pay a heavier penalty for being misclassified. It is because for a fixed point q, $d^H_\Omega(p_i, q)$ scales inversely with p_i's distance to the boundary. Therefore, we scale the penalty associated with p_i by ω_i, defined as the inverse of its distance to the boundary:

$$\omega_i \;=\; \left(\min_{j\in[m]} d^H_\Omega(L_j, p_i)\right)^{-1}.$$

Finally, we propose the following LP in the parameters $\mathbf{w}$, c, and ξ_i, $\mathbf{y}(i)$, for $i \in [n]$, to minimize the overall weighted penalty for a fixed Hilbert radius r:

$$\begin{aligned}
\text{Minimize:} \qquad & \Xi_r \;=\; \sum_{i\in[n]} \omega_i \xi_i \\
\text{Subject to:} \qquad & \mathbf{b}(r,i)^\top \mathbf{y}(i) \;\le\; c + \xi_i, \quad \forall i \in I^+ \\
& \mathbf{b}(r,i)^\top \mathbf{y}(i) \;\ge\; c - \xi_i, \quad \forall i \in I^- \\
& \mathbf{A}(r,i)^\top \mathbf{y}(i) \;=\; \mathbf{w}, \quad \forall i \in I \\
& \mathbf{y}(i) \;\ge\; \mathbf{0}, \quad \forall i \in I \\
& \mathbf{1}^\top \mathbf{w} \;=\; 1 \\
& \xi_i \;\ge\; 0, \quad \forall i \in I.
\end{aligned} \tag{14}$$

Let the optimum value derived from Opt. (14) be Ξ_r, and the corresponding separating hyperplane, $K_r = (\mathbf{w}, c)$. Recall that r attains values over $\Gamma_r = [r_1, r_2, \ldots, r_{M/\varepsilon}]$, where r_i's are any values satisfying $(i-1)\varepsilon \le r_i < i\varepsilon$, and $M = O(B + \log d)$. To find an appropriate classifier K^*, we take

$$K^* \;=\; K_s, \quad \text{where} \quad s \;=\; \arg\max_{i\in[M/\varepsilon]} \left(r_i - C\,\frac{\Xi_{r_i}}{n}\right).$$

Note that C is a user-defined constant, a weight associated with improper separation. A higher value would favor proper separation, while a lower value would favor a larger margin.

A slight drawback of this approach is that it no longer suffices to perform a binary search on the range Γ_r. Rather, we need to perform a grid search, i.e. solve Opt. (14) for every value of $r \in \Gamma_r$ in the worst case. This implies the overall running time for the soft-margin classifier is $O(\text{poly}(n, m, d, B, 1/\varepsilon))$.

4.2 A Nearest Neighbor-based Classifier

In this section, we consider an alternative approach to binary classification in the Hilbert geometry. Rather than using a separating hyperplane, we select two sites q^+ and q^-, and classify a point p according to which of these two sites it is closer to. In the Euclidean setting,

the bisector between two points is a hyperplane, and this is no different from SVM. However, in Hilbert, bisectors are generally not hyperplanes. (An analysis of the structure of Hilbert bisectors in the 2-dimensional case was presented in [17], where it was shown that they are piecewise conics.)

As before, we assume that Hilbert geometry is defined with respect to a polytope Ω in $\mathbb{R}^d$ defined by m bounding hyperplanes $\mathcal{L} = \{L_i\}_{i=1}^m$. Extending results of de la Harpe [12], Vernicos presented an elegant isometry from the polytopal Hilbert geometry (Ω, d_Ω^H) to a finite-dimensional normed vector space [42]. In particular, he presented an isometric mapping $f_\Omega : (\Omega, d_\Omega^H) \to (\mathbb{R}^{m-1}, ||\cdot||_{\Sigma_{2m}})$, where $||\cdot||_{\Sigma_{2m}}$ is a norm defined by a regular polytope Σ bounded by $2m$ facets in $\mathbb{R}^{m-1}$. In other words, the embedding space is $\mathbb{R}^{m-1}$ with Σ_{2m} as the unit ball.

Although one might be tempted to solve the SVM problem in the embedding space, the result would be far from ideal. Since f_Ω is nonlinear, a linear classifier $\widehat{K}$ in the embedding space would not translate to a simple classifier in the input space. Moreover, since f_Ω is not a bijection it is not clear how complex the intersection of $\widehat{K}$ with the image of Ω, $f_\Omega(\Omega)$, would be.

Instead, we propose a simple nearest neighbor-based classifier in the embedding space. Given Ω, let q_i, Q^+, and Q^- denote the images of p_i, P^+, and P^- under the embedding f_Ω. (q_i for all $i \in [n]$ can be computed in $O(m^2)$ time by solving m linear equations). We find two representative centers c^+, and c^-, for Q^+, and Q^- respectively such that the following is maximized:

$$\beta \;=\; \max_{c^+, c^- \in \mathbb{R}^{m-1}} \left(\min_{q^+ \in Q^+} \frac{\min_{q^- \in Q^-} \|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma}{\|\mathbf{q}^+ - \mathbf{c}^+\|_\Sigma}, \min_{q^- \in Q^-} \frac{\min_{q^+ \in Q^+} \|\mathbf{q}^- - \mathbf{q}^+\|_\Sigma}{\|\mathbf{q}^- - \mathbf{c}^-\|_\Sigma} \right). \quad (15)$$

Stated differently, β is the value such that any point q is at least β times away from its nearest neighbor in the opposite class, as from its representative. This implies that the points can be correctly classified using an approximate nearest-neighbor algorithm with approximation factor at most β.

This can be rewritten as an optimization problem over the variables $\mathbf{c}^+$, $\mathbf{c}^-$, and $1/\beta$

$$\begin{aligned} &\text{Minimize:} && \frac{1}{\beta} \\ &\text{Subject to:} && \|\mathbf{q}^+ - \mathbf{c}^+\|_\Sigma \;\le\; \frac{1}{\beta}\|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma, \quad \forall q^+ \in Q^+, \forall q^- \in Q^- \\ & && \|\mathbf{q}^- - \mathbf{c}^-\|_\Sigma \;\le\; \frac{1}{\beta}\|\mathbf{q}^- - \mathbf{q}^+\|_\Sigma, \quad \forall q^+ \in Q^+, \forall q^- \in Q^-. \end{aligned} \quad (16)$$

We note that $\|\mathbf{q}^+ - \mathbf{q}^-\|_\Sigma$ in the above optimization is a pre-determined constant. If we let $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{2m}\}$ denote the directions determining the norm polytope Σ, then the condition that $\|\mathbf{x}\|_\Sigma \le t$ can be expressed as a set of linear inequalities: $\mathbf{w}_i^\top \mathbf{x} \le t$, for $i \in [2m]$. Therefore, Opt. (16) is an LP in $O(m)$ variables, and $O(nm)$ inequalities.

References

1 Aditya Acharya, Auguste H. Gezalyan, Julian Vanecek, David M. Mount, and Sunil Arya. Support vector machines in the hilbert geometry. In *19th International Symposium on Algorithms and Data Structures, WADS 2025*, pages 3:1–3:20, 2025. doi:10.4230/LIPIcs.WADS.2025.3.

2 Antoine Allard and M. Ángeles Serrano. Navigable maps of structural brain networks across species. *PLOS Computational Biology*, 16:1–20, 2020. doi:10.1371/journal.pcbi.1007584.

3 Kristin P. Bennett and Colin Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explor. Newsl.*, 2, 2000. doi:10.1145/380995.380999.

4 Garrett Birkhoff. Extensions of Jentzsch's theorem. *Trans. Amer. Math. Society*, 85:219–227, 1957.

5 Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2016. doi:10.1007/978-0-387-45528-0.

6 Mathieu Blondel, André F. T. Martins, and Vlad Niculae. Learning with Fenchel-Young losses. *J. Mach. Learn. Research*, 21:1–69, 2020. URL: http://jmlr.org/papers/v21/19-021.html.

7 Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998. doi:10.1023/A:1009715923555.

8 Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4869–4880, 2019. doi:10.48550/arXiv.1910.12933.

9 H. Cho, B. DeMeo, J. Peng, and B. Berger. Large-margin classification in hyperbolic space. In *Proc. 22nd Internat. Conf. Artif. Intell. and Stat.*, pages 1832–1840. PMLR, 2019. doi:10.48550/arXiv.1806.00437.

10 C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20:273–297, 1995. doi:10.1007/BF00994018.

11 Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. doi:10.1017/CBO9780511801389.

12 P. de la Harpe. On Hilbert's metric for simplices. In G. A. Niblo and M. A. Roller, editors, *Geometric Group Theory*, pages 97–119. Cambridge University Press, 1993. doi:10.1017/CBO9780511661860.009.

13 D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Mach. Learn.*, 46:161–190, 2002. doi:10.1023/A:1012454411458.

14 B. Dhingra, C. Shallue, M. Norouzi, A. Dai, and G. Dahl. Embedding text in hyperbolic spaces. In *Proc. Twelfth Workshop Graph-Based Methods for Nat. Lang. Proc. (TextGraphs-12)*, pages 59–69, 2018. doi:10.48550/arXiv.1806.04313.

15 Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 5350–5360, 2018. doi:10.48550/arXiv.1805.09112.

16 Bernd Gärtner and Jirí Matousek. *Understanding and using linear programming*. Universitext. Springer, 2007. doi:10.1007/978-3-540-30717-4.

17 Auguste H. Gezalyan and David M. Mount. Voronoi diagrams in the hilbert metric. In *39th International Symposium on Computational Geometry, SoCG 2023, Dallas, Texas, USA, June 12-15, 2023*, volume 258 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.35.

18 B. Hashemi, K. Pasque, C. Teska, and R. Yoshida. Tropical attention: Neural algorithmic reasoning for combinatorial algorithms. In *Proc. 39th Annu. Conf. Neural Inf. Process. Systems*, 2025. doi:10.48550/arXiv.2505.17190.

19 T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36:1171–1220, 2008. doi:10.1214/009053607000000677.

20 S. Huang, N. Cai, P. P. Pacheco, S. Narrandes, Y. Wang, and W. Xu. Applications of support vector machine (SVM) learning in cancer genomics. *Cancer Genomics & Proteomics*, 15:41–51, 2018. doi:10.21873/cgp.20063.

21 M. Ito, Y. Seo, T. Yamazaki, and M. Yanagida. Geometric properties of positive definite matrices cone with respect to the Thompson metric. *Linear Algebra Appl.*, 435:2054–2064, 2011. doi:10.1016/j.laa.2011.03.063.

22 Thorsten Joachims. *Learning to classify text using support vector machines - methods, theory and algorithms*, volume 668 of *The Kluwer international series in engineering and computer science*. Kluwer, 2002. doi:10.1007/978-1-4615-0907-3.

23 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. doi:10.1007/BF02579150.

24 Jacek Karwowski and Frank Nielsen. Hilbert geometry of the symmetric positive-definite bicone: Application to the geometry of the extended gaussian family. *CoRR*, abs/2508.14369, 2025. doi:10.48550/arXiv.2508.14369.

25 L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk*, 244:1093–1096, 1979. URL: https://www.mathnet.ru/eng/dan42319.

26 Scott Krig. *Computer Vision Metrics - Textbook Edition*. Springer, 2016. doi:10.1007/978-3-319-33762-3.

27 Wu Lin, Valentin Duruisseaux, Melvin Leok, Frank Nielsen, Mohammad Emtiyaz Khan, and Mark Schmidt. Simplifying momentum-based positive-definite submanifold optimization with applications to deep learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202, pages 21026–21050. PMLR, 2023. doi:10.48550/arXiv.2302.09738.

28 B. C. Lins. *Asymptotic behavior and Denjoy-Wolff theorems for Hilbert metric nonexpansive maps*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2007.

29 J. A. De Loera and E. D. Kim. *Combinatorics and geometry of transportation polytopes: An update*, volume 625, pages 37–76. AMS, 2013. URL: http://www.ams.org/books/conm/625/12491.

30 G. Mishne, Z. Wan, Y. Wang, and S. Yang. The numerical stability of hyperbolic representation learning. In *Proc. 40th Internat. Conf. Mach. Learn.*, pages 24925–24949, 2023. URL: https://proceedings.mlr.press/v202/mishne23a.html.

31 M. Nickel and D. Kiela. Poincaré embeddings for learning hierarchical representations. *Adv. Neural Inf. Proc. Syst.*, 30:6341–6350, 2017. doi:10.48550/arXiv.1705.08039.

32 F. Nielsen and K. Sun. Clustering in Hilbert's projective geometry: The case studies of the probability simplex and the elliptope of correlation matrices. In Frank Nielsen, editor, *Geometric Structures of Information*, pages 297–331. Springer Internat. Pub., 2019. doi:10.1007/978-3-030-02520-5_11.

33 Frank Nielsen and Ke Sun. Non-linear embeddings in hilbert simplex geometry. In *Topological, Algebraic and Geometric Learning Workshops 2023, 28 July 2023, Honolulu, HI, USA*, volume 221 of *Proceedings of Machine Learning Research*, pages 254–266. PMLR, 2023. arXiv:2203.11434.

34 A. Papadopoulos and M. Troyanov. *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*. European Mathematical Society Publishing House, 2014. doi:10.4171/147.

35 W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao. Hyperbolic deep neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(12):10023–10044, 2021. doi:10.1109/TPAMI.2021.3136921.

36 D. Reeb, M. J. Kastoryano, and M. M. Wolf. Hilbert's projective metric in quantum information theory. *J. Math. Physics*, 52(8), 2011. doi:10.1063/1.3615729.

37 Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 4th edition, 2020. URL: http://aima.cs.berkeley.edu/.

38 Thibault Séjourné, Jean Feydy, François-Xavier Vialard, Alain Trouvé, and Gabriel Peyré. Sinkhorn divergences for unbalanced optimal transport. *CoRR*, abs/1910.12958, 2019. doi:10.48550/arXiv.1910.12958.

39 Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Information science and statistics. Springer, 2008. doi:10.1007/978-0-387-77242-4.

40 Richard Szeliski. *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science. Springer, 2022. doi:10.1007/978-3-030-34372-9.

41 V. N. Vapnik and A. Y. Chervonenkis. On a class of algorithms of learning pattern recognition. *Automation and Remote Control*, 25(6):937–945, 1964. (In Russian). URL: `http://mi.mathnet.ru/at11678`.

42 C. Vernicos. On the Hilbert geometry of convex polytopes. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 111–126. European Mathematical Society Publishing House, 2014. `doi:10.48550/arXiv.1406.0733`.

43 Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80, pages 5819–5827. PMLR, 2018. `doi:10.48550/arXiv1805.07091`.

On the Fragile Complexity of Geometric Algorithms

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Mayank Goswami
Department of Computer Science, Queens College CUNY, New York, NY, USA

John Iacono
Université libre de Bruxelles, Ixelles, Belgium

Indu Ramesh
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, USA

Abstract

Surprisingly, the question of bounding the maximum number of operations undergone by each individual element in an algorithm – known as the *fragile complexity* of the algorithm – has not received much attention. In a foundational paper, Afshani et al. (2019) developed the concept of fragility and explored classic problems such as sorting and selection from this perspective. Motivated by a suggestion for future research by Afshani et al., we initiate a study of fragile complexity in computational geometry. We obtain bounds on several time-honored questions in 2D such as computing the maxima, closest pair, convex hull, triangulation, and approximate Euclidean Minimum Spanning Tree (apx-EMST). Our algorithms for the maxima, convex hull, and triangulation problems are competitive with the classical algorithms in terms of worst-case runtime and guarantee polylogarithmic fragility. We present an $O(n \log^2 n)$ time algorithm that returns a 1.0125-apx-EMST and achieves $O(\log^2 n)$ fragility, thus matching the best known performance up to polylogarithmic factors.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fragile complexity, convex hull, maxima, closest pair, algorithmic complexity

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.2

Funding *Boris Aronov*: Partially supported by NSF Grant CCF-20-08551.
Mayank Goswami: Supported by National Science Foundation CCF-2503086. Part of this work was performed at the AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawai'i at Mānoa, USA, with the support of the National Science Foundation (USA) under Grant No. 2452276.
John Iacono: Research supported by the Fonds de la Recherche Scientifique – FNRS (Belgium). Part of this work was performed at the AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawai'i at Mānoa, USA, with the support of the National Science Foundation (USA) under Grant No. 2452276.
Indu Ramesh: Supported by NSF Grant CCF-20-08551.

1 Introduction

Comparison-based algorithms are a fundamental and well-researched topic in computer science, and comprise a good fraction of most introductory algorithms courses. The traditional measure of the complexity of an algorithm is the *total* number of comparisons performed. For an input of n elements, however, the traditional algorithms typically do not care how these comparisons are spread across the elements. For example, in the sorting algorithm

© Boris Aronov, Mayank Goswami, John Iacono, and Indu Ramesh;
licensed under Creative Commons License CC-BY 4.0

QUICKSORT, some pivots undergo $\Omega(n)$ many comparisons, which is higher than the average $O(\log n)$ comparisons per element for any optimal comparison-based sorting algorithm. Can one sort in $O(n \log n)$ comparisons where every element participates in $O(\log n)$ comparisons?

Questions of the above flavor were considered in the seminal work of Afshani et al. [1], who studied the fundamental problems of sorting, selection, and median via the lens of *fragile complexity*. A comparison-based algorithm $\mathcal{A}$ has fragile complexity $f(n)$ if, for every input $\mathcal{I}$ of n elements, any element participates in at most $f(n)$ comparisons during the execution of A on $\mathcal{I}$. The fragile complexity of a problem is the minimum fragile complexity of any comparison-based algorithm solving that problem. We will soon highlight (and later use) some results from this foundational paper.

There are several motivations for considering the fragile complexity of a problem. Sometimes a comparison with an element results in the element undergoing some destructive phenomenon. Consider athletes being ranked via matches (comparisons), or ranking consumable objects (smørrebrød, beer, etc.) by comparisons. Every comparison has a physical toll on the athlete and reduces the amount of the consumable object, and so it is reasonable to ask whether there are algorithms that spread the comparisons across the elements evenly. In addition to the above, there may be *privacy* advantages in not using an element too many times. Furthermore, a procedure that involves using every item roughly an equal number of times may be perceived to *have a lower bias* compared to a procedure where some elements are involved in many more operations than others.

The original paper [1] ends with a tantalizing open question: what about the fragile complexity of *geometric problems*? They state: "In particular, it would be interesting to study "geometric orthogonal problems" such as finding the maxima of a set of points, detecting intersections between vertical and horizontal line segments, kd-trees, axis-aligned point location and so on. All of these problems can be solved using algorithms that simply compare the coordinates of points." Apart from another work on the fragile complexity of adaptive algorithms [5], we are not aware of other research on fragile complexity.

In this work we address the fragile complexity of fundamental geometric problems that are standard in textbooks [10] for an introductory course on computational geometry: the problems we study are *maxima, closest pair, convex hull*, *triangulation*, and *approximate Euclidean Minimum Spanning Tree* (apx-EMST) problems. In most cases, existing algorithms for these problems have at least linear fragility. For example, the classical divide-and-conquer algorithm for closest pair compares a candidate closest-pair distance to a linear number of distances, forcing linear fragility. One could imagine needing to compute the closest pair of battery-powered sensors, where each operation involving a sensor, such as requesting its coordinates, uses up its battery, and no location information can be stored for later processing. In this setup, a fragility-sensitive algorithm would help control battery drainage for all sensors. The Euclidean MST has recently found numerous applications in clustering high-dimensional data [14], and one can imagine that finding algorithms for EMST with low fragility may offer privacy and fairness advantages.

A number of our conclusions are somewhat surprising: we argue, for example, that a triangulation of a point set, potentially containing linear-degree vertices, can be constructed with polylogarithmic fragility. We also argue that any closest-pair algorithm in an algebraic decision tree model has to operate on the pair of closest points at least $\Omega(\log n)$ times.

2 Overview and Results

We first define the models in which we will derive our upper and lower bounds.

Computation model. In this paper, every problem we consider takes as input an array or list of *items*, such as numbers, points, or lines, which have real-valued attributes such as coordinates. Unless otherwise stated, the algorithms use the *real RAM* model of computation (without floors or ceilings) where decisions are made based on the signs of bounded-degree polynomials in the input attributes. The lower bounds use *algebraic decision trees*; see [11] for an overview. For some problems, we use a more restricted model.

Fragile complexity model. Fix a specific algorithm solving a given problem, which stays in the algebraic decision tree/real RAM model. Consider the tree representing all possible computation and decision paths of the execution of the algorithm. The *fragility of an item* along an execution path is the number of operations it is involved in. The *worst-case fragility of an item* is the maximum number of operations it is involved in, along all possible executions paths. The *fragility of the algorithm* is the maximum fragility of any of the input items.

As we are also interested in the total work (or "cost" or "running time") of our algorithm, we additionally measure the total number of real RAM operations performed along a worst-case execution path. For a randomized algorithm, we also examine the *expected* cost and fragility. We summarize the prior work next, and then state our problems and results.

Prior Work. Prior work on fragile complexity has focused on comparison-based algorithms [1] and adaptive algorithms [5]. We utilize the following results from [1] in our work:

> MINIMUM: Given a list $X = \langle x_1, x_2, \ldots, x_n \rangle$ of n numbers, find the smallest number, only allowing comparisons of the form $x_i < x_j$; more precisely, return $\arg\min_i x_i$. If numbers repeat, return the index of *one* smallest.

▶ **Theorem 2.1** (Afshani et al. [1]). *There is an algorithm that solves* MINIMUM *with* $O(\log n)$ *fragile complexity and* $O(n)$ *runtime. Solving* MINIMUM *requires* $\Omega(\log n)$ *fragile complexity. The minimum element itself requires* $\Omega(\log n)$ *fragility.*

The upper bound in Theorem 2.1 comes from computing the minimum via tournament tree.

> SORTING: Given a list $X = \langle x_1, x_2, \ldots, x_n \rangle$ of n numbers, find a permutation σ of the indices $\langle 1, \ldots, n \rangle$ so that $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(n)}$.

▶ **Theorem 2.2** (Ajtai et al. [3]). *The AKS sorting network solves* SORTING *with* $O(\log n)$ *fragile complexity.* SORTING *has* $\Omega(\log n)$ *fragile complexity.*

As a warmup for fragility, here is a proof of the SORTING fragility lower bound: Sorting has a lower bound of $\Omega(n \log n)$ comparisons in the algebraic decision tree model. By the pigeonhole principle, the fragility of some element must be $\Omega(\log n)$.

Our results. We obtain results on the following geometric problems on sets of points through the lens of fragile complexity. To our knowledge, this has not been done before.

> STAIRCASE: The *staircase* of a list of points P is the x-ordered sequence of points in P where no other point has both a larger x- and a larger y-coordinate. Given a list of $P = \langle p_1, \ldots, p_n \rangle$ points, output the sequence of indices of the staircase of P.

> CLOSEST PAIR: Given a list $P = \langle p_1, p_2, \ldots, p_n \rangle$ of n points in the plane, find the closest pair among them. More precisely, return a pair (i, j) of indices so that $d(p_i, p_j) = \min_{i' \neq j'} \{d(p_{i'}, p_{j'})\}$.

Convex Hull: The *convex hull* is the smallest convex set enclosing a set of points. Given a list $P = \langle p_1, \ldots, p_n \rangle$ of points, produce the vertices of the convex hull of P in counterclockwise order.

Triangulation: A *triangulation* of a point set is a subdivision of its convex hull into triangles whose edges form a maximal set of non-crossing segments connecting pairs of points. Given a set $P = \langle p_1, \ldots, p_n \rangle$ of points, produce a triangulation of the point set.

Approximate Euclidean MST: The *Euclidean Minimum Spanning Tree* (EMST) of a point set P is a spanning tree of P whose edges have minimum total length among all spanning trees on P, where the length of an edge is the Euclidean (ℓ_2) distance between its vertices. A *c-approximate* EMST (apx-EMST) is a spanning tree of P whose total length is at most c times the length of the EMST. Given a set $P = \langle p_1, p_2, \ldots, p_n \rangle$ of n points in the plane, find an approximate Euclidean MST, for some $c \geq 1$.

Our results are given in the table below. Our bounds for staircase are tight; a logarithmic gap remains in the other problems.

Problem	Upper Bound	Lower Bound
Staircase	$O(\log n)$	$\Omega(\log n)$
Closest Pair	$O(\log^2 n)$	$\Omega(\log n)$
Convex Hull	$O(\log^2 n)$	$\Omega(\log n)$
Triangulation	$O(\log^2 n)$	$\Omega(\log n)$
Approximate Euclidean MST	$O(\log^2 n)$	$\Omega(\log n)$

Note that the last lower bound is for the exact Euclidean MST. Observe that, somewhat surprisingly, we can achieve polylogarithmic fragility for computing a triangulation potentially containing high-degree vertices.

3 Maxima and staircases

Let $P = \langle p_1, \ldots, p_n \rangle$ be a list of n points in the plane. A point $p = (x, y) \in P$ is *dominant* (or a *maximum*) if no other point of P has both a larger x- and a larger y-coordinate. We define *lexicographical order* on points in the plane as follows: $p < q$ if $x(p) < x(q)$, or $x(p) = x(q)$ and $y(p) < y(q)$. We consider Staircase and a related problem Maxima, the difference being in whether we insist on outputting the maxima in their lexicographic order:

Maxima: Given P as above identify all maxima, that is, compute the not necessarily ordered set of indices of the dominant elements of P.

We propose a divide-and-conquer algorithm that solves the potentially harder Staircase problem in $O(n \log n)$ time with fragility $O(\log n)$ and argue below that this is the best one can do even for Maxima.

Algorithm.

1. Sort the points in P in lexicographic order.
2. We now proceed recursively.
   a. Divide P at the median into P_{left} and P_{right}.
   b. Recursively find the staircase s_{left} of P_{left} and the staircase s_{right} of P_{right}.

c. Let p_{left} be the point of P_{left} corresponding to the first index of s_{left} and p_{right} the point of P_{right} corresponding to the first index of s_{right}. If $y(p_{\text{left}}) < y(p_{\text{right}})$, return s_{right}.

d. Otherwise, binary search for $y(p_{\text{right}})$ (the *key*) among the points corresponding to s_{left}. Return the section of s_{left} containing points with y-coordinate at least that of $y(p_{\text{right}})$, followed by s_{right}.

▶ **Lemma 3.1.** *The above algorithm for* Staircase *achieves $O(\log n)$ fragility with $O(n \log n)$ running time.*

Proof. Step 1, presorting, uses $O(\log n)$ fragility [3] and costs $O(n \log n)$ running time (this step is the bottleneck).

We argue that each point of the set P is involved in a logarithmic number of comparisons over the entire runtime of the recursive algorithm. Each point of P participates in a logarithmic number of subproblems. Step 2(a) makes no comparisons. Step 2(c) involves one comparison between two points of the current subproblem and therefore causes at most logarithmic fragility.

It remains to bound the fragility arising from the binary search in Step 2(d). Each binary search comparison is between a key and an element of the current subproblem; the latter participates in one comparison per binary search and therefore a logarithmic number of comparisons overall. The key ($y(p_{\text{right}})$) is compared a logarithmic number of times in a fixed binary search. The binary search is only triggered once for each possible p_{right} (when it is the topmost point of P_{right}, but not the topmost point in P). It will no longer be p_{right} at higher levels of recursion. Hence it only participates in a logarithmic number of comparisons in its capacity as a key, completing the proof of logarithmic fragility. ◀

In [17, Lemma 2.2] it was argued that a *comparison-model* decision tree for solving the Maxima problem must have at least $n!$ leaves and therefore requires $\Omega(n \log n)$ comparisons. In the *algebraic decision tree* model, Kirkpatrick and Seidel give a lower bound of $\Omega(n \log h)$ [15] for Maxima, where h is the number of maxima. By the pigeonhole principle, and because $h = \Theta(n)$ in the worst case, we obtain a lower bound of $\Omega(\log n)$ on fragility of Maxima.

4 Closest Pair

We describe a randomized algorithm with $O(\log^2 n)$ expected fragility to solve this problem. We also prove a lower bound on $\Omega(\log n)$ fragility of any algorithm solving Closest Pair in a more restricted model of computation we call the *CP-model*, which only allows comparisons of the form $x_i - x_j < d(p_k, p_l)$, $d(p_i, p_j) < d(p_k, p_l)$, between x-coordinates of the input points, and between y-coordinates.

Note that we can as well assume points are distinct, since identifying repeated points requires $\Theta(n \log n)$ work (in all models we consider) and therefore $\Theta(\log n)$ fragility in our model and can be run as a preprocessing step.

Closest Pair: Algorithm. We will follow the $O(n \log n)$-time closest-pair divide-and-conquer algorithm from [8], with adjustments for fragility. Assume we are given an array P of points $p_i = (x, y)$. The following are the steps our algorithm takes.

0. Pre-sort the points of P, once by x-coordinate and once by y-coordinate, storing them in arrays X and Y, respectively. Each point "knows" its position in these two arrays. That is, $X[i]$ not only stores the point of P with ith largest x-coordinate, but also its position in the Y array, and vice versa.

1. Divide the array X into two parts, X_L and X_R, of nearly equal size partitioned by the median x-coordinate x_m. Let Y_L and Y_R denote the corresponding array of points, sorted by y-coordinate – they can be computed from Y in linear time using the cross links mentioned above.
2. Recursively solve Closest Pair (starting at step 1) on (X_L, Y_L) and (X_R, Y_R), which return closest pairs (p_L, q_L) and (p_R, q_R), respectively. Let (p, q) be the pair that defines the smaller distance between these two pairs and let $\delta = d(p, q)$.
3. Consider the strip of width 2δ centered at the line $x = x_m$. We identify the points of P lying in this strip in the following fragility-sensitive manner (recall that we cannot store the value δ, but need to refer to p and q as necessary; we also need to watch the fragility of x_m). Specifically, we find the smallest index i_L in X so that $x_m - x_{i_L} < \delta$ (if it exists), by binary search. Similarly, we find the largest index i_R in X so that $x_{i_R} - x_m < \delta$. The portion $X[i_L \dots i_R]$ contains precisely the points of P in the desired strip.
4. We can extract the points of $X[i_L, \dots, i_R]$ into an array F ordered by y-coordinate by scanning through Y once and collecting points whose position in X lies in $[i_L, i_R]$.
5. As discussed in [8], any pair of points at distance less than δ from each other will appear at most seven places apart in F. Therefore, we now collect the following list of point pairs: a point $F[i]$ and a point $F[j]$ at most 7 positions beyond it, i.e., with $i < j \leq i + 7$. This produces no more than $7n$ pairs of points. We can now apply SampleMinimum from [1] to this list of pairs, to find the smallest distance defined by one them.
6. We return the pair with smaller of two distances: $\delta = d(p, q)$ and the closest pair from step 5.

▶ **Lemma 4.1.** *The algorithm above runs in $O(n \log n)$ with $O(\log^2 n)$ expected fragile complexity.*

Proof. Step 0 (presorting) costs $O(\log n)$ fragility using the AKS sorting network [3].

The algorithm then proceeds by divide and conquer with $O(\log n)$ levels. Step 1 does not require access to point coordinates. We will show through analysis of the further steps that step 2 requires $O(\log n)$ fragility. Step 3 (or computing i_L and i_R) involves binary search, which costs $O(\log n)$ fragility for points p and q and coordinate x_m; the other points potentially touched in binary search cost $O(1)$ fragility per point. Step 4 (creating the array F) involves constant fragility per point. For step 5, each point in F participates in $O(1)$ pair comparisons (by the geometric analysis in [8]). Utilizing the minimum algorithm from [1] in step 5 allows us to calculate the overall minimum with expected $O(\log n)$ fragile complexity – each value comparison in minimum computation translates to a comparison of two distances between points in P. Generating the solution pair in step 6 involves $O(1)$ fragility.

Since we can guarantee $O(\log n)$ fragility at each level of recursion, and it is possible that the same pair of points is the closest at all $\log n$ levels of recursion, the overall fragile complexity is $O(\log^2 n)$. ◀

Note that the fragility analysis of the above algorithm is tight in the worst case. Some points will participate in $\Omega(\log^2 n)$ operations. Indeed, consider a point set whose size n is a power of 2, in which the leftmost two points (p_1, p_2) define the smallest distance d. Furthermore, make sure that that the x-separation between any other pair of points is larger than d. Then (p_1, p_2) will be the solution of the leftmost subproblem on all levels of recursion and will be involved in a logarithmic number of binary search comparisons at each level.

Closest Pair: Lower Bound. We give a fragility-preserving reduction from MINIMUM to CLOSEST PAIR in the *CP-model*. This will show not only that CLOSEST PAIR has $\Omega(\log n)$ fragile complexity, but also the closest pair of points experiences $\Omega(\log n)$ fragility.

We start with an instance X of MINIMUM, which is a list consisting on n distinct numbers x_i, satisfying two additional constraints. We assume that $X \subset [0, 0.1]$ and that, for any two values, x_i and x_j (with $i \neq j$), either $x_i > 2x_j$ or $x_j > 2x_i$. Map each x_i to two points $p_{2i} = (0, 100^i)$ and $p_{2i+1} = (x_i, 100^i)$. This transforms an instance X of *Minimum*, with n numbers, to an instance $P = P(X) = \langle p_0, \ldots, p_{2n-1} \rangle$ of CLOSEST PAIR, with $2n$ points.

Consider any algorithm A that solves CLOSEST PAIR using only the comparisons mentioned. Note that we never need to actually compute the distance from a point to itself or compare the distance defined a pair of points to itself: the answer is known without performing the calculation. Consider an instance X to MINIMUM subject to the above constraints, transform it into an instance $P = P(X)$ of CLOSEST PAIR according to the mapping above, and run A on P. We start with an immediate fact.

▶ **Fact 4.2.** *Let d be the distance between two distinct points of P. Then, there exist integers $s \geq t > 0$ such that:*

(a) $d \in [100^s - 100^t, 100^s - 100^t + 0.1]$.

(b) *If $s = t$, $d \in [0, 0.1]$,*

(c) *if $s > t$, $d \geq 99$, and*

(d) *two such distances d, d' defined by distinct pairs of exponents (s, t) and (s', t'), with $s > t$ and $s' > t'$, can be compared based solely on the values of the exponents s, t, s', t' (the distances need not be computed nor compared).*

Two points in P are *twins* if they come from the same number in X, i.e., they are points p_{2r} and p_{2r+1} for some r, $0 \leq r \leq n - 1$.

▷ Claim 4.3. The closest pair of points in $P = P(X)$ must be the closest pair of twins. In symbols, it is the pair (p_{2i}, p_{2i+1}) with $i = \arg\min_{0 \leq i < n} x_i$.

Proof. By Fact 4.2(b)&(c), the smallest distance must correspond to some pair (p_{2r}, p_{2r+1}) of twins. But $d(p_{2r}, p_{2r+1}) = x_r$, for all r, and thus the smallest distance in P corresponds to the minimum in X, completing the proof. ◁

▶ **Lemma 4.4.** *Every comparison of the form $x_i - x_j < d(p_k, p_l)$ among the points of $P = P(X)$ can be resolved by examining the indices i, j, k, l, and by making at most two comparisons of two values from X.*

Proof. If $k = l$, p_k and p_l are the same point, and the comparison can be resolved by one comparison, $x_i < x_j$. If $i = j$, the comparison evaluates to TRUE. If both $k = l$ and $i = j$, the comparison evaluates to FALSE. So we assume $k \neq l$ and $i \neq j$ below.

Case 1: p_k and p_l are twins. In other words, $i = 2r$ and $j = 2r + 1$, or vice versa. Then $d(p_k, p_l) = x_r$, and the comparison is equivalent to $x_i < x_j + x_r$.

Recall that we assumed a "gap": for any two numbers $x_a, x_b \in X$, if $x_b < x_a$, then $x_a > 2x_b$; and if $x_a < x_b$, then $x_b > 2x_a$; otherwise $x_a = x_b$.

Suppose $x_i > x_j$ and therefore $x_i > 2x_j$, is TRUE. With one comparison of x_i and x_j exhausted, we compare x_i and x_r to check if $x_i > x_r$ (and therefore $x_i > 2x_r$). If this comparison returns TRUE, adding the two inequalities gives us $x_i > x_j + x_r$, implying that the original inequality of $x_i < x_j + x_r$ is FALSE, so we are done in two comparisons. Otherwise, $x_r > 2x_i$ or $x_r = x_i$. In both cases, $x_i < x_j + x_r$ is TRUE; two comparisons are enough.

If $x_i > x_j$ is FALSE, we check if $x_j > 2x_i$ by comparing x_i and x_j. Since $x_j + r > 2x_i + r$, the original comparison evaluates to TRUE. Otherwise $x_i = x_j$, which we don't need to check – the original comparison evaluates to TRUE.

Case 2: p_k and p_l are not twins. Suppose p_k and p_l are not twins. Then, by (c) of Fact 4.2, $d \geq 99$. Since $x_i - x_j \in [0, 0.1]$, the comparison evaluates to FALSE. ◀

▶ **Lemma 4.5.** *Every distance comparison of the form $d(p_i, p_j) < d(p_k, p_l)$ among the points of $P = P(X)$ can be resolved by just examining the values of the indices i, j, k, l or is the result of making at most one comparison of two of the original numbers from X.*

Proof. Fact 4.2 implies that if p_i, p_j, p_k, p_l have three or four distinct y-coordinates among them, the comparison can be resolved by just examining the y-coordinates, which are determined by the indices i, j, k, l. So it remains to handle cases where only two distinct y-coordinates appear among these, not necessarily distinct points. Note that we do not need to address the case of one y-coordinate, as that means at most two distinct points are involved in the distance comparison, which can be resolved without examining x-coordinates. Additionally, if there are no twins and exactly two distinct y-coordinates, the two pairs of points in the comparison are the same. Below are the remaining cases.

Case 1: Two Pairs of Twins. Suppose p_i and p_j are twins ($i = 2r$ and $j = 2r + 1$, or vice versa) and p_k and p_l are twins ($k = 2s$ and $l = 2s + 1$, or vice versa). Then $d(p_i, p_j) = x_r$, $d(p_k, p_l) = x_s$, so the comparison is equivalent to $x_r < x_s$, as claimed. Now suppose p_i and p_k are twins and p_j and p_l are twins. If p_i and p_j are even and p_k and p_l are odd, so that $i = 2r$ and $k = 2r + 1$ and $j = 2s$ and $l = 2s + 1$, then $d(p_k, p_l)$ is greater than $d(p_i, p_j)$ as it involves a positive x-coordinate difference, while $d(p_i, p_j)$ does not, so $d(p_i, p_j) < d(p_k, p_l)$ is TRUE. Now suppose p_i is even and p_j is odd and p_k is odd and p_l is even. Then $i = 2r$, $j = 2s + 1$, $k = 2r + 1$, $l = 2s$ and the distance comparison is equivalent to $x_s < x_r$. Similar reasoning applies if $(i, j), (k, l)$ pairs are interchanged.

Case 2: One Pair of Twins. If there is one pair of twins and two y-coordinates, then the comparison involves the same point, p, on both sides, and two other points. Suppose $i = 2s$, $k = j = 2r + 1$, and $l = 2s + 1$: without loss of generality, p_i and p_l are twins, and $p = p_j = p_k$ with an odd index. Then the y-coordinate differences cancel out on both sides, and the comparison of distances reduces to $x_r < |x_s - x_r|$, which can be resolved by comparing x_r to x_s, since by our assumption, either $2x_r < x_s$ or $2x_s < x_r$. If $j = k$ is even, we have $i = 2s$, $k = j = 2r$, and $l = 2s + 1$, which resolves to $0 < x_s$, which is TRUE. ◀

We call a comparison of the two forms above on $P(X)$ *useful* if it reduces to at most two comparisons of two elements in X; refer to the lemmas above. Any other comparison does not depend on the x_i values; we refer to these as *useless comparisons*.

An execution of A for Closest Pair on $P(X)$ can be used to solve Minimum on X by removing useless comparisons (i.e., resolving them without accessing X) and turning useful distance comparisons into comparisons between numbers of X. Note that the transformation preserves fragility as it does not require access to X – we do not explicitly construct $P(X)$, but rather simulate each operation of A on its points by translating them to operations on point indices and comparisons of elements in X.

▶ **Theorem 4.6.** *Closest Pair is at least as hard as Minimum and therefore has $\Omega(\log n)$ fragile complexity in the CP-model. Additionally, the closest pair itself has a fragile complexity of $\Omega(\log n)$.*

Note that we could also obtain an $\Omega(\log n)$ lower bound on fragility in the algebraic decision tree model (if we reduce, for example, from Element Uniqueness to Closest Pair in one dimension, to Closest Pair) [20], since this requires $\Omega(n \log n)$ operations and thus $\Omega(\log n)$ fragility, by pigeonhole principle. However, such reasoning does *not* yield any information about the fragility of the closest pair itself, which is why we provided the alternate reduction above.

5 Convex hull

Computing the *convex hull* of a set P of points is one of the time-honored problems in computational geometry with many algorithms described in the literature. However, on even casual examination, the best-known algorithms do not guarantee reasonable fragility. We observe that many (if not all) of the existing convex hull algorithms have $\Omega(n)$ fragile complexity. For example, Graham's scan [12] requires linear fragility due to one point possibly being involved in an orientation test a linear number of times. Jarvis' march, also known as the gift-wrapping algorithm [13], requires linear fragility due to a linear scan it performs. The ultimate planar convex hull algorithm of Kirkpatrick and Seidel [16] requires linear fragility due to how their marriage-before-conquest bridge-finding routine is implemented.

Note that if P is a multiset, we can remove duplicates in $O(n \log n)$ time with $O(\log n)$ fragility, and this does not affect the overall upper bounds. This preprocessing step allows us to assume that the points in P are distinct. To further simplify the description, we assume that the set is in *general position*: no three points are collinear and no two lie on the same vertical line. We will adapt the algorithm of Preparata and Hong for computing the convex hull [19]. The *upper hull* $\mathrm{UCH}(P)$ of a point set S is the set of points p on the boundary of its convex hull with the property that an open upward vertical ray emanating from p is disjoint from the convex hull. The upper convex hull of a finite point set can be conveniently represented as the sequence of its vertices, naturally ordered by the x-coordinate. The lower hull is defined symmetrically. We compute the convex hull by computing the upper and the lower hull of a point set separately and combining them. The last step takes $O(1)$ time and does not affect fragility, if the hulls are represented by suitable linked lists of indices.

We now describe the upper hull computation. We proceed by divide-and-conquer, so the interesting step is merging two upper hulls. Consider two upper hulls $\mathrm{UCH}(Q)$ and $\mathrm{UCH}(R)$, comprising points $q_n, q_{n-1}, \ldots, q_1$ and $r_1, \ldots, r_n$, respectively, in this x-order. A point r_j is a *tangent point* (of R) for q_i if all points of R lie on or under the line $\ell_{q_i r_j}$ supporting the segment $q_i r_j$. If q_i is also a tangent point (of Q) for r_j, $q_i r_j$ is the *upper tangent* of Q and R.

> Bridge-Finding: Find the upper tangent to the two upper hulls $\mathrm{UCH}(Q)$ and $\mathrm{UCH}(R)$, given as arrays of points $Q = \langle q_n, \ldots, q_1 \rangle$ and $R = \langle r_1, \ldots, r_n \rangle$, where Q is left of R and the points are sorted by the x-coordinate.

Algorithm for Bridge-Finding. We follow the exponential-then-binary search approach for the merge portion of mergesort used in [1], adapted to the geometry of convex hulls. In the exponential portion, we start with the rightmost point q_1 of $\mathrm{UCH}(Q)$ and the leftmost point r_1 of $\mathrm{UCH}(R)$, $q_1 r_1$ being a tentative bridge between the two upper hulls. The goal is to identify a tangent point r_l for q_1. In the exponential portion of the search, we fix q_1 and then, starting with $r_1 = r_{2^0}$, we examine points r_{2^k}, for $k = 0, 1, 2, \ldots$, until we find k for which $q_1 r_{2^k} r_{2^k+1}$ is oriented clockwise. If $q_1 r_{2^k-1} r_{2^k}$ is also oriented clockwise, $r_l = r_{2^k}$ – we found the tangent for q_1. Otherwise, the desired tangent point r_l for q_1 lies between $r_{2^{k-1}}$ and r_{2^k} and we use a binary search to identify it.

Let r_j be the current candidate in the binary search for the tangent point for q_1. Geometrically, if the segment $q_i r_j$ goes through UCH(R), r_{j+1} (and also all points beyond it) lie below $l_{q_i r_j}$, and r_{j-1} lies above $l_{q_i r_j}$, then r_j overshot r_l (i.e., $j > l$). Otherwise, $r_1, \ldots, r_{j-1}$ lie below $l_{q_i r_j}$, r_{j+1} lies above $l_{q_i r_j}$, and the binary search continues in the opposite direction.

When we finally identify the tangent r_l for q_1, if $q_1 r_l$ is not an upper tangent (meaning r_l is a tangent point for q_1, but q_1 is not the tangent for r_l), we can discard q_1 and the points $r_1, \ldots, r_{l-1}$.

We continue the exponential and binary search on $Q' = r_l, \ldots, r_n$ and $R' = q_2, \ldots, q_n$, swapping left and right and thus the roles of Q and R, starting with a tentative bridge of $r_l q_2$ (the details of undershooting, overshooting, and discarding are symmetric). The algorithm terminates when $q_i r_j$ is the upper tangent.

▶ **Lemma 5.1.** *No point during the execution of the algorithm* Bridge-Finding *participates as a non-key in more than $O(\log n)$ exponential and/or binary searches.*

Proof. Suppose exponential search from q_i begins at index r, and binary search proceeds in the vertices beginning at index $r + 2^l$ and ending at index $r + 2^{l+1}$. Suppose the tangent to q_i is r'. Any point p touched during exponential search is either discarded or becomes part of binary search. If p is left of r', it is discarded after the binary search and not touched again. Otherwise, if p is right of r', we will show that the distance between p and r, the starting point of exponential search, is halved each time.

In particular, we prove that $p - r \geq 2(p - r')$. Since $r' \geq r + 2^l$, $2r' \geq 2r + 2^{l+1}$. Subtracting r from both sides, we get $2r' - r \geq r + 2^{l+1}$. Since $p \leq r + 2^{l+1}$, $p \leq 2r' - r$. Subtracting the right side, we get $p - 2r' + r \leq 0$. Adding p to both sides, and rearanging, we obtain $2(p - r') \leq p - r$. Since the distance to r' is at least halved at least in each step, this cannot happen more than a logarithmic number of times; thus p can only be involved in a logarithmic number of binary searches. ◀

▶ **Lemma 5.2.** *The algorithm* Bridge-Finding *as described above achieves $O(\log n)$ fragile complexity with $O(n)$ runtime.*

Proof. Any point that acts as a key for exponential and/or binary search is touched $O(\log n)$ times and discarded. It could also be touched a constant number of times in a single exponential and/or binary search. By Lemma 5.1, no point is involved in more than $O(\log n)$ exponential and/or binary searches. The sidedness tests used to test the tangency and/or upper tangency condition can be implemented with $O(1)$ fragility using orientation tests on triples of points. Each time an element is touched, the two elements next to it are touched, but this only increases the fragility by a factor of at most three.

As far as the runtime is concerned, the combination of exponential and binary searches are designed to cover (up to constant factors) the same region of the upper hull as the classical algorithm of Preparata and Hong [19] and therefore cannot be slower than their linear-scan procedure that takes total linear time. ◀

We thank the anonymous reviewer for suggesting that the bridge-finding algorithm implemented by Overmars and van Leeuwen [18] also gives $O(\log n)$ fragility.

Algorithm for Convex Hulls. Presort the points of P by x-coordinate. Divide P in half via the median x-coordinate. Recursively compute the upper hull of both halves. Use Bridge-Finding to compute the upper tangent; this algorithm also discards points that do not appear on UCH(P). Compute the lower hull separately using a symmetric process. Combine the upper and lower hulls to construct the entire convex hull.

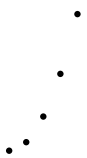


Figure 1 An adversarial input that forces $\Omega(\log^2 n)$ fragility for our convex hull algorithm: place points at (i, i^2) for $i = 0, \ldots, n-1$. Pictured for $n = 5$.

▶ **Theorem 5.3.** *The convex hull of a set of n points in the plane can be computed in $O(n \log n)$ time with $O(\log^2 n)$ fragile complexity using the algorithms above.*

Proof. First, we address correctness. The algorithm is correct for the same reason as the classic divide and conquer algorithm, and the exponential and binary searches used in Bridge-Finding make upward progress for the same reason as this algorithm, which finds the bridge in linear time [19]. Now, we address fragility. Presorting uses $O(\log n)$ fragility [3]. By Lemma 5.2, Bridge-Finding requires $O(\log n)$ fragile complexity, and there are $O(\log n)$ rounds of the algorithm. The final combine step uses zero fragility, as it consists of listing point indices. ◀

▶ **Lemma 5.4.** *The algorithm above has a lower bound of $\Omega(\log^2 n)$ fragility. In other words, the above fragility analysis is tight.*

Proof. In Figure 1, we give an example of an input that forces high fragility. The leftmost point $(0, 0)$ is part of the bridge at every level of recursion. At each level, it is touched a logarithmic number times in exponential search, and there are $\Theta(\log n)$ recursion levels. ◀

In the algebraic decision tree model, Ben-Or [4] gave a lower bound of $\Omega(n \log n)$ for the problem of determining if n points in the plane are in convex position, that easily reduces to Convex Hull, which implies a lower bound of $\Omega(\log n)$ fragility due to the pigeonhole principle.

Extension to triangulating a point set. In this section, we show that minor modifications to the algorithms above allow us to produce a triangulation of a point set with polylogarithmic fragility. This result is surprising because we would expect a linear lower bound due to fact that the resulting triangulation may have arbitrarily-high-degree vertices.

Algorithm. Sort the point set by x-coordinate and partition by the median x-coordinate. Now, divide and conquer. Recursively compute triangulations of each half.

We now describe how to complete a triangulation by triangulating the region delimited by the bridge and the two upper hulls (which are part of the triangulation of each half). Run Bridge-Finding from the previous section with the following modifications. After each round of exponential/binary search is complete, draw a chord between the key element q_i (symmetrically, r_i) and the tangent element of the other side r_j (symmetrically, q_j). Output the vertices that are discarded, as chords from these to q_i (or r_i) comprise a *fan* triangulation.

Continue exponential/binary search recursively as described in the previous section until $q_i r_j$ is the upper tangent, stopping to fill in chords as described.

Consider the path formed by traversing P in x-order. The final result of the above algorithm is a list of chords that, triangulates the region between this path and UCH(P). Run a symmetric process for the lower hull (which does not contain vertices of the upper hull except for the endpoints) to complete the triangulation of the convex hull of P.

▶ **Theorem 5.5.** *The algorithm above triangulates a point set with fragility $O(\log^2 n)$ in time $O(n \log n)$.*

Proof. The proof from the previous section applies, and the only extra operation(s) are outputting the fans before discarding the vertices. This is done by listing the generated chords using vertex indices – these operations do not affect the fragility. Since this modification only adds a constant amount of work for every discarded vertex and does not affect fragility, the runtime and fragility bounds follow. ◀

Note that Triangulation has a fragility lower bound of $\Omega(\log n)$ due to the pigeonhole principle and the application of the lower bound, [20, Theorem 5.2].

6 Approximate Euclidean minimum spanning tree

We describe an algorithm for computing a 1.0125-approximate EMST of a set of n points in the plane with $O(\log^2 n)$ fragility. Our algorithm will do $O(n \log^2 n)$ total work, where the operations performed are (a) comparisons between x- or y-coordinates of two points p_i and p_j, (b) projecting a point $p \in P$ on a fixed direction $d \in S^1$, obtaining $\pi_d(p)$, and (c) performing comparisons between x- or y-coordinates of such projected points. However, if $\pi_d(p)$ is used for any future comparisons, we increment the fragility of p accordingly. Note that since the Euclidean MST contains the closest pair of points as an edge, a lower bound of $\Omega(\log n)$ on the fragility follows.

We briefly recall some foundational papers on computing the EMST, and show why they are not immediately fragility-friendly. Our algorithm will not be an adaptation of a single known algorithm, but instead we will use ideas from two algorithms.

First, a classic result by Shamos and Hoey [21] states that the EMST is a subgraph of the Delaunay triangulation. Given that the Voronoi diagram can be computed in $O(n \log n)$ time, one obtains the $O(n)$-sized Delaunay triangulation in the same time bound. We can then extract the EMST from the Delaunay triangulation by any MST algorithm, e.g., running Kruskal's algorithm, in $O(n \log n)$ time. However, the Voronoi diagram and the Delaunay triangulation may contain high-degree vertices,[1] and furthermore such high-degree vertices will experience heavy fragility during Kruskal's algorithm, even if we used a fragility-sensitive sorting algorithm.

A similar high-degree issue arises even if one considers the relative neighborhood graph [10], or the Yao graph [22]. If an approximation of the EMST is allowed, one can replace the Yao graph by the Theta graph, but the same high-degree problem persists, as can be seen by an instance of one point p opposing $n-1$ points on a chain;[2] p has degree $\Omega(n)$ in both the Yao and Theta graphs.

Algorithm overview. Our first building block will be the algorithm by Agarwal et al. [2], that is based on a reduction from EMST to a collection of *bichromatic closest-pair problems* (BCP). An instance of BCP is provided with a set R of red and B of blue points, and the goal is to find $(r, b) \in R \times B$ with minimum $d(r, b)$. Agarwal et al. show that the EMST is contained in the graph formed by connecting all pairs (r, b) that are solutions to the obtained BCP instances. We will first show that this collection of BCP problems can be obtained in a

[1] It is not obvious whether these can be constructed in a manner avoiding linear fragility, such as a fan structure allowing for a compact representation as exploited by our triangulation algorithm.

[2] The chain can be made slightly convex or concave, to get a point set in general position.

fragility-sensitive manner, and that no point is involved in too many such subproblems. Another property of the BCP instances obtained after this reduction is that in each instance (R, B) of BCP, there is a double-wedge so that R is in one wedge and B in the antipodal one, and the angle of the wedge is small (at most $\pi/10$).

The obvious approach would be to next solve the BCP instances in a fragility-sensitive manner, perhaps like we did for closest pair in Section 4. While there are several $O(n \log n)$ algorithms known for BCP, we again hit a block when trying to adapt them for fragility. As discussed earlier, we do not know how to build the Voronoi diagram in a fragility-sensitive way due to the potential for high-degree vertices. Hence we cannot adapt algorithms that start with a Voronoi diagram and find the closest red for a blue or vice-versa (in fact, there would be the additional task of making the associated point location or search data structures fragility-friendly). Given the instances produced by the reduction are separated by a line (a consequence of the double-wedge property mentioned above), one can consider an $O(n \log n)$ randomized algorithm [6] that (assume R is left of the separating line and B to the right) picks a random $r \in R$, finds the closest blue b to r, and discards all points in R that are outside the left envelope of the disks of radii $|rb|$ at points in B. We do not know how to adapt this algorithm either. Finally, we remark that there are reductions from BCP to Closest Pair [9], but these only work in high ($\omega(\log n)$) dimensions.

Instead of solving the BCP instances exactly, we will find an approximate bichromatic closest pair in each instance. We will then show that the graph formed by edges between such approximate bichromatic closest pairs (a) has degree $O(\log n)$ for every vertex, (b) is connected, and (c) contains an approximate EMST. We can then extract an approximate EMST from this graph by running Kruskal's algorithm after a fragility-sensitive sorting of the edge lengths. To find an approximate bichromatic closest pair, our idea is inspired by the Theta graph: we take the direction d along the double-wedge (there will be $O(1)$ many choices for d), project points in R and B on d, and return the red with the rightmost projection and the blue with the leftmost projection, for which we solve Minimum in one dimension, in a fragility-sensitive manner using [1].

Step 1: Forming the bichromatic closest pair instances. The reduction from EMST to BCP instances in Agarwal et al. [2] proceeds as follows. Let $\alpha_0 = \frac{1}{2}\arcsin(\sqrt{5}-1)/2 \approx 19.08°$ be the largest angle satisfying $\tan 2\alpha < \cos 2\alpha$, and let $\alpha = \pi/10 < \alpha_0$. A unit vector $d \in S^1$ with $||d||_2 = 1$ will be called a direction. Define the cone $\text{Cone}(d, \alpha) \subset \mathbb{R}^2$ to be the region that is the union of all rays going through the origin that form an angle at most α with d. An instance (R, B) of BCP is called α-separated in direction d if there exists a point z and a direction d such that $R \subset z + \text{Cone}(-d, \alpha)$ and $B \subset z + \text{Cone}(d, \alpha)$. All instances of BCP returned after the reduction will be α-separated in some direction d, where there will be at most $\pi/\alpha = 10$ choices for d.

Define, for $0 \le j \le 19$, the directions $e_j = (\cos j\alpha, \sin j\alpha)$. Note that for every j, the pair (e_j, e_{j+1}) forms a basis of $\mathbb{R}^2$, and these 20 basis pairs further define 20 directions d_j that are the bisector directions for each basis pair. The algorithm will successively consider the basis (e_j, e_{j+1}) and for each j, return a collection of α-separated BCP instances in direction d_j, with the pairs (R, B) lying in the double wedge formed by (e_j, e_{j+1}) and $-(e_j, e_{j+1})$, with R in one wedge of angle α and B in the antipodal wedge. In the following, we assume j to be fixed, and express a point $p \in \mathbb{R}^2$ as $p = p_1 e_j + p_2 e_{j+1}$ in the basis (e_j, e_{j+1}). The algorithm takes as input two sets R and B (both initialized to P originally), the dimensionality k (initialized to two), and in calls itself recursively at most three times, where each recursive call either reduces the dimensionality, or halves the size of $|R \cup B|$ in the recursion.

The algorithm in [2] is as follows, and we present it with the needed twists for fragility. As a preprocessing step, we sort (using $O(\log(|R \cup B|))$ fragility) the points in R by both coordinates, points in B by both coordinates, and $R \cup B$ in both coordinates. Every point in R and B will know its position in the other lists where it resides, similar to our algorithm for closest pair in Section 4. If $k = 0$ then output (R, B) as an α-separated pair. Otherwise if $k \geq 1$, we extract the median x_k of the kth coordinate of the points in $R \cup B$ (this is readily available as $R \cup B$ is already sorted in both coordinates). We then partition R into R_ℓ and R_r where R_ℓ contains points in R whose kth coordinate is at most x_k, and R_r contains those whose kth coordinate is larger than x_k. We similarly partition B into B_ℓ and B_r. Note that this partitioning can be performed without comparisons as in Section 4, since the lists are already sorted. Finally, we make at most three recursive calls: (1) if $R_\ell \neq \emptyset$ and $B_r \neq \emptyset$, we recurse with dimensionality $k - 1$, R_ℓ and B_r, (2) if $R_\ell \neq \emptyset$ and $B_\ell \neq \emptyset$, we recurse with dimensionality k, R_ℓ and B_ℓ, and (3) if $R_r \neq \emptyset$ and $B_r \neq \emptyset$, we recurse with dimensionality k, R_r and B_r. Note that in (1) the dimensionality is reduced by one, and in (2) and (3) the size of the union is halved.

▶ **Lemma 6.1.** *There exists an algorithm that, given $P \subset \mathbb{R}^2$ with $|P| = n$ returns a collection of $O(n \log n)$ α-separated BCP instances $(R, B) \in P \times P$, such that*

(a) *each instance (R, B) is α-separated in one of the directions d_j for $0 \leq j \leq 19$,*
(b) *every point in P occurs in $O(\log n)$ many instances of BCP,*
(c) *every point in P has $O(\log^2 n)$ fragility,*
(d) *each edge in the EMST on P is the solution to one of the returned BCP instances.*

Proof. The bound on the number of instances follows from a standard recursion, since either the dimensionality is reduced or the problem size is halved. This also proves assertion (b) in the lemma above, since each point participates in at most two recursive calls of type (1) and $O(\log n)$ many recursive calls of type (2) or (3).

The algorithm outputs a pair only when dimensionality 0 is reached, which means that the algorithm has succeeded in finding a double wedge such that R_ℓ and B_r are in the two opposing wedges of angle α. This double wedge is in the direction d_j, and this implies assertion (a). To prove assertion (c), note that the sorting performed as preprocessing in each call induces $O(\log n)$ fragility, and since each point is in $O(\log n)$ recursive calls, we get a fragility bound of $O(\log^2 n)$. Note that the sizes of the recursive problems decreases, but this only reduces the total fragility by a constant factor, hence we ignore this effect. Finally, assertion (d) follows from [2, Lemma 4]. ◀

Step 2: Approximate bichromatic closest pair and apx-EMST. Consider an instance (R, B) of BCP that is α-separated in direction d_j, for some $0 \leq j \leq 19$. Without loss of generality, assume that $d_j = (1, 0)$ and that R lies in the left wedge of angle α and B lies in the right wedge of angle α. We project R and B on direction d_j (in this case the x-axis), and run a fragility-sensitive Maximum algorithm for $\pi(R)$ and a fragility-sensitive Minimum algorithm for $\pi(B)$, as in [1]. Let the point in R achieving the maximum be $\tilde{r}$ and the minimum in B be $\tilde{b}$. For every returned BCP instance, we add the corresponding edge $(\tilde{r}, \tilde{b})$ to form a graph G on P. This pair may be different from the bichromatic closest pair for this instance, which we will denote by (r^*, b^*).

Because each point is involved in at most $O(\log n)$ BCP instances (assertion (b) in Lemma 6.1), and it experiences $O(\log n)$ fragility during the projection and min/max algorithms in each instance, the total fragility of any point is bounded by $O(\log^2 n)$. More importantly, the degree of any vertex in G is again $O(\log n)$, since at worst it could be the

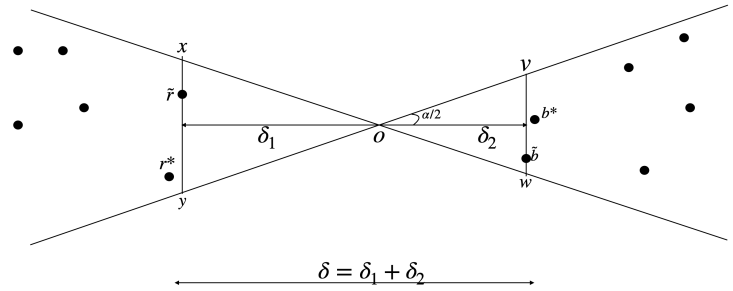


Figure 2 The situation in Lemma 6.2.

endpoint of an added edge of the type $(\tilde{r}, \tilde{b})$ in all $O(\log n)$ instances it belongs to. Thus G has $O(n \log n)$ many edges, and sorting all edges in G by their lengths takes $O(n \log^2 n)$ time. Let us consider the fragile complexity of this step. Each vertex in G (that is, each point in P) experiences a fragility of at most $O(\log^2 n)$, since each of its $O(\log n)$ many incident edges experiences $O(\log n)$ fragility during sorting.

The above considerations imply that if the following lemma is true, then running Kruskal's algorithm on G after sorting the costs gives us an approximate EMST.

▶ **Lemma 6.2.** *Consider the graph G formed by adding edges between the pair of vertices $(\tilde{r}, \tilde{b})$ as described above, and let (r^*, b^*) be the bichromatic closest pair in the corresponding instance. Then*

(a) $d(\tilde{r}, r^*) < d(r^*, b^*)$ *and* $d(\tilde{b}, b^*) < d(r^*, b^*)$,
(b) *G is connected,*
(c) $\frac{d(\tilde{r},\tilde{b})}{d(r^*,b^*)} \leq \sec(\alpha/2) = \sec(\pi/20) < 1.0125$, *and*
(d) *the edges of G contain a 1.0125-approximate EMST.*

Proof. Clearly, (d) follows from (b) and (c), since every edge of the EMST is of the form (r^*, b^*) for some instance, we replace it with an edge of approximate length, and G is connected. In the remainder, we prove (a)–(c).

For this, we will first make the simplifying assumption that no two points in P share a coordinate in any of the bases (e_j, e_{j+1}), and that no two points have the same projection in any of the directions d_j, for any $0 \leq j \leq 19$. It is not hard to remove this assumption by randomly perturbing the basis vectors above, as the argument does not hinge on them being uniformly distributed; it just relies on all angles being at most α_0 and on the union of the double wedges covering $\mathbb{R}^2$.

Refer to Figure 2. Since $\tilde{r}$ is the red point with maximum projection on the direction d (without loss of generality considered here to be horizontal), r^* must lie to the left of the line segment xy. Similarly b^* must lie to the right of the line segment vw. Let the horizontal distance between the projections of $\tilde{r}$ and $\tilde{b}$ be δ, and we break this up into $\delta_1 + \delta_2$, with δ_1 being the portion of it in the left wedge and δ_2 being the portion of it in the right wedge. Also note that since the direction d is the bisector of the wedge, the interior angles between the direction of the x-axis and either of the lines defining the wedge, are $\alpha/2$.

We first prove (c). Since $\tilde{r}$ is constrained to lie on the line xy and $\tilde{b}$ on vw, the maximum distance between them is obtained when $\tilde{r} = x$ and $\tilde{b} = w$. This distance is $\delta_1 \sec(\alpha/2) + \delta_2 \sec(\alpha/2) = \delta \sec(\alpha/2)$. Thus $d(\tilde{r}, \tilde{b}) \leq \delta \sec(\alpha/2)$. On the other hand, $d(r^*, b^*)$ is at least δ, as otherwise either r^* is to the right of xy or b^* is to the left of vw, contradicting that $\tilde{r}$ and $\tilde{b}$ were the points with the extremal projections. Therefore, $\frac{d(\tilde{r},\tilde{b})}{d(r^*,b^*)} \leq \frac{\delta \sec(\alpha/2)}{\delta} = \sec(\alpha/2)$. Recall that $\alpha = \pi/10$, plus some minor perturbation to make the above assumption hold, if necessary, which proves (c).

Next we prove (a). Assume the wedge is centered at the origin, so that r^* has x-coordinate equal to $-(\delta_1 + \epsilon_1)$ and b^* has x-coordinate $\delta_2 + \epsilon_2$, for some $\epsilon_1 > 0$, $\epsilon_2 > 0$. We claim that $\epsilon_1 \leq \delta(\sec(\alpha/2) - 1)$. To see this, assume the contrary. Then the distance $d(r^*, b^*)$ is at least $\delta_1 + \delta_2 + \epsilon_1 + \epsilon_2$, which is larger than $\delta + \epsilon_1$, and therefore by the assumption is larger than $\delta \sec(\alpha/2)$. However, we just showed that $d(\tilde{r}, \tilde{b}) \leq \delta \sec(\alpha/2)$, and this contradicts that (r^*, b^*) is the exact solution to the BCP instance.

Next, let us assume that the y-coordinate of r^* is t^* and that of $\tilde{r}$ is $\tilde{t}$. Thus $r^* = (-\delta_1 - \epsilon_1, t^*)$ and $\tilde{r} = (-\delta_1, \tilde{t})$. Note that $|\tilde{t}| \leq \delta_1 \tan(\alpha/2)$, since that is the y-coordinate of x and $\tilde{r}$ must lie on xy. Similarly, we get that $|t^*| \leq (\delta_1 + \epsilon_1) \tan(\alpha/2)$, since r^* is constrained to lie in the left wedge.

We obtain that the distance $d(\tilde{r}, r^*) \leq \epsilon_1 + |\tilde{t}| + |t^*|$. Using the bounds on ϵ_1, $|\tilde{t}|$ and $|t^*|$, this is at least

$$\begin{aligned} d(\tilde{r}, r^*) &\leq \epsilon_1 + |\tilde{t}| + |t^*| \\ &\leq \delta(\sec(\alpha/2) - 1) + \delta_1 \tan(\alpha/2) + (\delta_1 + \epsilon_1) \tan(\alpha/2) \\ &< \delta(\sec(\alpha/2) - 1) + \delta \tan(\alpha/2) + (\delta + \delta) \tan(\alpha/2) \\ &= \delta(3 \tan(\alpha/2) + \sec(\alpha/2) - 1) \\ &< \delta/2 < d(r^*, b^*). \end{aligned}$$

In the above we have used that $\delta_1 \leq \delta$, $\epsilon_1 < \delta$, and that for $\alpha = \pi/10$, $3 \tan(\alpha/2) + \sec(\alpha/2) - 1 \approx 0.48 < 1/2$. Finally, we use that $d(r^*, b^*) > \delta$. A similar argument can be used to prove that $d(\tilde{b}, b^*) < d(r^*, b^*)$ and this finishes the proof of (a).

Finally, we prove (b). Our proof is inspired by an argument by Callahan [7]. Consider any pair of points $p, q \in P$. We want to show that there is a path between them in G. Let T denote the Euclidean MST. We will show that G contains such a path by using induction on the length of the longest edge in the path connecting p to q in T. Let e be the longest edge in the path in T connecting p to q, and because e is an MST edge, it must be of the type $e = (r^*, b^*)$ for some bichromatic closest pair in one of the instances. Assume that r^* is encountered first on this path when walking from p to q. This edge $e = (r^*, b^*)$ is replaced by the approximate edge $(\tilde{r}, \tilde{b})$ in G. We claim that there is a path connecting p to $\tilde{r}$ in T that only uses edges cheaper than e – one goes from p to r^* in T using edges cheaper than e, and by (a), $d(r^*, \tilde{r}) < d(r^*, b^*) = |e|$. Applying the induction hypothesis we get that there is such a path from p to $\tilde{r}$ in G too. Similarly there is path in G from q to $\tilde{b}$ such that all edges have length at most $|e|$. Since the edge $(\tilde{r}, \tilde{b})$ is in G, this implies that there is path from p to q in G, and hence G is connected. This finishes the proof of correctness and the approximation guarantee of our algorithm, summarized as follows. ◀

▶ **Theorem 6.3.** *Given a set P of n points in $\mathbb{R}^2$, there exists an $O(n \log^2 n)$ time algorithm with $O(\log^2 n)$ fragility that returns a tree whose total length is at most* $\sec(\pi/20)(< 1.0125)$ *times that of the EMST.*

Open Problems. Several open problems emerged as a result of this research and at the suggestion of the anonymous referees. Is it possible to solve Convex Hull in $O(\log n \log h)$ fragility, where h is the number of hull vertices? Does there exist an algorithm to compute a Voronoi Diagram or Delaunay Triangulation in sublinear fragility? Is it possible to obtain a $\Omega(\log^2 n)$ lower bound on any problem addressed in this paper?

References

1 Peyman Afshani, Rolf Fagerberg, David Hammer, Riko Jacob, Irina Kostitsyna, Ulrich Meyer, Manuel Penschuck, and Nodari Sitchinava. Fragile Complexity of Comparison-Based Algorithms. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2019.2.

2 Pankaj K Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 203–210, 1990. doi:10.1145/98524.98567.

3 Miklós Ajtai, János Komlós, and Endre Szemerédi. An O(n log n) sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9. ACM, 1983. doi:10.1145/800061.808726.

4 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.

5 Prosenjit Bose, Pilar Cano, Rolf Fagerberg, John Iacono, Riko Jacob, and Stefan Langerman. Fragile complexity of adaptive algorithms. In *International Conference on Algorithms and Complexity*, pages 144–157. Springer, 2021. doi:10.1007/978-3-030-75242-2_10.

6 Prosenjit Bose, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Jan Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geometry*, 37(3):209–227, 2007. doi:10.1016/J.COMGEO.2006.03.006.

7 Paul B Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. The Johns Hopkins University, 1995.

8 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

9 Karthik C.S. and Pasin Manurangsi. On closest pair in Euclidean metric: Monochromatic is as hard as bichromatic. *Combinatorica*, 40(4):539–573, 2020. doi:10.1007/S00493-019-4113-1.

10 Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

11 Jeff Erickson. Lecture notes for cs 497: Concrete models of computation, spring 2003.

12 Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Lett.*, 1:132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

13 Ray A Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973. doi:10.1016/0020-0190(73)90020-3.

14 Rajesh Jayaram, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Massively parallel algorithms for high-dimensional euclidean minimum spanning tree. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3960–3996. SIAM, 2024. doi:10.1137/1.9781611977912.139.

15 David G Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of the First Annual Symposium on Computational Geometry*, pages 89–96, 1985. doi:10.1145/323233.323246.

16 David G Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. doi:10.1137/0215021.

17 H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975. doi:10.1145/321906.321910.

18 Mark H Overmars and Jan Van Leeuwen. Maintenance of configurations in the plane. *Journal of computer and System Sciences*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

19 Franco P. Preparata and Se June Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977. doi:10.1145/359423.359430.

20 Franco P Preparata and Michael I Shamos. *Computational Geometry: An Introduction*. Springer Science & Business Media, 2012.

21 Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975. doi:10.1109/SFCS.1975.8.

22 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. doi:10.1137/0211059.

Reachability with Restricted Reactions in Inhibitory Chemical Reaction Networks

Divya Bajaj ✉
University of Texas Rio Grande Valley, TX, USA

Bin Fu ✉
University of Texas Rio Grande Valley, TX, USA

Ryan Knobel ✉
University of Texas Rio Grande Valley, TX, USA

Austin Luchsinger ✉
University of Texas Rio Grande Valley, TX, USA

Aiden Massie ✉
University of Texas Rio Grande Valley, TX, USA

Pablo Santos ✉
University of Texas Rio Grande Valley, TX, USA

Ramiro Santos ✉
University of Texas Rio Grande Valley, TX, USA

Robert Schweller ✉
University of Texas Rio Grande Valley, TX, USA

Evan Tomai ✉
University of Texas Rio Grande Valley, TX, USA

Tim Wylie ✉
University of Texas Rio Grande Valley, TX, USA

Abstract

Chemical Reaction Networks (CRNs) are a well-established model of distributed computing characterized by quantities of molecular species that can transform or change through applications of reactions. A fundamental problem in CRNs is the reachability problem, which asks if an initial configuration of species can transition to a target configuration through an applicable sequence of reactions. It is well-known that the reachability problem in *general* CRNs was recently proven to be Ackermann-complete. However, if the CRN's reactions are restricted in both power, such as only deleting species (deletion-only rules) or consuming and producing an equal number of species (volume-preserving rules), *and* size (unimolecular or bimolecular rules), then reachability falls below Ackermann-completeness, and is even solvable in polynomial time for deletion-only systems.

In this paper, we investigate reachability under this set of restricted unimolecular and bimolecular reactions, but in the Priority-Inhibitory CRN and Inhibitory CRN models. These models extend a traditional CRN by allowing some reactions to be inhibited from firing in a configuration if certain species are present; the exact inhibition behavior varies between the models. We first show that reachability with Priority iCRNs mostly remains in P for deletion-only systems, but becomes NP-complete for one case. We then show that reachability with deletion-only reactions for iCRNs is mostly NP-complete, and PSPACE-complete even for $(1, 1)$-size (general) reactions. We also provide FPT algorithms for solving most of the reachability problems for the iCRN model. Finally, we show reachability for CRNs with states is already NP-hard for the simplest deletion-only systems, and is PSPACE-complete even for (general) $(1, 1)$-size reactions.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Complexity classes

Keywords and phrases Chemical Reaction Networks, Vector Addition Systems, Petri-nets, Reachability, Inhibitors, Void Reactions

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.3

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2604.17510

Funding This research was supported in part by National Science Foundation Grant CCF-2329918.

1 Introduction

Chemical Reaction Networks (CRNs) [4, 5] are an abstract model of molecular computation in which molecular *species* can evolve through applications of *reactions*. Although initially motivated by modeling natural chemical interactions, CRNs have been recently explored for

© Divya Bajaj, Bin Fu, Ryan Knobel, Austin Luchsinger, Aiden Massie, Pablo Santos, Ramiro Santos, Robert Schweller, Evan Tomai, and Tim Wylie;
licensed under Creative Commons License CC-BY 4.0

developing artificial chemical systems [23], computations of semilinear functions [8], digital logic [7, 16, 17], and neural networks [16], along with successful implementations in DNA Strand Displacement (DSD) systems [28].

CRNs are computationally equivalent to two classical models of infinite-state systems: Petri-nets [25] and Vector Addition Systems (VAS) [18, 19]. This equivalence connects molecular computation with decades of complexity-theoretic results concerning reachability [9, 14]. In all three models, the *reachability problem* asks whether a target configuration $\overrightarrow{C_t}$ can be reached from a given initial configuration $\overrightarrow{C_s}$ via a sequence of rule applications. Although certain characteristics of reachability in general CRNs (and equivalently, Petri-nets and VAS) were well-known for decades, such as EXPSPACE-hardness [21] and decidability [22], only recently has the problem been proven to be Ackermann-complete [10, 20].

A crucial limitation of the classical models is the absence of *zero-checking*. To address this, a natural and well-studied extension across all three models is to allow *inhibition*, which allows certain reactions to be disabled by the presence of designated species. Petri-nets with inhibitor arcs [15, 26], Priority Vector Addition Systems [13], and Inhibitory Chemical Reaction Networks [6] all explore this idea. Because inhibition enables zero-checking, these inhibitory extensions gain substantial power, achieving Turing Universality [6, 24]. However, due to the Turing-universal power of zero-checking, the reachability problem becomes undecidable in general for such systems with inhibition. For this reason, and to better understand the boundaries between decidable/tractable and undecidable/intractable, we limit our study to restricted classes of CRNs with inhibition that use only deletion-only or volume-preserving rules.

For CRNs, *deletion-only* (void) reactions never create any new species in the system (e.g., $\lambda_1 + \lambda_2 \to \emptyset$, or $\lambda_1 + \lambda_2 \to \lambda_1$), and *volume-preserving* reactions contain an equal number of reactants and products, ensuring that the system's volume never changes (e.g., $\lambda_1 + \lambda_2 \to \lambda_3 + \lambda_4$). These restrictions are well-motivated: unlike general CRN systems, which can exhibit complex and error-prone behaviors that are challenging to implement [29], deletion-only and volume-preserving systems are potentially simpler to realize and analyze [1, 12]. Moreover, in the absence of inhibition, such restrictions significantly reduce computational power: deletion-only systems cannot compute even simple functions [2], reachability with bimolecular void reactions is solvable in polynomial time [1, 12], and reachability with $(1, 1)$-size reactions is solvable in polynomial time [1]. These observations raise a central question: to what extent do inhibitory extensions increase the computational power of CRNs when restricted to these small-sized restricted reactions?

Under these structural restrictions, we consider two types of inhibitory CRN systems. The general *Inhibitory* CRN (iCRN) [6] model simply allows reaction execution to be inhibited by the presence of some subset of species in the configuration. The *Priority-Inhibitory* CRN (P-iCRN) model establishes a priority for each reaction, in which a reaction with priority k is inhibited from firing unless the first k species are absent from the system. Here, the order of species is used to determine the inhibitors for each reaction. In this paper, we study the reachability problem under deletion-only and volume-preserving reactions for both of these models.

Our Results

For the restricted reaction types studied in this paper, we use the notation (r, p) to denote reactions with r reactants and p products (e.g., (2,0) is a bimolecular void reaction and (1,1) is a volume-preserving unimolecular reaction). For reference, in a traditional CRN model,

Table 1 Reachability with Inhibition for restricted rule types. Void systems that are NP-complete in regular CRNs, $(k, k-i)$ where $k \geq 3$ and $i \geq 2$, are not listed since they are still NP-hard with inhibitions. Cells highlighted in blue link to interesting open directions (in Section 5) that extend the results in the table.

Reachability under Inhibition							
Rule Size	**CRN**	**Ref.**	**P-iCRN**	**Ref.**	**iCRN**		**Ref.**
Deletion-Only Systems							
$(1,0)$	P	-	P	Obs. 7	P		Obs. 23
$(2,0)$	P	[1]	P	Thm. 15	NPC	FPT	Thm. 24, 30
$(2,1)$	P	[12]	P	Thm. 17	NPC	FPT	Thm. 31, 33
$(k,k-1)$	P	[12]	P	Thm. 17	NPC	FPT	Thm. 32, 33
$(2,0),(2,1)$	P	[12]	NPC	Thm. 20	NPC		Thm. 34
General Systems							
$(1,1)$	P, NL-hard	[1]	Open		PSPACE-c		Thm. 35
$(2,2)$	PSPACE-c	[11]	PSPACE-c	Obs. 22	PSPACE-c		Obs. 37
General	Ack-c	[10, 20]	Decidable	[13, 26]	Undecidable		[6, 26]

reachability is in P for all of our considered void reactions [1, 12], NL-hard for $(1,1)$-size reactions [1], PSPACE-complete for $(2,2)$-size reactions [11], and Ackermann-complete for general systems [10, 20].

As for organization, we first provide formal descriptions of all CRN models studied in this paper and some additional preliminaries in Section 2. We then study reachability under the Priority iCRN model in Section 3. Here, we prove that reachability with our considered void reactions is mostly in P, but becomes NP-complete for $(2,0)+(2,1)$-size reactions. Section 4 looks at reachability for iCRNs. We show that reachability becomes NP-complete for all void reactions except for size $(1,0)$, and PSPACE-complete for $(1,1)$ and $(2,2)$-size reactions. We additionally provide FPT algorithms for solving reachability with void reactions of sizes $(2,0),(2,1)$, and $(k,k-1)$. The complete scope of our reachability results is shown in Table 1. Finally, in Section 5, we conclude with a discussion on some open problems for resolving complexity gaps of Table 1 and future potential work concerning CRNs using states as an unconventional yet powerful form of inhibition.

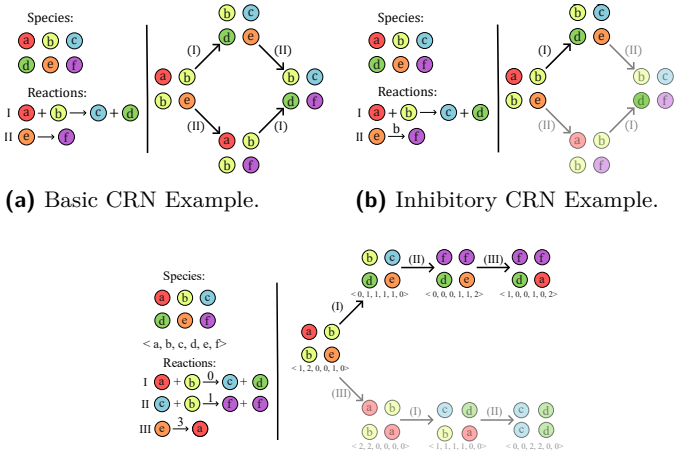
2 Preliminaries

We define the three chemical reaction network models considered in this paper: the basic CRN model (Section 2.1), the Inhibitory CRN model (Section 2.1.1), and the Priority iCRN model (Section 2.2.1).

2.1 Chemical Reaction Networks

A *chemical reaction network* (CRN) $\mathcal{C} = (\Lambda, \Gamma)$ is defined by a finite set of species Λ, and a finite set of reactions Γ where each reaction is a pair $(\vec{R}, \vec{P}) \in \mathbb{N}^{\Lambda} \times \mathbb{N}^{\Lambda}$, sometimes written $\vec{R} \longrightarrow \vec{P}$, that denotes the *reactant* species consumed by the reaction and the *product* species generated by the reaction. For instance, given $\Lambda = \{a, b, c\}$, the reaction $((2,0,0),(0,1,1))$ represents $2a \longrightarrow b + c$; 2 a species are removed, and 1 new b and c species are created. See Figure 1a for an example.

A *configuration* $\vec{C} \in \mathbb{N}^{|\Lambda|}$ of a CRN assigns integer counts to every species $\lambda \in \Lambda$, and we use notation $\vec{C}[\lambda]$ to denote that count. For a species $\lambda \in \Lambda$, we denote the configuration consisting of a single copy of λ and no other species as $\vec{\lambda}$. It is often useful to reference the



(a) Basic CRN Example. **(b)** Inhibitory CRN Example.

(c) Priority iCRN Example.

Figure 1 Example systems of the CRN models studied in this paper. Each arrow represents an application of a reaction. Transparent arrows represent invalid reaction applications. (a) Basic CRN. Observe how, despite two different reaction sequences being applicable in the initial configuration, both lead to the same terminal configuration. (b) Inhibitory CRN. Because Reaction 2 is inhibited by the species b, it cannot be applied in this system. (c) Priority iCRN. Reaction 3 is inhibited by species a and b, since both species are in the system, it's not an applicable reaction.

set of species whose counts are not zero in a given configuration. In such cases, the notation $\{\overrightarrow{C}\}$ is used. Formally, $\{\overrightarrow{C}\} = \{\lambda \in \Lambda \mid \overrightarrow{C}[\lambda] > 0\}$, and when convenient and clear from the context, we further use $\{\overrightarrow{C}\}$ to denote the configuration (vector) representation in which each element has a single copy. Finally, let $|\overrightarrow{C}| = \sum_{\lambda \in \Lambda} C[\lambda]$ denote the total number of copies of all species in a configuration, sometimes referred to as the *volume* of $\overrightarrow{C}$.

A reaction $(\overrightarrow{R}, \overrightarrow{P})$ is said to be *applicable* in configuration $\overrightarrow{C}$ if $\overrightarrow{R} \leq \overrightarrow{C}$; in other words, a reaction is applicable if $\overrightarrow{C}$ has at least as many copies of each species as $\overrightarrow{R}$. If the reaction $(\overrightarrow{R}, \overrightarrow{P})$ is applicable, it results in configuration $\overrightarrow{C'} = \overrightarrow{C} - \overrightarrow{R} + \overrightarrow{P}$ if it occurs, and we write $\overrightarrow{C} \rightarrow_{crn}^{(\Lambda,\Gamma)} \overrightarrow{C'}$, or simply $\overrightarrow{C} \rightarrow \overrightarrow{C'}$ when the model and CRN are clear from context.

A *void* reaction is any rule that does not create any new copies of any species type, and can only delete or preserve existing species copies. Thus, a void reaction either has no products or has products that are a subset of its reactants (in which case these products are termed *catalysts*).

▶ **Definition 1** (Discrete Chemical Reaction Network). *A discrete chemical reaction network (CRN) is an ordered pair* (Λ, Γ) *where* Λ *is an ordered alphabet of species, and* Γ *is a set of rules over* Λ.

▶ **Definition 2** (Basic CRN Dynamics). *For a CRN* (Λ, Γ) *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$*, we say that* $\overrightarrow{A} \rightarrow_{crn}^{(\Lambda,\Gamma)} \overrightarrow{B}$ *if there exists a rule* $(\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$*, and* $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}$.

If there exists a finite sequence of configurations such that $\overrightarrow{C} \rightarrow \overrightarrow{C}_1 \rightarrow \ldots \rightarrow \overrightarrow{C}_n \rightarrow \overrightarrow{D}$, then we say that $\overrightarrow{D}$ is *reachable* from $\overrightarrow{C}$ and we write $\overrightarrow{C} \rightsquigarrow \overrightarrow{D}$. A configuration is said to be *terminal* if no reactions are applicable. We also define an *initial configuration* for a CRN as its starting configuration. A *CRN System* T is then defined as a pair of a CRN model and its initial configuration.

The following sections define extensions of the basic CRN model by way of defining modified dynamics. For each model, assume the concepts of reachability and terminality are derived from the dynamics in the same manner as for the basic CRN model.

2.1.1 Inhibitory CRNs

A reaction γ is said to be *inhibited* by a species λ when the reaction γ may only be applied if λ is absent in the system. We define an inhibitor mapping $\mathcal{I} : \Gamma \to \mathbb{P}(\Lambda)$ that maps a reaction to a subset of species that inhibit the reaction. An *Inhibitory CRN* $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ as defined by [6] is then a basic CRN along with the mapping $\mathcal{I}$. See Figure 1b for an example.

▶ **Definition 3** (Inhibitory Dynamics). *For a Inhibitory CRN* $((\Lambda, \Gamma), \mathcal{I})$ *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$*, we say that* $\overrightarrow{A} \to^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{IC}}} \overrightarrow{B}$ *if there exists a rule* $\gamma = (\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$*,* $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}$*, and* $A[\lambda] = 0, \forall \lambda \in \mathcal{I}(\gamma)$*.*

2.2 Vector Addition Systems

A d-dimensional *Vector Addition System* (VAS) $\mathcal{V} = (\overrightarrow{C_0}, \mathcal{T})$ is defined by [19] as an initial configuration vector $\overrightarrow{C_0} \in \mathbb{Z}^d_{\geq 0}$, and a finite set of transitions $\mathcal{T} \subseteq \mathbb{Z}^d$.

▶ **Definition 4** (VAS Dynamics). *For a VAS* $(\overrightarrow{C_s}, \mathcal{T})$ *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$*, we say* $\overrightarrow{A} \to_{\mathcal{V}} \overrightarrow{B}$ *if there exists a transition* $\overrightarrow{y} \in \mathcal{T}$ *such that* $\overrightarrow{A} + \overrightarrow{y} = \overrightarrow{B}$*.*

The concept of priority on transitions in a Vector Addition System is defined in [13]. Each transition in a d-dimensional *Priority Vector Addition System* (PVAS) is mapped to an integer in $[0, d]$. For a system in configuration $\overrightarrow{A}$, a transition $\overrightarrow{y}$ with priority $p \leq d$ is applicable iff $\forall i \leq p$, $\overrightarrow{A}[i] = 0$ and $\overrightarrow{A} + \overrightarrow{y} \in \mathbb{Z}^d_{\geq 0}$.

The correspondence between Vector Addition Systems and Chemical Reaction Networks is direct, especially in the absence of catalysts. A transition vector $\overrightarrow{y}$ in a VAS can be written as a reaction $(\overrightarrow{R}, \overrightarrow{P})$ in CRN where

$$(\overrightarrow{R}[\lambda_i], \overrightarrow{P}[\lambda_i]) = \begin{cases} (|\overrightarrow{y}[i]|, 0) & \text{if } \overrightarrow{y}[i] < 0 \\ (0, \overrightarrow{y}[i]) & \text{if } \overrightarrow{y}[i] > 0 \\ (0, 0) & \text{otherwise} \end{cases}$$

Since PVAS are essentially VAS with prioritized inhibition on transitions, it is natural to compare this model to iCRNs, which have unprioritized inhibition on reactions. While Vector Addition Systems are not capable of *catalytic* transitions, it is known that a catalytic reaction in a CRN can be replaced by two non-catalytic reactions that use a unique intermediate species. Therefore, Vector Addition Systems are equivalent to Chemical Reaction Networks in general. However, this equivalence may break down under restricted settings such as deletion-only or small-size rules. This observation motivates a reformulation of PVAS as Priority iCRN in order to compare these results to inhibitory CRNs without priority.

2.2.1 Priority iCRN

We define a *priority* mapping $\mathcal{P} : \Gamma \to [0, |\Lambda|]$ that maps a reaction to an integer in $[0, |\Lambda|]$. A *Priority iCRN* $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ is defined as a basic CRN along with mapping $\mathcal{P}$. Because the priority relies on the ordering of species, we make our set of species Λ in a Priority iCRN an ordered set. See Figure 1c for an example.

▶ **Definition 5** (Priority Inhibitory Dynamics). *For a Priority iCRN* $((\Lambda, \Gamma), \mathcal{P})$ *and configurations* $\overrightarrow{A}$ *and* $\overrightarrow{B}$*, we say that* $\overrightarrow{A} \to^{(\Lambda,\Gamma)}_{\mathcal{C}_{\mathcal{PI}}} \overrightarrow{B}$ *if there exists a rule* $\gamma = (\overrightarrow{R}, \overrightarrow{P}) \in \Gamma$ *such that* $\overrightarrow{R} \leq \overrightarrow{A}$*,* $\overrightarrow{A} - \overrightarrow{R} + \overrightarrow{P} = \overrightarrow{B}$*, and* $A[\lambda_i] = 0, \forall i \in [1, \mathcal{P}(\gamma)]$*.*

2.3 Reachability

▶ **Definition 6** (Reachability Problem). *Given any CRN System with a species set Λ and a reaction set Γ, an initial configuration $\overrightarrow{C_s}$ and a destination (target) configuration $\overrightarrow{C_t}$: is $\overrightarrow{C_t}$ reachable from $\overrightarrow{C_s}$ following the dynamics of the model.*

We also consider a special variant of reachability referred to as the *empty configuration reachability* problem. Here, the destination configuration is always the empty configuration $\overrightarrow{0}$, where all species have a count of zero. Thus, the problem can be re-interpreted as asking if there exists a sequence of reactions that can delete all present species in $\overrightarrow{C_s}$.

3 Priority Inhibitory CRNs

We start our analysis of reachability in inhibitory systems with Priority iCRNs. We first show that the problem does not leave P with size $(2,0)$ or $(2,1)$ reactions. However, when considering systems with both $(2,0)$ and $(2,1)$ size rules, we show that the problem becomes NP-complete, even with a maximum priority of 1.

Priority iCRNs with Only Void Rules

We state the following observation for completeness.

▶ **Observation 7.** *Given a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ with only void rules of size $(1,0)$, the reachability problem is solvable in $\mathcal{O}(|\Lambda| + |\Gamma|)$ time.*

Proof. Given a Priority iCRN, start configuration $\overrightarrow{C_s}$ and target configuration $\overrightarrow{C_t}$:

1. For any species λ_i, if $\overrightarrow{C_t}[\lambda_i] \neq 0$ then remove all reactions γ_j such that $\mathcal{P}(\gamma_j) \geq i$. Since we only consider void rules, such reactions will never be applicable in any configuration along any path from $\overrightarrow{C_s}$ through $\overrightarrow{C_t}$.
2. For $i = 1$ to $|\Lambda|$, if there exists a reaction of form $\lambda_i \to \phi$:
   a. then apply the reaction $\overrightarrow{C_s}[\lambda_i] - \overrightarrow{C_t}[\lambda_i]$ times.
   b. else, skip to step (3).
3. If the final configuration is reached, return "yes", otherwise return "no". ◀

Void Rules of Size $(2,0)$. We now show that the reachability problem in Priority iCRNs with $(2,0)$ reactions is also solvable in polynomial time. To do so, we start by showing that the general reachability problem for a given Priority iCRN with $(2,0)$ void rules can be reduced to the empty reachability problem in the given CRN in Lemmas 8, 9, and 10. We then reduce the empty configuration reachability problem in the Priority iCRN to the known maximum b-matching problem, which is solvable in polynomial time [3], in Lemma 12.

Given a $(2,0)$ void Priority iCRN system $((\Lambda, \Gamma), \mathcal{P})$ such that $k = \max_{\gamma \in \Gamma} \mathcal{P}(\gamma)$. We can partition the set of species Λ into set of inhibiting species $\{\lambda_1, \ldots, \lambda_k\}$ and set of non-inhibiting species $\{\lambda_{k+1}, \ldots, \lambda_{|\Lambda|}\}$.

▶ **Lemma 8.** *Given two configurations $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$ in a $(2,0)$ void Priority iCRN $\mathcal{C}_{\mathcal{PI}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$, if $\forall i \in [1,k]$, $\overrightarrow{C_t}[\lambda_i] = 0$.*

Proof. Given that $\forall i \leq k, \overrightarrow{C_t}[\lambda_i] = 0$, $\overrightarrow{C_s}[\lambda_i] - \overrightarrow{C_t}[\lambda_i] = \overrightarrow{C_s}[\lambda_i]$. Therefore, all inhibited reactions will be applicable at some point.

Since void rules only delete species, and do not add new counts for species, for all non-inhibiting species λ_j with $j > k$, if $\overrightarrow{C_t}[\lambda_j]$ is reachable from $\overrightarrow{C_s}[\lambda_j]$, then $\overrightarrow{0}[\lambda_j]$ is reachable from $\overrightarrow{C_s}[\lambda_j] - \overrightarrow{C_t}[\lambda_j]$. ◀

▶ **Lemma 9.** *Given two configurations $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$ in a $(2,0)$ void Priority iCRN $\mathcal{C}_{\mathcal{PI}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$, if $\exists i \leq k, \overrightarrow{C_t}[\lambda_i] \neq 0$.*

Proof. We modify the given Priority iCRN $\mathcal{C}_{\mathcal{PI}}$ system as follows. We remove all reactions $\gamma \in \Gamma$ such that $\mathcal{P}(\gamma) \geq i$, $\forall i \leq k$ if $\overrightarrow{C_t}[\lambda_i] \neq 0$. This is because such reactions will never be applicable as they will always be inhibited. Due to this modification, the species $\lambda_i, \ldots, \lambda_k$ no longer permanently act as inhibitors. The new reaction set Γ' contains the remaining inhibited reactions with priority greater than 0. We now compute k' over Γ' as $k' = \max_{\gamma \in \Gamma'} \mathcal{P}(\gamma)$. We partition the set Λ again into set of inhibiting species $\{\lambda_1, \ldots, \lambda_{k'}\}$ and set of non-inhibiting species $\{\lambda_{k'+1}, \ldots, \lambda_{|\Lambda|}\}$. From Lemma 8, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$ in a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma'), \mathcal{P})$. ◀

▶ **Lemma 10.** *Given two configurations $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$ in a $(2,0)$ void Priority iCRN $\mathcal{C}_{\mathcal{PI}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$.*

Proof. Follows from Lemmas 8 and 9. ◀

▶ **Definition 11** (*b*-matching)**.** *Given a graph $G = (V, E)$ and some edge capacity function $u : E \to \mathbb{N} \cup \{\infty\}$, a b-value function $b : V \to \mathbb{N}$, and a $\delta(v)$ function that returns the set of incident edges of vertex v, find a maximum assignment $f : E \to \mathbb{N}$ s.t. $f(e) \leq u(e)$ for all $e \in E$ and $\sum_{e \in \delta(v)} f(e) \leq b(v)$ for all $v \in V$. We call it a perfect b-matching if $\sum_{e \in \delta(v)} f(e) = b(v)$ for all $v \in V$.*

The runtime of the maximum b-matching problem is strictly polynomial and runs in $O(|V|^2 \log(|V|)(|E| + |V| \log(|V|)))$ [3]. We use this runtime to solve the reachability problem for Priority iCRNs with only $(2,0)$-size reactions in polynomial time.

▶ **Lemma 12.** *Given a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ with only void rules of size $(2,0)$, the empty reachability problem is decidable in $O(|\Lambda|^2 \log(|\Lambda|)(|\Gamma| + |\Lambda| \log(|\Lambda|)))$ time.*

Proof. To solve the empty-configuration reachability problem in $(2,0)$ void Priority iCRN, we reduce it to an instance of the maximum b-matching problem.

Given a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ with starting configuration $\overrightarrow{C_s}$, we construct a b-matching instance by building a graph $G = (V, E)$ as follows. Here we assume that the given Priority iCRN only includes reactions $\gamma : \lambda_i + \lambda_j \to \phi$ where $i, j > \mathcal{P}(\gamma)$; any self-inhibiting reactions that exist (i or j is less than the priority) are ignored. We turn every species $\lambda_i \in \Lambda$ into a vertex $v_i \in V$ and set the b-values for each of the vertices $b(v_i) = \overrightarrow{C_s}[\lambda_i]$. For each $\gamma : \lambda_i + \lambda_j \to \phi$ in Γ such that $i \neq j$, add an undirected edge $e = (v_i, v_j)$ in E. For any reaction of the form $\lambda_i + \lambda_i \to \phi$, add vertices v_i^1, v_i^2 to the set V and set the b-values $b(v_i^1) = b(v_i^2) = \left\lfloor \frac{b(v_i)}{2} \right\rfloor$. We also add undirected edges $(v_i, v_i^1), (v_i, v_i^2), (v_i^1, v_i^2)$ in E.

From the rules above, we have the graph G well defined. We now present the following claims regarding G.

▷ Claim 13. If $\overrightarrow{C_s}$ can reach the empty configuration, then there exists a perfect b-matching in the graph G.

This is straightforward for the case when applying a reaction of the form $\lambda_i + \lambda_j \to \phi$ such that $i \neq j$. The reaction application results in the removal of one copy of species λ_i and λ_j. This corresponds to an edge matching of (v_i, v_j) on the graph G according to the construction above. We reduce the values of $b(v_i)$ and $b(v_j)$ by one, and the pair (v_i, v_j) form a perfect matching. For the case when applying a reaction of form $\lambda_i + \lambda_i \to \phi$, two copies of λ_i are deleted from the configuration. This rule application corresponds to selecting edges (v_i, v_i^1)

and (v_i, v_i^2). We reduce the counts of $b(v_i^1)$ and $b(v_i^2)$ by one, and that of $b(v_i)$ by two. The remaining b-values of vertices v_i^1 and v_i^2 are reduced by selecting edge (v_i^1, v_i^2). And the pairs (v_i, v_i^1), (v_i, v_i^2), and (v_i^1, v_i^2) form a perfect matching.

▷ Claim 14. If there exists a perfect b-matching in the graph G, then $\overrightarrow{C_s}$ can reach the empty configuration.

If there exists a perfect b-matching for the graph G, there exists an assignment $f : E \to \mathbb{N}$ such that $\sum_{e \in \delta(v)} f(e) = b(v)$ for all $v \in V$. From the construction of the graph, all edges $(v_i, v_j) \in E$ such that $i \neq j$ correspond to a reaction $\lambda_i + \lambda_j \to \phi$ in Γ in the Priority iCRN. And, edges of the form $(v_i, v_i^1), (v_i, v_i^2) \in E$ correspond to a reaction $\lambda_i + \lambda_i \to \phi$ in Γ. We argue that, given any perfect b-matching, we can apply a corresponding set of reactions in a rearranged order based on their priority to reach the empty configuration.

Let M contain the set of edges to be a perfect b-matching for G. We apply reactions in Priority iCRN corresponding to the edges in M as follows.

1. Apply all reactions $\gamma = \lambda_i + \lambda_j \to \phi$ corresponding to edges $e_k = (v_i, v_j)$, $f(e_k)$ times, if $e_k \in M$, $i \neq j$ and $\mathcal{P}(\gamma) = 0$. All such reactions are uninhibited and can be applied.
2. Apply all reactions $\gamma = \lambda_i + \lambda_i \to \phi$ corresponding to edges $e_k = (v_i, v_i^1)$ and $e_\ell = (v_i, v_i^2)$, $f(e_k)$ times, if $e_k, e_\ell \in M$ and $\mathcal{P}(\gamma) = 0$. The vertices v_i^1 and v_i^2 are only connected to each other and to vertex v_i. To achieve perfect b-matching for vertices v_i^1 and v_i^2, assigned values $f(e_k)$ and $f(e_\ell)$ must be exactly equal, $f(e_k) = f(e_{\ell)} = b(v_i^1) - f(e_m)$ where $e_m = (v_i^1, v_i^2)$. Applying such a reaction will delete $2 * f(e_k)$ copies of species λ_i.
3. For all p in 1 through $\max_{\gamma \in \Gamma} \mathcal{P}(\gamma)$:
   a. apply all reactions $\gamma = \lambda_i + \lambda_j \to \phi$ corresponding to edges $e_k = (v_i, v_j)$, $f(e_k)$ times, if $e_k \in M$, $i \neq j$ and $\mathcal{P}(\gamma) = p$.
   b. apply all reactions $\gamma = \lambda_i + \lambda_i \to \phi$ corresponding to edges $e_k = (v_i, v_i^1)$ and $e_\ell = (v_i, v_i^2)$, $f(e_k)$ times, if $e_k, e_\ell \in M$, and $\mathcal{P}(\gamma) = p$.

Furthermore, the reactions applied in each iteration in Step 3 are guaranteed to be applicable (absence of inhibitors and sufficient count of reactants) because (1) all species count go to zero; (2) no reaction that uses a reactant λ_i has a priority $p > i$; and (3) each reaction e_k is only applied $f(e_k)$ times. If multiple reactions use a species λ_i, the total applications of all those reactions $\sum_{e \in \delta(v_i)} f(e)$ will be exactly $b(v_i)$. Therefore, rearranging applications of each of those reactions will not affect the availability of reactants in later applications. ◀

▶ **Theorem 15.** *Given a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ with only void rules of size $(2, 0)$, the reachability problem is decidable in $O(|\Lambda|^2 \log(|\Lambda|)(|\Gamma| + |\Lambda| \log(|\Lambda|)))$ time.*

Proof. Given a Priority iCRN with starting configurations $\overrightarrow{C_s}$ and target configuration $\overrightarrow{C_t}$, create the configuration $\overrightarrow{C_s} - \overrightarrow{C_t}$. From Lemma 12, empty configuration reachability is decidable from $\overrightarrow{C_s} - \overrightarrow{C_t}$ in polynomial time. Finally, from Lemma 10, empty configuration reachability in $(2, 0)$ Priority iCRN implies reachability. ◀

▶ **Lemma 16.** *Given a CRN $\mathcal{C} = (\Lambda, \Gamma)$ with only void rules of size $(k, k-1)$, the reachability problem is decidable in $\mathcal{O}(|\Lambda|^2|\Gamma|)$ time (Complete proof in [12]).*

Proof (Sketch). The dynamic programming algorithm uses a $|\Lambda| \times (|\Lambda| + 1)$ table D of boolean entries, where each row represents a different species. Since each $(k, k-1)$ catalytic void reaction reduces the count of exactly one species, only one of the reactions need to be applied at most $\overrightarrow{C_s}[\lambda_i] - \overrightarrow{C_t}[\lambda_i]$ times to reduce the count of a species λ_i. Hence, only $|\Lambda|$ distinct reactions need to be considered to reach the target configurations, if possible. The total number of rule applications, however, will be polynomial in the volume of the system.

A bottom-up approach is used to determine the ordering of the species in which their count needs to be reduced. This is essential since a species λ_i shouldn't go to zero before it is used as a catalyst in other reactions that need to be applicable. The algorithm starts by filling first column of D with 1 for species who have reached their target count. It then progresses column-by-column, applying reactions based on the ordering on species. If the last column has all 1s, then the configuration is reached. ◀

▶ **Theorem 17.** *Given a Priority iCRN $\mathcal{C}_{\mathcal{PI}} = ((\Lambda, \Gamma), \mathcal{P})$ with only void rules of size $(k, k-1)$, the reachability problem is decidable in $\mathcal{O}(|\Lambda|^2|\Gamma|)$ time.*

Proof. This follows as a modification of the dynamic programming algorithm for general CRNs with only void rules of size $(k, k-1)$ from Lemma 16. Let $\overrightarrow{C_f}$ denote the target configuration, with $\lambda_1, \ldots, \lambda_{|\Lambda|}$ denoting the order assigned to the species. Remove all rules of the form $\lambda_j + (.) \to^k (.)$, where $j < k$. The rest of the system remains the same.

Start with a counter $l = |\Lambda|$. Construct an $|\Lambda| \times (|\Lambda| + 1)$ table $D(s, j)$ of boolean entries, where each row represents a different species. Reduce the count of each species to $max(k, s^f)$, where s^f represents the final count of species s, if there exists a rule that can do so. Starting from the first column $j = 0$, place a 1 if the respective species is already in its final count. Then, for each entry $D(s, j)$, place a 1 if $D(s, j-1)$ is a 1, or if there exists a reaction $\gamma \in \Gamma$ for $s \in \Lambda$ that reduces s to its final count such that 1) $\overrightarrow{C_f}[s] > 0$ or 2) $s = \lambda_l$, where all the reactants of γ have either reached their final counts or will not prevent the reaction from occurring once they do. Any time species λ_l has found a reaction to reach a final count 0, set $l = l - 1$. If column $|\Lambda| + 1$ contains all 1's, then reachability is possible. Otherwise, it is not.

The algorithm follows the same bottom-up approach as for general CRNs. The main difference is that now there exists a predetermined ordering that species must go to zero for other rules to be applicable. Thus, any species λ_j such that $j < l$ is not considered until species λ_l has found a suitable reaction and reaches a count of zero. The proof then follows from general CRNs. ◀

▶ **Lemma 18** (Rearrangement Lemma for Void (Priority) iCRNs). *For any sequence of applicable void rules A in a given iCRN, there exists a sequence B that is a permutation of A such that all applications of a given rule type occur contiguously.*

Proof. Consider a sequence of applicable void rules A for a given (Priority) iCRN that is not contiguous. We construct a contiguous sequence as described in the Rearrangement Lemma in [12]. For the given sequence A, suppose rule γ occurs at positions i and j such that $i < j - 1$, and there is at least one non-γ rule in between i and j. Construct a sequence A' by shifting the rule γ at position i up to position $j - 1$, and shifting all rules in between down one position.

This new sequence must be applicable as (1) the only rule that moved to a higher index in the sequence is of type γ, and we know that γ is applicable at position $j - 1$ since it is applicable at position j, and (2) because, in a system with just void rules, species are only deleted and not produced. Therefore, all rules that occur between positions i and j must not be inhibited by the reactants of γ (since the reactants of γ are present until position j); therefore, such rules can be applied before all occurrences of γ.

Since this swapping preserves the applicability of the sequence while reducing the number of non-contiguous blocks of one rule type in the sequence, we can repeat this process of swapping rule positions until the sequence is contiguous. ◀

▶ **Lemma 19.** *The reachability problem for (Priority) iCRNs with only void reactions is in NP.*

Proof. Given a sequence of applicable void rules in a given iCRN, we encode this sequence with a sequence of rule types accompanied by a count on the number of applications of each rule type, which must exist by Lemma 18. Although the original sequence is potentially exponential in length, the result of the contiguous sequence can be computed in polynomial time. Therefore, we utilize a contiguous sequence of applicable rules as a certificate for the reachability problem. ◀

▶ **Theorem 20.** *The reachability problem in Priority iCRN with only* $(2,0)$ *and* $(2,1)$ *void rules with maximum priority* 1 *is NP-complete.*

Proof. Due to space constraints, we provide a proof sketch here, while the complete proof can be found in the full version of the paper on arxiv.

From Lemma 19, we have that the reachability problem for Priority iCRNs with purely void rules is in NP. To show hardness, we reduce from Vertex Cover. Given an instance of VC $\langle G, k\rangle$ where $k > 0$, we construct a Priority iCRN $\mathcal{C}_{\mathcal{PI}}$. We create one copy of v for all $v \in V$ and one copy of e for all $e \in E$. We also add two new species to the $\mathcal{C}_{\mathcal{PI}}$ system. In the initial configuration, we add $|V| - k$ copies of a species R, used to eliminate $|V| - k$ vertices, and k copies of a species X, used to remove the remaining k vertices. Denote this configuration the starting configuration $\overrightarrow{C_s}$. We also order our species set as: $\Lambda = \langle R, v_1, \ldots, v_{|V|}, e_1, \ldots, e_{|E|}, X\rangle$.

For all $v \in V$, we create the *assignment reaction* $v + R \to \emptyset$ and the *clean-up reaction* $v + X \xrightarrow{1} \emptyset$. For all edges $e \in E$, for all vertices $v \in V$ where v is incident to e, we create the *covering reaction* $v + e \xrightarrow{1} v$. Each of the clean-up and covering reactions have priority 1.

The target configuration $\overrightarrow{C_t}$ is the empty configuration $\overrightarrow{0}$. A vertex cover of size k can only exist in G iff all the species copies can be completely deleted. ◀

▶ **Lemma 21.** *The reachability problem in (Priority) iCRNs with only reactions of size* $(2,2)$ *is in PSPACE.*

Proof. Given that the system is volume-preserving, the reachability problem is solvable by a non-deterministic Turing machine using only polynomial space, where the tape of the machine stores a configuration (counts of species). Therefore, the problem is in NPSPACE, and hence in PSPACE [27]. ◀

▶ **Observation 22.** *The reachability problem in Priority iCRNs with only reactions of size* $(2,2)$ *is PSPACE-complete.*

Proof. The hardness follows from the reachability problem in regular CRNs with only $(2,2)$-size rules being PSPACE-complete [11], and membership follows from Lemma 21. ◀

4 Inhibitory CRNs

We now shift our focus to the reachability problem in the iCRN model. Specifically, we prove that reachability of iCRNs with void rules of size $(2,0)$, $(2,0) + (2,1)$, and more generally $(k, k-1)$ are all NP-complete. Alongside these hardness results, we present FPT algorithms for each of these reachability problems. To conclude, we examine iCRN systems with rules of size $(1,1)$ and show that the reachability problem is PSPACE-complete.

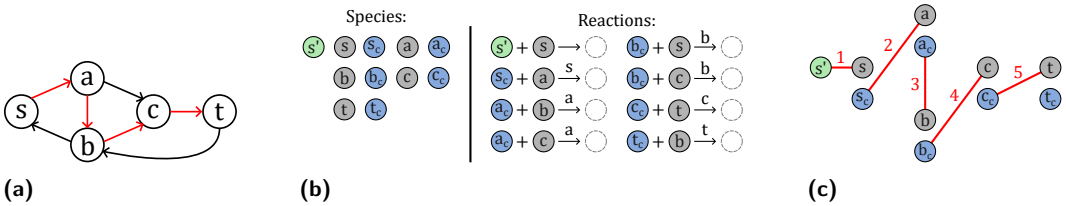


Figure 2 Transforming an instance of HAMPATH into an instance of iCRN reachability with only $(2,0)$-size reactions. (a) is a directed graph G with a Hamiltonian path highlighted in red. (b) is the set of species and reactions of $\mathcal{C}_{\mathcal{IC}}$ as constructed from G. (c) is the complete reachability reduction, as well as showing how the Hamiltonian path is traversed with $\mathcal{C}_{\mathcal{IC}}$. The red line between two species copies represents their deletion by a $(2,0)$ void reaction; the number represents the order the reaction was applied.

4.1 iCRNs with Only Void Rules

For completeness, we include the following observation.

▶ **Observation 23.** *Given an iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ with only void rules of size $(1,0)$, the reachability problem is decidable in $\mathcal{O}(|\Lambda||\Gamma|)$ time.*

Proof. Run any applicable rule in the system, reducing the corresponding species to its final count. If the final configuration is reached, then reachability is possible. Otherwise, if no more rules can be applied, reachability is not possible. ◀

▶ **Theorem 24.** *The reachability problem in iCRNs with only rules of size $(2,0)$ is NP-complete.*

Proof. Due to space constraints, we provide a proof sketch here, while the complete proof can be found in full version of the paper on arxiv.

Lemma 19 shows reachability in void iCRNs is in NP. To show hardness, we reduce from the Hamiltonian path problem. Given a HAMPATH instance $\langle G, s, t\rangle$, we transform it into an instance of reachability with an iCRN $\mathcal{C}_{\mathcal{IC}}$. For each vertex $v \in V$, we create the species v and v_c. We also add an extra species s'. For each directed edge $\{u, v\} \in E$, we create the *choosing reaction* $u_c + v \xrightarrow{u} \emptyset$. We also create the *starting reaction* $s' + s \to \emptyset$. Let the initial configuration $\overrightarrow{A}$ be a single copy of each species and the target configuration $\overrightarrow{B}$ be a single copy of t_c. A Hamiltonian path from s to t can only exist in G iff the system can delete all copies except for t_c. An example reachability instance is illustrated in Figure 2. ◀

We now explore the reachability problem for a $(2,0)$ void iCRN with a constant number of inhibiting species. We follow the same format as in Priority iCRN section to show that the reachbility problem is polynomial time solvable for a constant number of inhibitors. To do so we show that reachability problem in such an iCRN can be reduced to the empty reachability problem in Lemmas 25, 26, and 27. Finally, we reduce the empty reachability problem in the given iCRN to the empty reachability problem in Priority iCRN with $(2,0)$ reactions.

Given a $(2,0)$ void iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$, where the set of species Λ is partitioned into the set of inhibiting species $\Lambda_{\mathcal{I}} = \bigcup_{\gamma \in \Gamma} \mathcal{I}(\gamma)$ and the set of remaining species $\Lambda \setminus \Lambda_{\mathcal{I}}$. We use $\overrightarrow{0}$ to denote an empty configuration where all species have a count of zero.

▶ **Lemma 25.** *Given two configurations $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$ in a (2,0) void iCRN $\mathcal{C}_{\mathcal{IC}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$, if $\forall \lambda_i \in \Lambda_{\mathcal{I}}, \overrightarrow{C_t}[\lambda_i] = 0$.*

Proof. Follows from Lemma 8 because the proof does not rely on any ordering of inhibitors. ◀

▶ **Lemma 26.** *Given two configurations $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$ in a (2,0) void iCRN $\mathcal{C}_{\mathcal{IC}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$, if $\exists \lambda_i \in \Lambda_{\mathcal{I}}, \overrightarrow{C_t}[\lambda_i] \neq 0$.*

Proof. The proof is similar to that of Lemma 9. Here we remove any reactions inhibited by species $\lambda_i \in \Lambda_{\mathcal{I}}$. Now λ_i is no longer an inhibitor. The remaining set of reactions is Γ'. From Lemma 25, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$ in a iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma'), \mathcal{I})$. ◀

▶ **Lemma 27.** *Given two configurations $\overrightarrow{C_s}$ and C_t in a (2,0) void iCRN $\mathcal{C}_{\mathcal{IC}}$, $\overrightarrow{C_s} \rightsquigarrow \overrightarrow{C_t} \iff \overrightarrow{C_s} - \overrightarrow{C_t} \rightsquigarrow \overrightarrow{0}$.*

Proof. Follows from Lemmas 25 and 26. ◀

▶ **Lemma 28.** *Given any iCRN system $(\mathcal{C}_{\mathcal{IC}}, \overrightarrow{C_s})$ and an ordering on inhibitors, we can construct a Priority iCRN system $(\mathcal{C}_{\mathcal{PI}}, \overrightarrow{C_s'})$ that contains the same inhibitors in the given ordering. This construction takes $O(|\Lambda| \cdot |\Gamma|)$ where Λ and Γ are the set of species and reactions in the given iCRN.*

Proof (Sketch). We construct an ordered species set Λ' containing all species from the given iCRN. The species are arranged such that all the inhibitors (in order) appear first in the set. The configuration and reactions are also rearranged to match the ordering of species in Λ'. For each reaction, the priority is set to be the maximum index of all inhibitors. Finally, we remove any reactions that are self-inhibiting, i.e., the priority of the reaction is greater or equal to the index of at least one of the reactants. The complete proof can be found in full version of the paper on arxiv. ◀

▶ **Lemma 29.** *Given an iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ with only void rules of size $(2, 0)$, the empty configuration reachability problem is decidable in $O(c! \cdot f(\Lambda, \Gamma))$ time where c is the number of inhibitors and $f(\Lambda, \Gamma) = |\Lambda|^2 \log(|\Lambda|)(|\Gamma| + |\Lambda| \log(|\Lambda|))$.*

Proof. Given an iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ with $(2, 0)$ void rules and some start configuration $\overrightarrow{C_s}$. Given that the iCRN has at most c inhibitors, there exist at most $c!$ different orderings in which the inhibitors are removed.

To solve the empty configuration reachability problem in iCRN, we reduce it to $c!$ instances of the empty configuration reachability problem in Priority iCRN. From Lemma 12 we know that this problem is solvable in $O(f(\Lambda, \Gamma))$. For each ordering $\langle i_1, i_2, \ldots, i_c \rangle$ where i_j is some species that acts as an inhibitor, we construct a Priority iCRN as described in Lemma 28. We now show that the empty configuration is reachable in the given iCRN iff the empty configuration is reachable in at least one of the $c!$ Priority iCRNs.

▷ Claim 1. If the empty configuration is reachable in iCRN, then it is also reachable in at least one of the constructed Priority iCRNs.

This is true simply by construction of the Priority iCRNs. If the iCRN reaches the empty configuration, then all species, including inhibitor species, go to zero. Therefore, there must exist an ordering in which these inhibitors are deleted. Since we consider every possible ordering, at least one of the $c!$ Priority iCRNs orderings will reach the empty configuration.

▷ Claim 2. If the empty configuration is reachable in at least one of the constructed Priority iCRNs, then the iCRN reaches the empty configuration.

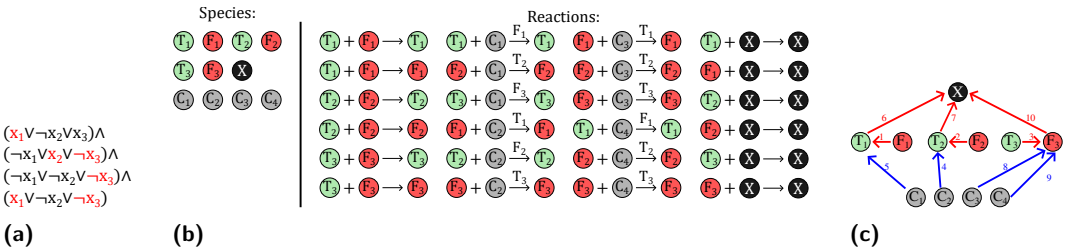


Figure 3 Transforming an instance of 3SAT into an instance of iCRN reachability with only $(2,1)$-size reactions. (a) is a 3-CNF formula Φ. A satisfying assignment of Φ is represented by the red letters. (b) is the set of species and reactions of $\mathcal{C}_{\mathcal{IC}}$ as constructed from Φ. (c) is the complete reachability reduction, as well as showing how the formula is satisfied with $\mathcal{C}_{\mathcal{IC}}$. The red line between two species copies represents their deletion by a $(2,1)$ void reaction.

Let the ordering corresponding to the Priority iCRN for which the empty configuration is reachable be $\langle i_1, \ldots, i_c \rangle$. Modify the reaction set Γ of the given iCRN by removing all reactions of the form $i_j + _ \to \phi$ or $_ + i_j \to \phi$ if they are inhibited by species $i_{k>j}$ since i_j is deleted before i_k. For the remaining reactions, apply the reactions in the same order in which their corresponding reactions in $\mathcal{C}_{\mathcal{PI}}$ are applied. If none of the $c!$ Priority iCRNs reach the empty configuration, then the given iCRN will also not reach the empty configuration. ◀

▶ **Theorem 30.** *Given an iCRN $\mathcal{C}_{\mathcal{IC}} = ((\Lambda, \Gamma), \mathcal{I})$ with only void rules of size $(2,0)$, the reachability problem is decidable in $O(c! \cdot f(\Lambda, \Gamma))$ time where c is the number of inhibitors and $f(\Lambda, \Gamma) = |\Lambda|^2 \log(|\Lambda|)(|\Gamma| + |\Lambda| \log(|\Lambda|))$.*

Proof. Similar to the case of $(2,0)$ Priority iCRN, proof follows from Lemmas 27 and 29. ◀

▶ **Theorem 31.** *The reachability problem in (2,1) void iCRN is NP-complete.*

Proof. Due to space constraints, we provide a proof sketch here, while the complete proof can be found in full version of the paper on arxiv.

From Lemma 19, we have that the reachability problem in void iCRNs is in NP. To show hardness, we reduce from the 3-satisfiability problem. Given an instance of 3SAT $\langle \Phi \rangle$, we create a new iCRN $\mathcal{C}_{\mathcal{IC}}$. For each variable $x_i \in X_n$, we create the species T_i and F_i. Additionally, for each clause $c_j \in C_m$, we create the species c_j. Finally, we create the species X. For each variable x_i, we also create the *assignment reactions* $T_i + F_i \to T_i$ and $T_i + F_i \to F_i$. For each clause c_j and a variable of c_j x_k, we create the *clause reaction* $C_i + T_k \xrightarrow{F_k} T_k$ if assigning true to x_k satisfies c_j, or $C_i + F_k \xrightarrow{T_k} F_k$ if the false assignment satisfies instead. Finally, for each variable, we create the *clean-up reaction* $T_i + X \to X$ and $F_i + X \to X$. The initial configuration $\overrightarrow{C_s}$ is a single copy of each species and the target configuration $\overrightarrow{C_t}$ is a single copy of X. Φ can only be satisfied iff the system can delete all copies except for X. An example reachability instance is illustrated in Figure 3. ◀

▶ **Theorem 32.** *The reachability problem in $(k, k-1)$ void iCRN is NP-complete.*

Proof. Follows from Theorem 31. Given an iCRN $\mathcal{C}_{\mathcal{IC}}$, we modify it by adding a species d and modifying each reaction to include $k-2$ copies of d for both reactants and products (i.e., $T_x + F_x + d \to T_x + d$ for $k = 3$). We include $k-2$ copies of d for $\overrightarrow{C_s}$ and $\overrightarrow{C_t}$. Since d acts as a catalyst species for all reactions, its count will never change, and it does not affect the behavior of $\mathcal{C}_{\mathcal{IC}}$. Thus, the forward and reverse directions from Lemma 31 follow here. ◀

▶ **Theorem 33.** *Given an iCRN $\mathcal{C}_{\mathcal{CI}} = ((\Lambda, \Gamma), \mathcal{I})$ with only void rules of size $(k, k-1)$, the reachability problem is decidable in $\mathcal{O}(c! \cdot |\Lambda|^2|\Gamma|)$ time, where c is the number of inhibitors.*

Proof. This follows by using the algorithm for Priority iCRN void systems with size $(k, k-1)$ rules from Theorem 17 as a subroutine. Denote this algorithm g. Consider an arbitrary ordering of the c inhibitors $\lambda_1, \ldots, \lambda_c$. Run algorithm g with this ordering. If a solution is found, then reachability is possible. Otherwise, consider another ordering until all are considered. The algorithm then runs in $\mathcal{O}(c! \cdot |\Lambda|^2|\Gamma|)$ time. ◀

We show that reachability is NP-complete with $(2, 0)$ and $(2, 1)$ reactions, even if only one species is used to inhibit reactions.

▶ **Theorem 34.** *The reachability problem in (2,0) and (2,1) void iCRNs is NP-complete, even with a single inhibitor species.*

Proof. From Lemma 19, we have that the reachability for iCRNs with purely void rules is in NP. We now show hardness with a similar reduction as in Theorem 20. We construct our iCRN $\mathcal{C}_{\mathcal{IC}}$ in a similar manner, where our species set is now unordered. Here, the species set $\Lambda = \{R, v_1, \ldots, v_{|V|}, e_1, \ldots, e_{|E|}, X\}$. Our reaction set contains *assignment reactions* $v + R \to \phi$, *clean-up reactions* $v + X \xrightarrow{R} \phi$, and *covering reactions* $v + e \xrightarrow{R} v$. The initial and target configurations remain the same. ◀

4.2 iCRNs with General rules

▶ **Theorem 35.** *The reachability problem in iCRNs with $(1, 1)$ rules is PSPACE-complete.*

Proof. PSPACE membership follows from Lemma 21. We now show hardness. Given a Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r)$ with bounded tape length n, we create an iCRN $\mathcal{C}_{\mathcal{IC}} = (\Lambda, \Gamma_C)$. The set Λ consists of the helper species D, H, and E, the tape symbols $\{x_i, 0_i, 1_i\}$ for each tape cell $i \in [1, n]$, and the following sets of species: $\mathcal{Q} = \{Q_y | q_y \in Q\}$, $\mathcal{Q}^0 = \{\, Q_y 0_i \,\}_{q_y \in Q,\ i \in [1,n]}$ and $\mathcal{Q}^1 = \{\, Q_y 1_i \,\}_{q_y \in Q,\ i \in [1,n]}$. Table 2 shows the reactions created.

The initial configuration I of the system consists of one instance of 0_i or 1_i for each tape cell i, according to the initial tape of $\mathcal{M}$. Given the start state q_s, we also add a single copy of Q_s. If the tape head starts at cell i, we include one instance of each species x_j for all $j \neq i$, i.e., $\{x_j\}_{j \neq i}$. Finally, we include one copy each of the helper species D and H. The target configuration T is volume I copies of E.

Forward Direction. Assume we are given a Turing Machine that will reach a halt state by applying a series of transitions from δ, each according to the tape value at head and the state. The corresponding iCRN $\mathcal{C}_{\mathcal{IC}}$ will likewise reach target configuration T by applying a series of reactions representing those transitions, according to the current configuration C. When a single D is present in C, C maps to the Turing Machine. Q_y indicates $\mathcal{M}$ is in state $q_y \in Q$. The Tape head is at tape cell j where x_j is absent. The data for tape cell$_i$ is saved in the presence of a copy of 1_i, or one of 0_i. All reactions shown in Table 2 except 1.3, 7.1 and 7.2 handle normal transitions, updating the given configuration to map to the result of the next transition. In this order, the reactions: (1) read the tape, (2) determine the next state, (3) mark the tape head ready to move, (4) change the state, (5) write and move the tape head, and (6) indicate ready for next transition. If the current state is a halt state, reaction 1.3 is applied, than (7) is repeatedly applied until we reach a terminal configuration. The final terminal configuration will only be T if the TM's tape and state input deterministically transitions to q_a according to δ.

Table 2 Reaction set for the $(1,1)$ iCRN construction.

No.	Notation	Reaction	Intuition
1.1	$\mathcal{Q}_H : Q_a, Q_r$	$D \xrightarrow{\mathcal{Q}_H, \overline{\mathcal{Q}_y}, x_i, 0_i} Q_y 1_i$	Read the tape can combine with state, unless state is halt.
1.2	$\overline{\mathcal{Q}_y} : \mathcal{Q} \setminus Q_y$	$D \xrightarrow{\mathcal{Q}_H, \overline{\mathcal{Q}_y}, x_i, 1_i} Q_y 0_i$	
1.3	$\overline{\mathcal{Q}_H} : \mathcal{Q} \setminus \mathcal{Q}_H$	$D \xrightarrow{\overline{\mathcal{Q}_H}} E$	
2	$\overline{\mathcal{Q}^V}_{\{y,i\}} : \{\mathcal{Q}^0 \cup \mathcal{Q}^1\} \setminus Q_y V_i$ $z : \delta_{state}(y, V)$	$Q_y \xrightarrow{D, E, \overline{\mathcal{Q}^V}_{\{y,i\}}} Q_{zi}$	Prepare to change into the appropriate z according to δ.
3.1		$1_i \xrightarrow{D, E, \mathcal{Q}, x_i} x_i$	Mark read as done by producing x_i.
3.2		$0_i \xrightarrow{D, E, \mathcal{Q}, x_i} x_i$	
4		$Q_{zi} \xrightarrow{1_i, 0_i} Q_z$	Finish changing state after producing x_i.
5	$b : \delta_{dir}(y, V)$ $c : \delta_{write}(y, V)$	$x_{i+b} \xrightarrow{E, 1_i, 0_i, \overline{\mathcal{Q}^V}_{\{y,i\}}, Q_{zi}} c_i$	Move+write based on state and value read V.
6		$Q_y V_i \xrightarrow{x_{i+b}} D$	Once moved, replace D.
7.1	$\mathcal{Q}' : \{\mathcal{Q}^1 \cup \mathcal{Q}^0\}$	$H \xrightarrow{D, \mathcal{Q}'} E$	If $\mathcal{M}$ halts, delete H, and go to term config.
7.2	$S : \Lambda \setminus \{E, Q_r\}$	$S \xrightarrow{H} E$	

Reverse Direction. To reach the target configuration T of iCRN $\mathcal{C}_{\mathcal{IC}}$ which was designed to simulate bounded length N TM A, reaction 7.2 must run $2N$ times. This can only happen if Q_a was present, preventing reactions 1.1 and 1.2, and not being Q_r, which would also prevent the reactions, but could not be removed by 7.2. For Q_a to be present, it either was in I, or was produced by a type 4 reaction. To have Q_a produced by a type 4 reaction, the most recent type 2 reaction must have produced Q_{ai}. For any reaction of type 2 to be applicable, D cannot be present, so before this reaction could occur, a reaction of type 1.1 or 1.2 must have been run. It produced a copy of $Q_y V_i$ where Q_y represents the state the system was in before Q_a, and V is the symbol at our tape head. $Q_y V_i$ prevents all type 2 reactions except $\delta_{state}(y, V)$, so we could only have reached Q_a if $\delta_s state(y, V) = q_a$. For our configuration to have included a 1_i or 0_i and Q_y which would lead to Q_a, they either must have been in I, or were produced by repetitions of the same process. This process also includes one reaction of type 3 and one of type 5, which together update the tape. Thus we could only have reached the target configuration if the initial tape values and State deterministically Halt in an accept state. ◀

▶ **Theorem 36.** *The reachability problem in iCRN with* $(1,1)$ *rules and at most 1 inhibitor per rule is NP-Hard.*

Proof. We show this by a reduction from 3SAT. Consider a 3SAT instance of n variables $X_1, \ldots X_n$ and m clauses $C_1, \ldots C_m$, and assume no clause contains both x_i and $\overline{x_i}$. From this instance, we create the following $(1,1)$ iCRN system and start and end configurations for the Reachability problem.

For each variable $x_i \in X_n$, we create the species y_i, x_i, T_i, and F_i. For each clause $c_j \in C_m$, we create the species c_j^1,c_j^2,c_j^3, d_j^1,d_j^2,d_j^3, and S_j. Additionally, for each variable, we create the reactions $y_i \to T_i$, $y_i \to F_i$, $x_i \xrightarrow{x_{i-1}} T_i$ (no inhibitor for $i = 1$), and $x_i \xrightarrow{x_{i-1}} F_i$ (no inhibitor for $i = 1$). For each clause, we create the reactions $c_j^k \xrightarrow{x_n} d_j^k$, either $d_j^k \xrightarrow{T_i} S_j$

if $\overline{x_i}$ satisfies the k^{th} literal of clause C_j or $d_j^k \xrightarrow{F_i} S_j$ if x_i satisfies the k^{th} literal of clause C_j, and $d_j^k \to W$. Finally, let the initial configuration be one copy for each of y_i, x_i, and c_j^1, c_j^2, c_j^3, and the target configuration be one copy for each of T_i, F_i, S_j, and $2m$ copies of W.

Forward Direction. Suppose the 3SAT formula is satisfiable. We can then reach the destination configuration by first transitioning each x_i species into the appropriate true/false species T_i/F_i that matches the satisfying assignment for the 3SAT formula. Next, convert exactly one of the three c_j species from each clause into S_j (based on which literal was satisfied by the satisfying variable assignment), and the remaining two into copies of species W. Finally, convert each y_i into the opposite truth value taken on by x_i (to ensure there is exactly one T_i and F_i for each i).

Reverse Direction. Suppose we can reach the destination configuration, which specifically requires creating each species S_j. Each S_j can only be created by the absence of some F_i where the clause contains literal X_i, or the absence of a T_i where the clause contains $\bar{X}_i$. Since the c_j to d_j transformation must precede the creation of S_j, we know that each x_i must be converted to a T_i or F_i before the S_j's can be created. Therefore, the only possible way to create each S_j is for each x_i to select a truth assignment that corresponds to a satisfying assignment of the given 3SAT formula. ◀

▶ **Observation 37.** *The reachability problem for iCRNs with rule size* $(2, 2)$ *is PSPACE-complete.*

Proof. The proof is analogous to the proof of Observation 22. ◀

5 Conclusion

This paper provides a thorough treatment of reachability in restricted CRNs with inhibition. Our results (Table 1) show a clear boundary between tractable / intractable cases under prioritized and unprioritized inhibition. For Priority iCRNs, reachability remains solvable in polynomial time for most unimolecular, bimolecular, and mostly catalytic void rules, but becomes NP-complete with combined $(2, 0)$ and $(2, 1)$ void rules. For general iCRNs, nearly all cases become NP-complete (except $(1, 0)$), and we give FPT algorithms for the $(2, 0)$, $(2, 1)$, and $(k, k-1)$ cases. For volume-preserving reactions, $(1, 1)$ reachability – polynomial-time solvable for basic CRNs – drastically increases to PSPACE-complete for general iCRNs.

While many of our complexity classifications are tight, several gaps remain in Table 1. For inhibitory systems with $(2, 0)$ and $(2, 1)$ void rules, we establish NP-completeness with even one inhibitor, but it remains open whether alternative parameters (e.g., the number of inhibited rules) yield FPT algorithms or whether the problem is instead W[1]-hard. It may be possible to improve our existing FPT algorithms with some pruning method that avoids exhaustively trying all inhibitor permutations. Another key question is the complexity of $(1, 1)$ Priority iCRNs: the problem is in P for basic CRNs and PSPACE-complete for general inhibition, but it is unclear where prioritized inhibition falls. Lastly, we consider *CRNs with States*, analogous to VASS, where reactions are conditioned on a global state; this model can be viewed as a form of inhibition and motivates studying its reachability complexity. In the full version of this paper, we formalize CRNs with States and provide preliminary results that show reachability is NP-hard for $(1, 0)$ rules and PSPACE-complete for $(1, 1)$ rules.

References

1 Robert M. Alaniz, Bin Fu, Timothy Gomez, Elise Grizzell, Andrew Rodriguez, Marco Rodriguez, Robert Schweller, and Tim Wylie. Reachability in restricted chemical reaction networks. *Theoretical Computer Science*, 1056:115514, 2025. doi:10.1016/j.tcs.2025.115514.

2 Rachel Anderson, Alberto Avila, Bin Fu, Timothy Gomez, Elise Grizzell, Aiden Massie, Gourab Mukhopadhyay, Adrian Salinas, Robert Schweller, Evan Tomai, and Tim Wylie. Computing threshold circuits with void reactions in step chemical reaction networks. In *10th conference on Machines, Computations and Universality (MCU 2024)*, 2024.

3 Richard P. Anstee. A polynomial algorithm for b-matchings: An alternative approach. *Information Processing Letters*, 24(3):153–157, 1987. doi:10.1016/0020-0190(87)90178-5.

4 Rutherford Aris. Prolegomena to the rational analysis of systems of chemical reactions. *Archive for Rational Mechanics and Analysis*, 19(2):81–99, January 1965. doi:10.1007/BF00282276.

5 Rutherford Aris. Prolegomena to the rational analysis of systems of chemical reactions ii. some addenda. *Archive for Rational Mechanics and Analysis*, 27(5):356–364, January 1968. doi:10.1007/BF00251438.

6 Kim Calabrese and David Doty. Rate-independent continuous inhibitory chemical reaction networks are turing-universal. In *International Conference on Unconventional Computation and Natural Computation*, pages 104–118. Springer, 2024. doi:10.1007/978-3-031-63742-1_8.

7 Luca Cardelli, Marta Kwiatkowska, and Max Whitby. Chemical reaction network designs for asynchronous logic circuits. *Natural computing*, 17:109–130, 2018. doi:10.1007/S11047-017-9665-7.

8 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13:517–534, 2014. doi:10.1007/S11047-013-9393-6.

9 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic bioprocesses*, pages 543–584. Springer, 2009. doi:10.1007/978-3-540-88869-7_27.

10 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1229–1240. IEEE, 2022.

11 Javier Esparza, Mikhail Raskin, and Chana Weil-Kennedy. Parameterized analysis of immediate observation petri nets. In Susanna Donatelli and Stefan Haar, editors, *Application and Theory of Petri Nets and Concurrency*, pages 365–385, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-21571-2_20.

12 Bin Fu, Timothy Gomez, Ryan Knobel, Austin Luchsinger, Aiden Massie, Marco Rodriguez, Adrian Salinas, Robert Schweller, and Tim Wylie. Reachability in deletion-only chemical reaction networks. In *Proc. of the International Conference on DNA Computing and Molecular Programming*, DNA, 2025.

13 Roland Guttenberg. Flattability of priority vector addition systems. *arXiv preprint arXiv:2402.09185*, 2024. doi:10.48550/arXiv.2402.09185.

14 Michel Henri Théodore Hack. *Decidability questions for Petri Nets.* PhD thesis, Massachusetts Institute of Technology, 1976.

15 Michel Henri Theódore Hack. Petri net language. Technical report, Massachusetts Institute of Technology, 1976.

16 Allen Hjelmfelt, Edward D Weinberger, and John Ross. Chemical implementation of neural networks and turing machines. *Proc. of the National Academy of Sciences*, 88(24):10983–10987, 1991.

17 Hua Jiang, Marc D Riedel, and Keshab K Parhi. Digital logic with molecular reactions. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 721–727. IEEE, 2013. doi:10.1109/ICCAD.2013.6691194.

18 Richard M Karp and Raymond E Miller. Parallel program schemata: A mathematical model for parallel computation. In *8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, pages 55–61. IEEE, 1967. doi:10.1109/FOCS.1967.27.

19 Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

20 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1241–1252. IEEE, 2022.

21 Richard Lipton. The reachability problem requires exponential space. *Research Report 62. Department of Computer Science, Yale University*, 1976.

22 Ernst W Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 238–246, 1981. doi:10.1145/800076.802477.

23 Rajiv Teja Nagipogu and John H. Reif. Neural crns: A natural implementation of learning in chemical reaction networks. *ACS Synthetic Biology*, 14(10):3899–3912, 2025. doi:10.1021/acssynbio.5c00099.

24 James Lyle Peterson. Petri net theory and the modeling of systems, 1981.

25 Carl Adam Petri. *Communication with automata*. PhD thesis, Technische Universitat Darmstadt, 1966.

26 Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008. Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008). doi:10.1016/j.entcs.2008.12.042.

27 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

28 David Soloveichik, Georg Seelig, and Erik Winfree. Dna as a universal substrate for chemical kinetics. *Proc. of the National Academy of Sciences*, 107(12):5393–5398, 2010.

29 Boya Wang, Chris Thachuk, Andrew D Ellington, Erik Winfree, and David Soloveichik. Effective design principles for leakless strand displacement systems. *Proceedings of the National Academy of Sciences*, 115(52):E12182–E12191, 2018.

Improved Bounds for Online TSP on the Half-Line

Júlia Baligács
University of Warsaw, Poland

Yann Disser
TU Darmstadt, Germany

Linda Thelen
TU Darmstadt, Germany

Abstract

In the open online traveling salesperson problem, requests appear over time at different positions of a metric space. A single agent traveling at unit speed must serve all requests, by visiting their positions, with the objective of minimizing the completion time. We propose online algorithms for this problem for the case that the metric space is the half-line. First, we present a 2-competitive algorithm in which the server always either moves at unit speed or remains stationary, which improves on the previously best-known ratio of 2.04. We further observe that algorithms must sometimes move slower than unit speed to achieve competitive ratios below 2. We introduce a second algorithm that beats the barrier of 2 by carefully modulating its speed, and prove a tight bound of 1.75 on its competitive ratio. Finally, we slightly strengthen a known lower bound on the best-possible competitive ratio on the half-line from 1.627 to 1.646.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, competitive analysis, server problems, online TSP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.4

Funding *Júlia Baligács*: Supported by the project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).

1 Introduction

We study the online traveling salesperson problem (TSP) on the half-line. In this problem, requests of the form $r = (t, p)$ are revealed over time, where $t \in \mathbb{R}_{\geq 0}$ denotes the release time and $p \in \mathbb{R}_{\geq 0}$ the position of the request. A single server, initially located at the origin and moving at speed at most 1, must serve every request by visiting position p at some time not earlier than time t. In the online setting, the algorithm only learns about a request at its release time, whereas an offline algorithm knows the entire request sequence from time 0. The objective is to minimize the makespan, i.e., the time when all requests are served. Note that the offline setting does not coincide with the classical TSP, as the offline algorithm is still restricted to serving a request after its release time. Furthermore, we consider the *open* version of the problem, where, in contrast to the *closed* version, the server *does not* need to return to the origin.

Online TSP models dynamic routing scenarios arising, for instance, in robotic maintenance, cleaning, or collection tasks where jobs appear over time. In many such settings, the depot lies at the boundary of a one-dimensional operating region, such as inspection along a track, pipeline, or corridor, so that the underlying metric space is the half-line, making it a natural case to study. While the online TSP has been extensively studied in many variants [3, 4, 5, 12, 18, 19, 20, 25], in particular for the underlying metric space being the real line $\mathbb{R}$ or higher-dimensional Euclidean spaces $\mathbb{R}^n$, we focus here on algorithms tailored specifically to the half-line $\mathbb{R}_{\geq 0}$. Although this setting appears simple at first glance,

© Júlia Baligács, Yann Disser, and Linda Thelen;
licensed under Creative Commons License CC-BY 4.0

obtaining optimal strategies turns out to be surprisingly challenging. The half-line has fundamentally different structural properties due to the lack of symmetry around the origin, and it seems that optimal strategies cannot be obtained by simple adaptations of known algorithms. For instance, we will see that, unlike any algorithm previously proposed for online TSP, an optimal online strategy on the half-line must already start moving before the first request is revealed.

As usual in online optimization, we measure the quality of a (deterministic) algorithm in terms of *competitive analysis*. For an algorithm ALG and a request sequence σ, let $\text{ALG}(\sigma)$ denote the completion time of ALG on σ, and let $\text{OPT}(\sigma)$ denote the completion time of an optimal offline solution. We say that ALG is *ρ-competitive* if $\text{ALG}(\sigma) \leq \rho\,\text{OPT}(\sigma)$ for every request sequence σ. The *competitive ratio of* ALG is the infimum over all such ρ, and the *competitive ratio of the problem* is the best ratio achievable by any online algorithm.

Throughout the paper, we assume without loss of generality that every request $r = (t, p)$ satisfies $t \geq p$. Indeed, since the server moves at speed at most 1, even the offline optimum cannot reach position p before time p, and is therefore unaffected by this assumption. Moreover, an online algorithm could only benefit from learning about a request earlier. Hence, this assumption is justified when the request sequence is constructed adversarially and does not affect the competitive ratio of the problem.

Finally, we introduce the following notational conventions. Given an online algorithm ALG, we write $\text{pos}(t) \in \mathbb{R}_{\geq 0}$ for the position of the server at time t. For a request sequence σ, we sometimes abbreviate the completion times $\text{ALG}(\sigma)$ and $\text{OPT}(\sigma)$ by ALG and OPT when the sequence is clear from context. At any point in time, the server can move in (at most) two directions. We refer to the direction towards the origin as *left* and to the direction away from the origin as *right*.

Our results

We begin by introducing the algorithm LEFTFIRST (cf. Algorithm 1), which follows a very simple strategy: it prioritizes requests that lie to the server's left over those to its right. We show that this algorithm achieves a competitive ratio of 2, improving on the previously best known ratio of 2.04 [11]. We further observe that no algorithm restricted to moving either at full speed or not at all can achieve a better competitive ratio.

▶ **Theorem 1.** *The algorithm* LEFTFIRST *is* 2*-competitive for online TSP on the half-line. Moreover, this ratio is optimal among all algorithms in which the server is restricted to speeds* 0 *and* 1*.*

In particular, the theorem shows that fundamentally new ideas are required to beat the barrier of 2. Our main contribution is to propose such an idea in the form of a parameterized algorithm $\text{LEFTGUARD}(\alpha)$ for $\alpha \in (0, 1]$ (cf. Algorithm 2). Its strategy can be summarized as follows. The algorithm maintains at all times that $\text{pos}(t) \leq \alpha t$, enabling the server to reach requests to its left in a timely manner. It assigns strict deadlines to such requests and prioritizes requests to its right if and only if these deadlines can still be met. Our main result is a careful analysis of this algorithm. We show that, for $\alpha = 0.75$, it achieves a competitive ratio of 1.75. After identifying the most difficult instances for LEFTGUARD, a key ingredient of our proof is an LP-duality argument that yields sharp performance bounds. Moreover, we prove that this analysis is tight in the sense that $\text{LEFTGUARD}(\alpha)$ has competitive ratio at least 1.75 for every choice of α.

▶ **Theorem 2.** $\text{LEFTGUARD}(0.75)$ *is 1.75-competitive for open online TSP on the half-line. Moreover, the choice* $\alpha = 0.75$ *is optimal.*

We complement our result by a general lower bound on the best-possible competitive ratio that slightly improves the known lower bound of 1.627 [26].

▶ **Theorem 3.** *Every algorithm for open online TSP on the half-line has competitive ratio at least 1.6463.*

Related work

While this is, to the best of our knowledge, the first work to specifically focus on open online TSP on the half-line, the problem has been studied before on other metric spaces. Most notably, Bjelde et al. tightly analyzed the problem on the line with a competitive ratio of 2.04 [11]. Their upper bound also applies to the special case of the half-line and, prior to our work, was the best known upper bound in this setting. Their lower bound extends to general metric spaces and remains the best known lower bound there. The best known algorithm for general metric spaces, introduced by Bonifaci and Stougie, achieves a competitive ratio of $1 + \sqrt{2} \approx 2.41$ [13].

By contrast, the closed version of the problem is tightly analyzed in all three cases: It has a competitive ratio of 2 on general metric spaces [5], of 1.64 on the line [5, 11] and of 1.5 on the half-line [12]. The optimal algorithm for the closed version on the half-line follows a strategy that can be viewed as the "opposite" of our algorithm LeftFirst: That algorithm moves right whenever a request to the server's right is available and otherwise moves left.

Online TSP has also been investigated in several related settings, for instance when the metric space is a circle [20], when distances are asymmetric [3], or when the request sequence satisfies additional structural assumptions [12, 25].

The online TSP can be viewed as a special case of the extensively studied *online dial-a-ride problem* [2, 7, 9, 10, 17, 24, 26], where each requests is a transportation request consisting of a release time, a starting position, and a destination. In other words, the online TSP corresponds to the special case of online dial-a-ride in which starting position and destination always coincide. Classical online dial-a-ride strategies include Ignore, which repeatedly recomputes a schedule for the currently unserved requests [8, 23]; Replan, which greedily recomputes an optimal tour whenever a new request arrives [5, 8, 23]; and Lazy, which deliberately delays service to aggregate requests before executing a tour [7, 26].

Other online variants of the TSP focus on different types of uncertainty. In *online graph exploration* [6, 15, 16, 22], the metric space is initially unknown and all vertices must be visited. In *universal TSP* [14, 21], the goal is to find a fixed master tour that is competitive for any subset of active requests. Recently, Abrahamsen et al. [1] introduced a model where requests are revealed one-by-one and must be irrevocably sorted to minimize total travel distance, rather than minimizing the makespan for requests arriving over time.

2 Warm-up: moving at speed 1 or 0

Before considering the problem in full-generality, we first consider the easier case where the server is only allowed to move with full speed or not move at all. We first observe a simple lower bound on the competitive ratio of this variant of the problem.

▶ **Lemma 4.** *If the server is only allowed to move at speed 1 or 0, the competitive ratio of open online TSP on the half-line is at least 2. The same lower bound applies when the server is not allowed to move before the first request is revealed.*

Algorithm 1 LeftFirst.

```
if there is a left-request:
    MoveLeft
else
    if there is a right-request:
        MoveRight
    else
        Stay
```

Proof. Fix an algorithm Alg in which the server can move only at speed 0 or 1. Then there exists some time $\varepsilon > 0$ such that, before time ε, the server has not changed its speed. Hence $\text{pos}(\varepsilon) \in \{0, \varepsilon\}$. If $\text{pos}(\varepsilon) = 0$, we release the request $r = (\varepsilon, \varepsilon)$, and if $\text{pos}(\varepsilon) = \varepsilon$, we release the request $r' = (\varepsilon, 0)$. In both cases, we have $\text{Opt} = \varepsilon$, while Alg cannot complete the request before time 2ε. Therefore, the competitive ratio is at least 2. Observe that even if the server is allowed to move at intermediate speeds, as long as it does not move before the first request is revealed, the same argument still applies. ◀

Next, we give an algorithm whose competitive ratio matches this lower bound.

2.1 A 2-competitive algorithm

When considering algorithms for online TSP on the half-line (or even on the line), observe that, at any point in time, the algorithm only needs to consider at most two requests: the request with the leftmost position among all unserved requests to the algorithm's left (referred to as the *left-request*) and the request with the rightmost position among all unserved requests to the algorithm's right (referred to as the *right-request*). At some times, there may be no left-request or no right-request. This is the case when all unserved requests lie on the same side of the server. If there are multiple requests at the same position, we select the one with the latest release time. By serving the left- and right-request, the algorithm necessarily also serves all other requests on the way. In our analysis, we therefore consider only left- and right-requests. The online algorithm is thus defined by a strategy that determines when to serve the left-request, when to serve the right-request, and when to wait.

The algorithm LeftFirst is formally defined in Algorithm 1 and follows a simple strategy: It prioritizes left-requests over right-requests. More precisely, whenever a left-request is available, the server executes command MoveLeft, i.e., travels towards the origin at full speed (note that MoveLeft is never invoked when the server is located in the origin). If no left-request is available but a right-request exists, the server executes command MoveRight, i.e., it moves away from the origin at full speed. If there are no unserved requests, the server follows command Stay, i.e., remains stationary.

Before turning to the analysis of the algorithm, we state a simple observation that will be useful throughout our paper. Since the offline optimum cannot serve requests before they are released, the following holds.

▶ **Observation 5.** *If t is the release time of any request of the input sequence, then* $\text{Opt} \geq t$.

Now we have all the prerequisites at hand to analyze the algorithm LeftFirst.

▶ **Theorem 6.** LeftFirst *is 2-competitive.*

Proof. Fix an arbitrary request sequence. Let $r = (t, p)$ denote the last request served by LeftFirst. First, observe that, if r is a left-request, we have

$$\textsc{LeftFirst} \leq t + (\mathrm{pos}(t) - p) \leq t + \mathrm{pos}(t) \leq 2t \overset{\text{Obs. 5}}{\leq} 2\textsc{Opt},$$

where we have made used of the fact that $\mathrm{pos}(t) \leq t$ since the server cannot move at speed more than 1. Therefore, we can assume from now on that r is a right-request.

Next, observe that, if the server does not move left after time t, we have

$$\textsc{LeftFirst} \leq t + (p - \mathrm{pos}(t)) \leq t + p \leq 2t \overset{\text{Obs. 5}}{\leq} 2\textsc{Opt},$$

where we have used the assumption that every request fulfills $t \geq p$.

Therefore, we can assume from now on that there was an unserved left-request after time t and let $r_L = (t_L, p_L)$ denote the last left-request served by LeftFirst. Observe that we have

$$\textsc{LeftFirst} \leq t_L + (\mathrm{pos}(t_L) - p_L) + (p - p_L). \tag{1}$$

We distinguish now the following two cases: (1) The offline optimum serves r_L before r or (2) The offline optimum serves r before r_L. In the first case, we have

$$\textsc{Opt} \geq t_L + (p - p_L). \tag{2}$$

With (1) and (2), we obtain

$$\textsc{LeftFirst} \leq t_L + (\mathrm{pos}(t_L) - p_L) + (p - p_L) = \underbrace{t_L + p - p_L}_{\leq \textsc{Opt}} + \underbrace{\mathrm{pos}(t_L)}_{\leq t_L \leq \textsc{Opt}} - \underbrace{p_L}_{\geq 0} \leq 2\textsc{Opt}.$$

In the second case, i.e., where Opt serves r before r_L, we have

$$\textsc{Opt} \geq t + (p - p_L). \tag{3}$$

Moreover, observe that we have $\mathrm{pos}(t_L) \leq t$ because either $t \geq t_L \geq \mathrm{pos}(t_L)$ or $t_L \geq t$ and therefore $r = (t, p)$ was an unserved right-request at time t_L. With (1) and (3), we obtain

$$\textsc{LeftFirst} \leq t_L + \underbrace{\mathrm{pos}(t_L)}_{\leq t} - \underbrace{p_L}_{\geq 0} + p - p_L \leq \underbrace{t_L}_{\leq \textsc{Opt}} + \underbrace{t + p - p_L}_{\leq \textsc{Opt}} \leq 2\textsc{Opt}.$$

To summarize, we have in either case that LeftFirst ≤ 2Opt, which completes the proof of the theorem. ◀

Note that Theorem 6 and Lemma 4 together complete the proof of Theorem 1.

3 Improved algorithm for open online TSP on the half-line

In this section, we prove our main result (Theorem 2), i.e., we prove that the competitive ratio of open online TSP on the half-line is at most 1.75. For this, we begin with introducing the algorithm LeftGuard.

Algorithm 2 LeftGuard(α).

if *there is a left-request* $r_L = (t_L, p_L)$**:**
  if *there is a right-request* $r_R = (t_R, p_R)$ *with* $\max\{t + p_R - \mathrm{pos}(t), \frac{p_R}{\alpha}\} + p_R - p_L \leq (1+\alpha)t_L$**:**
    MoveRight(α)
  else
    MoveLeft
else
  MoveRight(α)

Algorithm MoveRight(α).

if $\mathrm{pos}(t) < \alpha t$**:**
  move right at speed 1
else
  move right at speed α

Algorithm MoveLeft.

move left at speed 1

3.1 The algorithm LeftGuard

The algorithm LeftGuard is formally defined in Algorithm 2. It is parameterized by a value $\alpha \in (0, 1]$ and we let t denote the current time. Its intuition can be summarized as follows: First, recall that no algorithm is better than 2-competitive if the server does not move while being idle (Lemma 4), so we let the server move at speed α before the first request is revealed. As before, we consider always at most two requests – the left-request and the right-request. The most important property of our algorithm is that every left-request $r_L = (t_L, p_L)$ has a strict deadline, meaning that it must be served no later than at time $(1 + \alpha)t_L$. To make this possible, the server maintains at every time t that $\mathrm{pos}(t) \leq \alpha t$. Therefore, whenever a new left-request $r_L = (t_L, p_L)$ appears, the server is located at $\mathrm{pos}(t_L) \leq \alpha t_L$ so that, by moving left immediately, position $p_L \geq 0$ can be reached by time $(1 + \alpha)t_L$. However, if the server can serve a right-request without violating the deadlines of left-requests, it proceeds to do so. Note that a right-request $r_R = (t_R, p_R)$ can be reached by time $\max\{t + p_R - \mathrm{pos}(t), \frac{p_R}{\alpha}\}$, where the first term corresponds to moving from $\mathrm{pos}(t)$ to p_R at unit speed, and the second term is the arrival time at p_R in case $\mathrm{pos}(t) \leq \alpha t$ becomes tight. Therefore, the server prioritizes a right-request over a left-request if and only if $\max\{t + p_R - \mathrm{pos}(t), \frac{p_R}{\alpha}\} + p_R - p_L \leq (1 + \alpha)t_L$.

The algorithm definition contains the subroutines MoveLeft and MoveRight(α). When MoveLeft is invoked, it simply orders the server to move towards the origin at unit speed (observe that MoveLeft is never invoked when the server is located at the origin). MoveRight(α) depends on the parameter α and orders the server to move away from the origin at full speed maintaining that $\mathrm{pos}(t) \leq \alpha t$. In other words, the server moves at speed 1 as long as $\mathrm{pos}(t) < \alpha t$ and at speed α when $\mathrm{pos}(t) = \alpha t$. In particular, the server always moves either at speed 1 or at speed α. When it moves at speed α, we say that it *slows down*.

Let us summarize the most important properties that immediately follow from the algorithm's definition.

▶ **Observation 7.** LeftGuard(α) *fulfills the following properties.*

i) *At every time t, the server position fulfills* $\mathrm{pos}(t) \leq \alpha t$.

ii) *Every left-request $r_L = (t_L, p_L)$ is served no later than at time $(1 + \alpha)t_L$, unless a new left-request appeared before time $(1 + \alpha)t_L$.*

iii) *Every right request $r_R = (t_R, p_R)$ is served no earlier than at time p_R/α. Moreover, if the server is slowed down (i.e., moving at speed α) at the time of serving r_R, it is served precisely at time p_R/α.*

3.2 Upper bound for the competitive ratio of LeftGuard

We now turn to analyzing the algorithm LeftGuard. More precisely, we show the following.

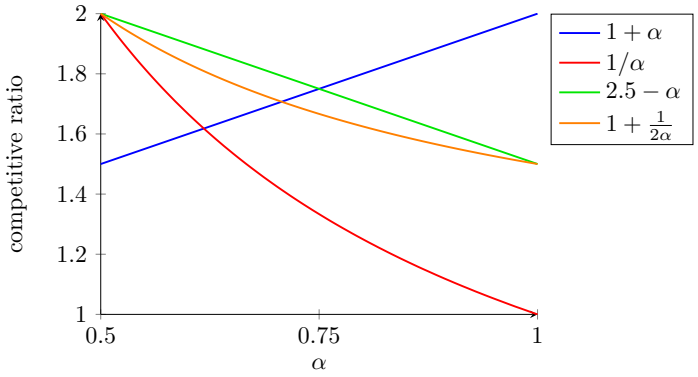


Figure 1 Bounds on the competitive ratio from Theorem 8 depending on α.

▶ **Theorem 8.** *For $\alpha \in (0.3, 1]$, the competitive ratio of* LeftGuard(α) *is upper-bounded by* $\max\{1+\alpha, 1/\alpha, 2.5-\alpha, 1+1/(2\alpha)\}$.

Note that, by setting $\alpha = 0.75$, we obtain a competitive ratio of 1.75 (cf. Figure 1), i.e., the statement of the first part of Theorem 2 follows.

We now prove Theorem 8. To this end, for the remainder of this subsection, we fix an arbitrary input sequence of requests, and we show that LeftGuard(α) achieves the claimed ratio on this sequence. We denote by Opt an optimum offline solution, or its completion time, depending on context. We denote the last right-request served by LeftGuard(α) by $r_R = (t_R, p_R)$, and the last left-request served by LeftGuard(α) by $r_L = (t_L, p_L)$. In case there is no left-request, it is immediate that the competitive ratio is bounded by $1/\alpha$ (by Observations 5 and 7 iii), and in case there is no right-request, it is bounded by $1+\alpha$ (by Observations 5 and 7 ii). Similarly, we immediately obtain the desired bounds in the following two cases.

▶ **Observation 9.** *If one of the following conditions hold, then* LeftGuard(α) *completes the request sequence by* $\max(1+\alpha, 1/\alpha)$Opt*:*

i) LeftGuard(α) *serves r_L after r_R.*

ii) LeftGuard(α) *serves r_R while moving at speed α*

Therefore, we assume from now on that r_L and r_R both exist, and that the server served first r_L and then moved at full speed to r_R. Next, we identify two further simple cases within these assumptions.

▶ **Lemma 10.** *Assume that* LeftGuard(α) *serves r_L before r_R and always moves at unit speed after serving r_L. If one of the following two conditions hold, the request sequence is completed by time* $(1+\alpha)$Opt.

i) Opt *serves r_L before r_R.*

ii) $t_L \le t_R$.

Proof. In both cases, Opt serves neither of r_L, r_R earlier than at time t_L, and then travels to the other request, which takes $p_R - p_L$ time units. Therefore,

$$\text{Opt} \ge t_L + p_R - p_L\,. \tag{4}$$

By assumption, LeftGuard first serves r_L and then moves at full speed to p_R. By Observation 7 ii), r_L is reached no later than $(1+\alpha)t_L$ and then moving to r_R takes $p_R - p_L$ time units. It follows that the request sequence is completed by time

$$(1+\alpha)t_L + p_R - p_L = t_L + p_R - p_L + \alpha t_L \overset{(4)}{\le} \text{Opt} + \alpha t_L \overset{\text{Obs. 5}}{\le} (1+\alpha)\text{Opt}\,. \qquad \blacktriangleleft$$

The remaining case turns out to be the most difficult to analyze and constitutes the crux of our proof. While the two previous results hold for any $\alpha \in (0, 1]$, we need in the following lemma that $\alpha \geq 0.3$. This is a reasonable assumption, since for $\alpha \leq 0.3$, the bound $1/\alpha$ from Observation 9 is already signigicantly worse than the bound of 2 achieved by LeftFirst.

▶ **Lemma 11.** *Assume that* LeftGuard(α) *serves* r_L *before* r_R *and always moves at unit speed after serving* r_L*. Additionally, assume that* $t_L > t_R$ *and* Opt *serves* r_R *before* r_L*. Then* LeftGuard(α) *completes the request sequence by time* $\max(2.5 - \alpha, 1 + 1/(2\alpha))\textsc{Opt}$.

Proof. First, observe that in the given setting, when r_L appeared at time t_L, the server decided to not detour to serve r_R before r_L. In other words, at time t_L, the condition in line 2 of Algorithm 2 was violated, i.e., we have

$$\max\left\{t_L + p_R - \text{pos}(t_L), \frac{p_R}{\alpha}\right\} + (p_R - p_L) > (1 + \alpha)t_L\,. \tag{5}$$

Further, note that the completion time of LeftGuard(α) is given by

$$\textsc{LeftGuard}(\alpha) \leq t_L + (\text{pos}(t_L) - p_L) + (p_R - p_L) = t_L + \text{pos}(t_L) + p_R - 2p_L \tag{6}$$

and the completion time of the offline optimum is bounded by

$$\textsc{Opt} \geq t_R + p_R - p_L. \tag{7}$$

Next, we will make use of the following observation: The behavior of LeftGuard is independent of multiplying all release times and positions with the same factor $x > 0$. In particular, this allows us to assume w.l.o.g. that $\textsc{Opt} = 1$ by rescaling with $1/\textsc{Opt}$ (if $\textsc{Opt} = 0$, then $\textsc{Alg} = 0$ is trivial). Summarizing our findings, it follows that the competitive ratio is bounded by the maximum value of the following optimization problem (here, Alg, Opt, $t_L, t_R, p_L, p_L, \text{pos}(t_L)$ are interpreted as variables of the optimization problem):

$$\begin{aligned}
\max\quad & \textsc{Alg} && \\
\text{s.t.}\quad & \textsc{Opt} \geq t_L && \text{(Observation 5)}\\
& \textsc{Opt} \geq t_R + p_R - p_L && (7)\\
& \textsc{Alg} \leq t_L + \text{pos}(t_L) + p_R - 2p_L && (6)\\
& \max(t_L + p_R - \text{pos}(t_L), p_R/\alpha) + p_R - p_L \geq (1 + \alpha)t_L && \text{(5, relaxed)}\\
& \text{pos}(t_L) \leq p_R && (r_R \text{ is a right-request})\\
& t_R \geq p_R && \text{(by assumption)}\\
& \textsc{Opt} = 1 && \text{(scaling)}\\
& \textsc{Alg}, \textsc{Opt}, t_R, t_L, p_R, p_L, \text{pos}(t_L) \geq 0\,. &&
\end{aligned}$$

Observe that the problem is not linear due to the maximum-operation in (5). Therefore, we distinguish now the two cases $t_L + p_R - \text{pos}(t_L) \geq p_R/\alpha$ and $t_L + p_R - \text{pos}(t_L) < p_R/\alpha$. For each of the two cases, we obtain a linear program, which allows us to use duality. In the end, the competitive ratio will be bounded by the maximum of the two values of the optimization problems.

First, we consider the case $t_L + p_R - \text{pos}(t_L) \geq p_R/\alpha$. Here, we obtain the following LP, where it turns out that the condition $\text{pos}(t_L) \leq p_R$ can be dropped.

$$
\begin{aligned}
\max\quad & \textsc{Alg} \\
\text{s.t.}\quad & \textsc{Opt} \geq t_L && \text{(Observation 5)} \\
& \textsc{Opt} \geq t_R + p_R - p_L && (7) \\
& \textsc{Alg} \leq t_L + \mathrm{pos}(t_L) + p_R - 2p_L && (6) \\
& t_L + p_R - \mathrm{pos}(t_L) + p_R - p_L \geq (1+\alpha)t_L && \text{(5, relaxed)} \\
& t_R \geq p_R && \text{(by assumption)} \\
& \textsc{Opt} = 1 && \text{(scaling)} \\
& \textsc{Alg}, \textsc{Opt}, t_R, t_L, p_R, p_L, \mathrm{pos}(t_L) \geq 0\,.
\end{aligned}
$$

A relaxation of the LP and its dual (D) can equivalently be written as

$$
\text{(P)} \qquad \max \quad (1,0,0,0,0,0,0)^\top \boldsymbol{x} \quad \text{s.t.}
$$

$$
\begin{pmatrix}
0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 1 & -1 & 0 \\
1 & 0 & 0 & -1 & -1 & 2 & -1 \\
0 & 0 & 0 & \alpha & -2 & 1 & 1 \\
0 & 0 & -1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \boldsymbol{x} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
$$

$$
\boldsymbol{x} \geq \mathbf{0}
$$

$$
\text{(D)} \qquad \min \quad (0,0,0,0,0,1)^\top \boldsymbol{y} \quad \text{s.t.}
$$

$$
\begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 \\
-1 & -1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & -1 & 0 \\
1 & 0 & -1 & \alpha & 0 & 0 \\
0 & 1 & -1 & -2 & 1 & 0 \\
0 & -1 & 2 & 1 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0
\end{pmatrix} \boldsymbol{y} \geq \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
$$

$$
\boldsymbol{y} \geq \mathbf{0}\,.
$$

By weak duality, any feasible solution to the dual provides an upper bound on the optimum value of the primal. The dual solution $\boldsymbol{y} = (1-\alpha, 1.5, 1, 1, 1.5, 2.5-\alpha)^\top$ is feasible for $\alpha \in (0,1]$ and has objective value $2.5 - \alpha$.

Next, we consider the case $t_L + p_R - \mathrm{pos}(t_L) < p_R/\alpha$. Here, we obtain the following LP, where it turns out that the condition $\textsc{Opt} \geq t_L$ can be dropped.

$$
\begin{aligned}
\max\quad & \textsc{Alg} \\
\text{s.t.}\quad & \textsc{Opt} \geq t_R + p_R - p_L && (7) \\
& \textsc{Alg} \leq t_L + \mathrm{pos}(t_L) + p_R - 2p_L && (6) \\
& \frac{1}{\alpha} p_R + p_R - p_L \geq (1+\alpha)t_L && \text{(5, relaxed)} \\
& \mathrm{pos}(t_L) \leq p_R && (r_R \text{ is a right-request}) \\
& t_R \geq p_R && \text{(by assumption)} \\
& \textsc{Opt} = 1 && \text{(scaling)} \\
& \textsc{Alg}, \textsc{Opt}, t_R, t_L, p_R, p_L, \mathrm{pos}(t_L) \geq 0\,.
\end{aligned}
$$

As before, we can write the dual (D) of the relaxed LP as

$$\text{(D)} \quad \begin{array}{l} \min \quad (0,0,0,0,0,1)^\top \boldsymbol{y} \quad \text{s.t.} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 1+\alpha & 0 & 0 & 0 \\ 1 & -1 & -\frac{\alpha+1}{\alpha} & -1 & 1 & 0 \\ -1 & 2 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \end{pmatrix} \boldsymbol{y} \geq \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ \boldsymbol{y} \geq \boldsymbol{0}\,. \end{array}$$

The dual solution $\boldsymbol{y} = (1 + \frac{1}{2\alpha}, 1, \frac{1}{1+\alpha}, 1, 1 + \frac{1}{2\alpha}, 1 + \frac{1}{2\alpha})^\top$ is feasible for $\alpha \geq 0.3$ (More precisely, the second-to-last inequality holds for $\alpha \geq (\sqrt{17} - 3)/4 \approx 0.281$. All other inequalities hold for all $\alpha > 0$.) The corresponding objective value is $1 + \frac{1}{2\alpha}$.

Together, we obtain that the competitive ratio is bounded by $\max(2.5-\alpha, 1+1/(2\alpha))$. ◀

Let us summarize the different cases we considered to prove Theorem 8.

Proof of Theorem 8. First, we noted that, in case there is no left-request or no right-request, we easily obtain a bound of $\max(1 + \alpha, 1/\alpha)$ on the competitive ratio. Therefore, we can assume that a left- and right-request exist and let r_L denote the last-served left-request and r_R the last-served right-request. By Observation 9, we can assume that the server first served r_L and then moved at full-speed to r_R (otherwise, we obtain a bound of $\max(1 + \alpha, 1/\alpha)$). By Lemma 10, we can additionally assume that Opt serves r_R before r_L and that r_R was released before r_L (otherwise, we obtain a bound of $1 + \alpha$). Finally, we showed in Lemma 11 that, under these assumptions, the competitive ratio is bounded by $\max(2.5 - \alpha, 1 + 1/(2\alpha))$. ◀

3.3 Lower bound for the competitive ratio of LeftGuard

In this section, we present two lower-bound constructions for LeftGuard, showing that the analysis in Section 3.2 is tight in the sense that no choice of α yields a competitive ratio below 1.75 for LeftGuard(α) (i.e., we prove the second part of Theorem 2).

▶ **Lemma 12.** *The competitive ratio of* LeftGuard(α) *is at least* $\max(1 + \alpha, 1/\alpha)$ *for every* $\alpha \in (0, 1]$.

Proof. First, consider the input sequence consisting of the single request $(1, 0)$. Note that LeftGuard(α) executes MoveRight(α) before time 1 so that $\text{pos}(1) = \alpha$. Therefore, the request is served by time $1 + \alpha$. On the other hand, Opt serves the request immediately at time 1 by waiting in the origin. Therefore, the competitive ratio is at least $1 + \alpha$.

Next, consider the input sequence consisting of the single request $(1, 1)$. Since $(1, 1)$ is a right-request, by Observation 7 iii), LeftGuard(α) completes it not earlier than at time $1/\alpha$, while we have $\text{Opt} = 1$. Therefore, the competitive ratio is at least $1/\alpha$. ◀

▶ **Lemma 13.** *The competitive ratio of* LeftGuard(α) *is at least* $2.5 - \alpha$ *for* $\alpha \in [0.5, 1]$.

Proof. Consider the input sequence consisting of the three requests $(\frac{\alpha}{1+\alpha}, 0)$, $(0.5+\varepsilon, 0.5+\varepsilon)$ and $(1, 0)$ where $0 < \varepsilon < 0.5$. Observe that $\alpha/(1 + \alpha) \leq 0.5$ for $\alpha \leq 1$ so the requests indeed arrive in the order given above.

Opt can serve the requests by moving to position $0.5+\varepsilon$ at time $0.5+\varepsilon$ and then moving to position 0, i.e., we have

$$\text{Opt} = 1 + 2\varepsilon. \tag{8}$$

On the other hand, LeftGuard(α) moves away from the origin with speed α until time $\alpha/(1+\alpha)$ so that $\text{pos}(\alpha/(1+\alpha)) = \alpha^2/(1+\alpha)$. By moving back to the origin at unit speed, LeftGuard(α) could serve the first request at time

$$\frac{\alpha}{1+\alpha} + \frac{\alpha^2}{1+\alpha} = \alpha = (1+\alpha) \cdot \frac{\alpha}{1+\alpha}\,,$$

which is the maximum time allowed to serve the first request according to the algorithm definition. Therefore, LeftGuard(α) cannot decide to serve another right-request before serving the left-request at position 0, or before another left-request is revealed. Hence, LeftGuard(α) serves the left-request at position 0 at time $\alpha \leq 1$, and then moves right at maximum speed maintaining $\text{pos}(t) \leq \alpha t$ to serve the right-request at position $0.5+\varepsilon$. We obtain $\text{pos}(1) = 1-\alpha < 0.5+\varepsilon$ (where we have used that $\alpha \geq 0.5$), i.e., the request $(0.5+\varepsilon, 0.5+\varepsilon)$ is still unserved when the new left-request $(1,0)$ arrives.

Serving the right-request first, Alg would arrive at the left-request at time

$$1 + (0.5+\varepsilon) - \text{pos}(1) + 0.5 + \varepsilon = 2 + 2\varepsilon - (1-\alpha) = 1 + \alpha + 2\varepsilon > (1+\alpha) \cdot 1\,,$$

hence LeftGuard(α) has to move to the left-request first before serving the right-request. It follows that LeftGuard(α) finishes at time

$$1 + \text{pos}(1) + 0.5 + \varepsilon = 1 + (1-\alpha) + 0.5 + \varepsilon = 2.5 - \alpha + \varepsilon\,.$$

Together with (8) and letting $\varepsilon \to 0$, we obtain that the competitive ratio is at least $2.5-\alpha$. ◀

For $\alpha \leq 0.5$, Lemma 12 implies that the competitive ratio of LeftGuard(α) is at least $1/\alpha \geq 2$. For $\alpha > 0.5$, Lemmas 12 and 13 together imply that the competitive ratio is at least $\max(1+\alpha, 2.5-\alpha) \geq 1.75$. Thus, this completes the proof that LeftGuard(α) has competitive ratio at least 1.75 for any choice of α (i.e., the second part of Theorem 2).

4 General lower bound for online TSP on the half-line

In the following, we fix an arbitrary online algorithm Alg and prove a general lower bound for the competitive ratio of online TSP on the half-line.

▶ **Theorem 3.** *Every algorithm for open online TSP on the half-line has competitive ratio at least 1.6463.*

It is immediate to see that no online algorithm for the TSP on the half-line can achieve a competitive ratio strictly smaller than 3/2: At time 1, we reveal either a request at 0 if $\text{pos}(1) \geq 1/2$, or a request at 1 otherwise. In both cases, we have Alg $\geq 3/2$ and Opt $= 1$.

For the remainder of this section, we fix $\rho \in (3/2, 5/3]$ and assume that Alg has competitive ratio strictly less than ρ. We progressively introduce requests that, based on the fact that it must be ρ-competitive if no further requests appear, constrain Alg's behavior more and more. Specifically, we present two fixed requests, followed by three requests that depend on Alg's behavior. Based on the derived restrictions on Alg, we show that the resulting request sequence yields a lower bound of $\rho \geq 1.6463$. We denote by $\text{serve}(r_i)$ the time when the request $r_i = (t_i, p_i)$ is served by Alg and formally define the adversarial request sequence as follows.

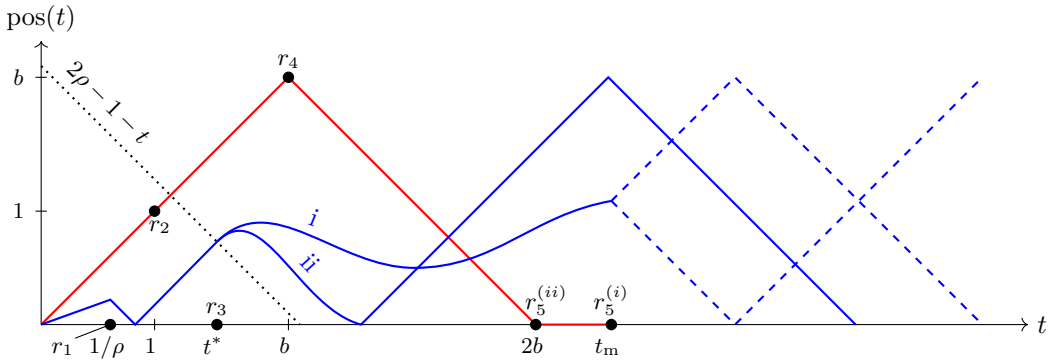


Figure 2 Adversarial request sequence for the lower bound in Theorem 3 for $\rho = 1.64$ with best ALG in blue and the offline optimum in red. Curved lines in ALG's trajectory indicate that the online algorithm has degrees of freedom in its behavior, whereas straight lines indicate that any other behavior increases the lower bound on the competitive ratio. Dashed lines show that there are two possible trajectories. The trajectories marked (i) and (ii) correspond to the two cases in the definition of r_5.

▶ **Definition 14** (Adversarial request sequence). *We present an adversarial request sequence* $(r_1, r_2, r_3, r_4, r_5)$ *consisting of the requests*

1. $r_1 := (1/\rho, 0)$ *and* $r_2 := (1, 1)$,
2. $r_3 := (t^*, 0)$, *where* $t^* := \min\{t \geq 1 : \mathrm{pos}(t) = 2\rho - 1 - t\}$,
3. $r_4 := (b, b)$ *for some* $b \geq t^* + 1/2$ *to be determined later,*
4. r_5 *depending on* ALG*'s behavior:*
   - **i.** *if* $\mathrm{pos}(2b) < b/2$ *and* $\mathrm{serve}(r_4) > 2b$, *then* $r_5 := (t_\mathrm{m}, 0)$ *with* $t_\mathrm{m} := \min\{t \geq 2b : \mathrm{pos}(t_\mathrm{m}) = b/2\}$,
   - **ii.** *otherwise,* $r_5 := (2b, 0)$.

We show in Lemmas 15 and 18 that this sequence is well-defined, in the sense that the time t^* in the second step and, if needed, the time t_m in the fourth step exist.

The adversarial sequence is illustrated in Figure 2 for $\rho = 1.64$. The figure also shows the trajectory of the optimum offline solution for the entire request sequence, as well as the best possible trajectory of ALG.

By $\sigma_{\leq i} := (r_1, r_2, \ldots, r_i)$ we denote the sequence of requests revealed up to and including request r_i, and we write $\mathrm{ALG}(\sigma_{\leq i})$ and $\mathrm{OPT}(\sigma_{\leq i})$ for the cost of ALG and the offline optimum, respectively, for serving the request sequence $\sigma_{\leq i}$. Moreover, we write $\mathrm{ALG}[\sigma_{\leq i}]$ for the schedule ALG computes for the request sequence $\sigma_{\leq i}$, i.e., ALG's behavior if only the requests $r_1, \ldots, r_i$ were revealed. Note that the serving times $\mathrm{serve}(r_i)$ are always w.r.t. the full schedule $\mathrm{ALG}[\sigma_{\leq 5}]$.

The following lemma establishes that the time t^* as described in the second step of Definition 14 exists and falls within certain bounds.

▶ **Lemma 15.** *After releasing the requests* r_1 *and* r_2 *of Definition 14, there exists a time* $t^* \in [\rho + (3/(2\rho)) - 1, 2]$ *such that* $\mathrm{pos}(t^*) = 2\rho - 1 - t^*$. *Moreover,* $\mathrm{serve}(r_2) > t^*$.

Proof. Observe that, before the first request r_1 is revealed, the server needs to be able to handle the case where the first request is of the form (t, t) with $t \leq 1/\rho$. In this case, the offline optimum's cost is t and the server must be able to serve the request before time ρt. Note that $\mathrm{pos}(t) \leq t$ since the server is restricted to unit speed. It follows that $t + (t - \mathrm{pos}(t)) \leq \rho t$.

Equivalently, we have

$$\mathrm{pos}(t) \geq (2-\rho)t \quad \text{for all } t \in [0, 1/\rho].$$

This implies that, when r_1 is revealed at time $1/\rho$, we have $\mathrm{pos}(1/\rho) \geq (2-\rho)/\rho$, and it follows that

$$\mathrm{serve}(r_1) \geq t_1 + |\mathrm{pos}(t_1) - p_1| = 1/\rho + \mathrm{pos}(1/\rho) \geq (3-\rho)/\rho. \tag{9}$$

Moreover, $\mathrm{serve}(r_1) \leq 1$ because ALG is ρ-competitive for the request sequence $\sigma_{\leq 1}$, i.e., r_1 is already served when r_2 is revealed. From (9), it follows that

$$\mathrm{serve}(r_2) \geq \mathrm{serve}(r_1) + |p_2 - \mathrm{pos}(\mathrm{serve}(r_1))| = \mathrm{serve}(r_1) + 1 \geq (3-\rho)/\rho + 1 = 3/\rho.$$

With this, we obtain that there is a time t^* where $\mathrm{pos}(t^*) = 2\rho - 1 - t^*$: Let $\ell(t) := 2\rho - 1 - t$. At time $\mathrm{serve}(r_1)$, we have $\mathrm{pos}(\mathrm{serve}(r_1)) = 0 < \ell(\mathrm{serve}(r_1))$. When request $r_2 = (1,1)$ is served at time $\mathrm{serve}(r_2) \geq 3/\rho$, we have $\mathrm{pos}(\mathrm{serve}(r_2)) = 1 > 2\rho - 1 - (3/\rho) \geq \ell(\mathrm{serve}(r_2))$, where the first inequality holds for $\rho \leq 1.82$. Therefore, by continuity of ℓ, there exists $t^* \in (\mathrm{serve}(r_1), \mathrm{serve}(r_2))$ such that $\mathrm{pos}(t^*) = \ell(t^*) = 2\rho - 1 - t^*$.

It remains to prove the claimed upper and lower bounds on t^*. To this end, note that r_2 is served no earlier than time $t^* + (1 - \mathrm{pos}(t^*)) = t^* + 1 - (2\rho - 1 - t^*) = 2t^* + 2 - 2\rho$. Moreover, $\mathrm{OPT}(\sigma_{\leq 2}) \leq 1 + 1/\rho$ (wait in $p_1 = 0$ until time $t_1 = 1/\rho$ and then move up to $p_2 = 1$) so that $2t^* + 2 - 2\rho \leq \rho\mathrm{OPT}(\sigma_{\leq 2}) \leq \rho + 1$. Reformulating and using $\rho \leq 5/3$ yields

$$t^* \leq \frac{3\rho - 1}{2} \leq 2\,.$$

To bound t^* from below, observe that, using (9) and $\mathrm{pos}(\mathrm{serve}(r_1)) = p_1 = 0$, we have for all $t \geq \mathrm{serve}(r_1)$ that $\mathrm{pos}(t) \leq \mathrm{pos}(\mathrm{serve}(r_1)) + t - \mathrm{serve}(r_1) \leq t - (3-\rho)/\rho$. By definition of time t^*, it therefore holds that

$$2\rho - 1 - t^* = \mathrm{pos}(t^*) \leq t^* - (3-\rho)/\rho\,,$$

or, equivalently, $t^* \geq \rho + 3/(2\rho) - 1$. ◀

The time t^* is chosen precisely such that ALG is forced to serve r_2 before r_3, as the alternative would not be competitive in case no more requests are released.

▶ **Observation 16** (Planned order of r_2 and r_3)**.** $\mathrm{ALG}[\sigma_{\leq 3}]$ *serves* $r_2 = (1,1)$ *before* $r_3 = (t^*, 0)$.

Proof. Otherwise, $\mathrm{ALG}[\sigma_{\leq 3}]$ would finish the request sequence no earlier than

$$t^* + (\mathrm{pos}(t^*) - 0) + 1 = t^* + 2\rho - 1 - t^* + 1 = 2\rho \overset{\text{Obs. 17}}{\geq} \rho\mathrm{OPT}(\sigma_{\leq 3})\,,$$

contradicting that ALG has competitive ratio strictly less than ρ. ◀

To prove restrictions on ALG's behavior beyond this, we use the fact that ALG's competitive ratio is strictly less than ρ for the request sequences $\sigma_{\leq 3}$ and $\sigma_{\leq 4}$, whose optimum offline cost can be bounded.

▶ **Observation 17.** *It holds that*

$$\mathrm{OPT}(\sigma_{\leq 3}) \leq 2 \quad \textit{and} \quad \mathrm{OPT}(\sigma_{\leq 4}) \leq t^* + b.$$

Proof. An offline algorithm can serve $\sigma_{\leq 3}$ by moving to $p_2 = 1$ at time $t_2 = 1$ and then moving back to $p_1 = p_3 = 0$ to serve $r_1 = (1/\rho, 0)$ and $r_3 = (t^*, 0)$ at time 2, since $t^* \leq 2$, achieving completion time 2.

For $\sigma_{\leq 4}$, the offline optimum can wait in 0 until time t^* and then move to $p_4 = b > 1$, serving r_2 on the way. ◀

We can now prove lower bounds on the competitive ratio of ALG, depending on which request r_5 is released.

▶ **Lemma 18.** *If* $\mathrm{pos}(2b) < b/2$ *and* $\mathrm{serve}(r_4) > 2b$*, then the adversarial request sequence is well-defined in the sense that* t_m *as described in Definition 14 exists, and*

$$\rho \geq 1 + \frac{3b/2}{\rho(t^* + b) - b/2}.$$

Proof. Since $\mathrm{pos}(2b) < b/2$ and $r_4 = (b, b)$ is not yet served at time $2b$, the server has to cross position $b/2$ at some time t_m after $2b$. Therefore, the request $r_5 = (t_\mathrm{m}, 0)$ is well-defined.

Between times b and t_m, only requests up to r_4 are present, so the offline optimum cost is $\mathrm{OPT}(\sigma_{\leq 4}) \leq t^* + b$ by Observation 17. Since ALG is ρ-competitive, $\mathrm{ALG}[\sigma_{\leq 4}]$ must therefore serve $r_4 = (b, b)$ before time $\rho(t^* + b)$. This implies that

$$t_\mathrm{m} + b/2 = t_\mathrm{m} + |p_4 - \mathrm{pos}(t_\mathrm{m})| \leq \rho(t^* + b)\,. \tag{10}$$

Since $\mathrm{pos}(t_\mathrm{m}) = b/2$ and since $r_4 = (b, b)$ and $r_5 = (t_\mathrm{m}, 0)$ are not yet served by ALG at time t_m, we have $\mathrm{ALG}(\sigma_{\leq 5}) \geq t_\mathrm{m} + 3b/2$. Moreover, $\mathrm{OPT}(\sigma_{\leq 5}) = t_\mathrm{m}$. It follows that

$$\rho \geq \frac{\mathrm{ALG}}{\mathrm{OPT}} \geq \frac{t_\mathrm{m} + 3b/2}{t_\mathrm{m}} = 1 + \frac{3b/2}{t_\mathrm{m}} \overset{(10)}{\geq} 1 + \frac{3b/2}{\rho(t^* + b) - b/2}\,.$$ ◀

To prove a lower bound on the competitive ratio in the second case, we first establish that ALG must plan to serve r_3 before r_4, and we derive a lower bound on the time r_3 is served. We prove the claimed serving order by contradiction: assuming the opposite, we can construct an additional request for which the resulting request sequence even yields a lower bound of 1.75. To establish the lower bound on the serving time of r_3, we make use of the fact that ALG must plan to serve r_2 before r_3 (Observation 16). The details of both proofs can be found in the full version of this paper.

▶ **Lemma 19** (Serving order of r_3 and r_4)**.** $\mathrm{ALG}[\sigma_{\leq 4}]$ *serves* $r_3 = (t^*, 0)$ *before* $r_4 = (b, b)$*.*

▶ **Lemma 20** (Serving time of r_3)**.** *If* $b \geq t^* + 1/2$*, it holds that* $\mathrm{serve}(r_3) \geq 3 + 2t^* - 2\rho$*.*

Combining the previous two lemmas, we can prove a lower bound on the competitive ratio for the second case. Here, the main observation is that it is always optimal for ALG to serve r_3, then r_4 and then r_5, which allows us to use the lower bound on $\mathrm{serve}(r_3)$ to obtain a lower bound on the competitive ratio for $\sigma_{\leq 5}$.

▶ **Lemma 21.** *Let* $b \geq t^* + 1/2$*. If* $\mathrm{pos}(2b) \geq b/2$ *or* $\mathrm{serve}(r_4) \leq 2b$*, then*

$$\rho \geq 1 + \frac{3 + 2t^* - 2\rho}{2b}\,.$$

Proof. The adversarial sequence concludes by releasing $r_5 = (2b, 0)$ at time $2b$. The optimum offline cost for the entire sequence is $\mathrm{OPT}(\sigma_{\leq 5}) = 2b$ (immediately move to b and back to the origin in time $2b$). By Lemma 19, $\mathrm{ALG}[\sigma_{\leq 4}]$ serves $r_3 = (t^*, 0)$ before $r_4 = (b, b)$.

If $\text{serve}(r_4) \leq 2b = t_5$, i.e., r_4 has already been served when r_5 appears, then we have $\text{serve}(r_3) < \text{serve}(r_4) < \text{serve}(r_5)$. It follows that $\textsc{Alg}(\sigma_{\leq 5}) \geq \text{serve}(r_3) + |p_4 - p_3| + |p_5 - p_4| = \text{serve}(r_3) + 2b$. In the other case (i.e., $\text{serve}(r_4) > 2b$), we have $\text{pos}(2b) \geq b/2$ and $\text{serve}(r_4) > 2b$ by assumption. Moreover, we can show that $\text{serve}(r_3) < 2b$, i.e., r_3 is served before r_4 and r_5: Otherwise, since $\textsc{Alg}[\sigma_{\leq 4}]$ serves r_3 before r_4 (Lemma 19), it holds that

$$2\rho b \overset{b \geq t^*}{\geq} \rho(t^* + b) \overset{\text{Obs. } 17}{\geq} \rho\textsc{Opt}(\sigma_{\leq 4}) \geq \textsc{Alg}(\sigma_{\leq 4}) \geq 2b + |\text{pos}(2b) - p_3| + |p_4 - p_3| \geq 7b/2\,,$$

contradicting $\rho \leq 5/3$. Since $\text{pos}(2b) \geq b/2$ and $\text{serve}(r_4) > 2b$, the fastest way for $\textsc{Alg}$ to complete the sequence $\sigma_{\leq 5}$ is to serve $r_4 = (b, b)$ before $r_5 = (2b, 0)$. As before, we obtain $\textsc{Alg}(\sigma_{\leq 5}) \geq \text{serve}(r_3) + 2b$. It follows that

$$\rho \geq \frac{\textsc{Alg}(\sigma_{\leq 5})}{\textsc{Opt}(\sigma_{\leq 5})} \geq \frac{\text{serve}(r_3) + 2b}{2b} = 1 + \frac{\text{serve}(r_3)}{2b} \overset{\text{Lem. } 20}{\geq} 1 + \frac{3 + 2t^* - 2\rho}{2b}\,. \qquad \blacktriangleleft$$

To prove Theorem 3, we also need the following two technical lemmas, which are proven in the full version of this paper.

▶ **Lemma 22.** *Let $\rho \in [3/2, 5/3]$ and $t^* \geq \frac{7}{5}$. If b satisfies*

$$\frac{3 + 2t^* - 2\rho}{2b} = \frac{3b/2}{\rho(t^* + b) - b/2} \tag{11}$$

then $b \geq t^* + 1/2$.

▶ **Lemma 23.** *Assume that $\rho > 3/2$ is fixed and $b(t)$ is chosen such that*

$$\frac{3 + 2t - 2\rho}{2b(t)} = \frac{3b(t)}{2\rho(t + b(t)) - b(t)} =: \text{LB}(t).$$

Then $\text{LB}(t)$ is increasing in t.

Proof of Theorem 3. Lemmas 18 and 21 combined yield that, for every $b \geq t^* + 1/2$ and $\rho \in (3/2, 5/3]$, we have

$$\rho \geq 1 + \min\left(\frac{3b/2}{\rho(t^* + b) - b/2}, \frac{3 + 2t^* - 2\rho}{2b}\right) =: \text{LB}(t^*, \rho, b). \tag{12}$$

The remainder of the proof is purely analytical and dedicated to showing that, for every $\rho \in (3/2, 5/3]$ and $t^* \geq \rho + 3/(2\rho) - 1$, there exists $b \geq t^* + 1/2$ such that, in (12) one of the values in the minimum is at least 1.64634.

To this end, we choose the maximal $b(t^*, \rho)$ such that the two values in the minimum coincide. By Lemma 15, we have that $t^* \geq \rho + \frac{3}{2\rho} - 1 \geq \frac{3}{2} + \frac{3}{2} \cdot \frac{3}{5} - 1 = 7/5$, and therefore, Lemma 22 gives $b(t^*, \rho) \geq t^* + 1/2$, so that the lower bound still applies. Moreover, it follows from Lemma 23 that the resulting lower bound $\text{LB}(t^*, \rho, b(t^*, \rho))$ is increasing in t^*. Therefore, we obtain a general lower bound by setting $t^* = \rho + 3/(2\rho) - 1$ (since we have proven in Lemma 15 that t^* is lower bounded by this value). Plugging this in, we obtain

$$\rho \geq 1 + \min\left(\frac{3b}{2\rho^2 - 2\rho + 2\rho b + 3 - b}, \frac{3 + \rho}{2\rho b}\right)$$

resulting in

$$b = \frac{2\rho^2 + 5\rho - 3 + \sqrt{52\rho^4 + 116\rho^3 - 59\rho^2 + 186\rho + 9}}{12\rho}$$

and a lower bound on the competitive ratio of

$$1 + \frac{6(\rho+3)}{2\rho^2+5\rho-3+\sqrt{52\rho^4+116\rho^3-59\rho^2+186\rho+9}},$$

which decreases in $\rho > 0$ and equals ρ at $\rho > 1.6463$. Therefore, we established a lower bound of at least 1.6463 for every $\rho \leq 1.6463$. ◀

References

1 Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In *Proceedings of the 32nd Annual European Symposium on Algorithms (ESA)*, pages 5:1–5:15, 2024. doi:10.4230/LIPIcs.ESA.2024.5.

2 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 639–650, 2000. doi:10.1007/3-540-46541-3_53.

3 Giorgio Ausiello, Vincenzo Bonifaci, and Luigi Laura. The on-line asymmetric traveling salesman problem. *Journal of Discrete Algorithms*, 6(2):290–298, 2008. doi:10.1016/J.JDA.2007.03.002.

4 Giorgio Ausiello, Marc Demange, Luigi Laura, and Vangelis Th. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92(2):89–94, 2004. doi:10.1016/J.IPL.2004.06.013.

5 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001. doi:10.1007/S004530010071.

6 Júlia Baligács, Yann Disser, Irene Heinrich, and Pascal Schweitzer. Exploration of graphs with excluded minors. *Journal of Computer and System Sciences*, 156:103725, 2026. doi:10.1016/J.JCSS.2025.103725.

7 Júlia Baligács, Yann Disser, Farehe Soheil, and David Weckbecker. Tight analysis of the lazy algorithm for open online dial-a-ride. In *Proceedings of the 18th International Algorithms and Data Structures Symposium (WADS)*, pages 43–64, 2023. doi:10.1007/978-3-031-38906-1_4.

8 Alexander Birx. *Competitive analysis of the online dial-a-ride problem*. PhD thesis, TU Darmstadt, 2020.

9 Alexander Birx and Yann Disser. Tight analysis of the smartstart algorithm for online dial-a-ride on the line. *SIAM Journal on Discrete Mathematics*, 34(2):1409–1443, 2020. doi:10.1137/19M1268513.

10 Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. *Algorithmica*, 2022. doi:10.1007/S00453-022-01061-4.

11 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online TSP on the line. *ACM Transactions on Algorithms*, 17(1):3:1–3:58, 2021. doi:10.1145/3422362.

12 Michiel Blom, Sven O. Krumke, Willem E. de Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001. doi:10.1287/IJOC.13.2.138.10517.

13 Vincenzo Bonifaci and Leen Stougie. Online k-server routing problems. *Theory of Computing Systems*, 45(3):470–485, 2008. doi:10.1007/S00224-008-9103-4.

14 George Christodoulou and Alkmini Sgouritsa. An improved upper bound for the universal TSP on the grid. *SIAM J. Comput.*, 53(5):1476–1523, 2024. doi:10.1137/17M1154849.

15 Romain Cosson. Breaking the $k/\log k$ barrier in collective tree exploration via tree-mining. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4264–4282, 2024. doi:10.1137/1.9781611977912.148.

16 Yann Disser, Jan Hackfeld, and Max Klimm. Tight bounds for undirected graph exploration with pebbles and multiple agents. *Journal of the ACM*, 66(6):40(41), 2019. doi:10.1145/3356883.

17 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001. doi:10.1016/S0304-3975(00)00261-9.

18 Patrick Jaillet and Xin Lu. Online traveling salesman problems with service flexibility. *Networks*, 58(2):137–146, 2011. doi:10.1002/NET.20454.

19 Patrick Jaillet and Xin Lu. Online traveling salesman problems with rejection options. *Networks*, 64(2):84–95, 2014. doi:10.1002/NET.21559.

20 Vinay A. Jawgal, V. N. Muralidhara, and P. S. Srinivasan. Online travelling salesman problem on a circle. In *In Proceedings of the 15th International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 325–336, 2019. doi:10.1007/978-3-030-14812-6_20.

21 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 386–395, 2005. doi:10.1145/1060590.1060649.

22 Bala Kalyanasundaram and Kirk R. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138, 1994. doi:10.1016/0304-3975(94)90155-4.

23 Sven O. Krumke. Online optimization competitive analysis and beyond. Habilitation thesis, Zuse Institute Berlin, 2001.

24 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *Proceedings of the 3rd International Conference on Approximation and Online Algorithms (WAOA)*, pages 258–269, 2006. doi:10.1007/11671411_20.

25 Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness helps: An O(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 200–214, 2002. doi:10.1007/3-540-45753-4_18.

26 Maarten Lipmann. *On-line routing*. PhD thesis, Technische Universiteit Eindhoven, 2003.

Arranging Pairwise Disjoint Shapes to Partition Point Sets

Robert D. Barish[1]
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, Japan

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

Abstract

We consider the fine-grained complexity of covering a set of n points $\mathcal{P}$ in the Euclidean plane using a fixed set of geometric objects corresponding to rigid-body translations and, where permitted, rotations of a specified shape Υ. Under the Exponential Time Hypothesis (ETH), and both with and without a pairwise disjointness constraint, we establish that no $2^{o(\sqrt{n})}$-time algorithm can exist for this problem in the following cases: (case 1) translatable unit disks; (case 2) translatable fixed-area axis-aligned squares; or (case 3) translatable and rotatable fixed-area equilateral triangles. Furthermore, by way of establishing the #P-completeness under parsimonious reductions of positive 1-in-3-SAT with a cubic planar 3-connected clause-variable incidence graph – pertinent to hardness reductions for counting tilings {(Moore & Robson; *Discrete Comput. Geom.* **26**(4); 2001), (Pak & Yang; *J. Comb. Theory. Ser. A* **120**(7); 2013)} – we establish in each case that there exists a quadratic time reduction from #SAT to counting the possible coverage-induced partitions of $\mathcal{P}$. Finally, we consider constraints on the density of the points in $\mathcal{P}$ that make our coverage problems tractable. In particular, letting Υ be any (not necessarily connected) subregion of a radius $\frac{1}{2}$ disk characterized by a semi-algebraic function, and letting $\mathcal{P}$ be a set of n points, we consider the density requirements that: (constraint 1) every 3 points have a minimum bounding disk of radius greater than 1; or (constraint 2) any 5 points have a minimum bounding disk of radius at least 2. Here, when Υ is part of the input, under both (constraint 1) and (constraint 2), and with and without a pairwise disjointness requirement, we show that finding a minimum cardinality set of translatable and/or rotatable instances of Υ covering all points in $\mathcal{P}$ is fixed-parameter tractable in the size of the semi-algebraic description of Υ.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Problems, reductions and completeness

Keywords and phrases geometric covering, geometric packing, clustering, ply, bounded ply, planar geometry, frequency assignment problem, Exponential Time Hypothesis (ETH), Counting Exponential Time Hypothesis (#ETH)

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.5

Funding *Robert D. Barish*: JSPS Kakenhi grant 25K21149.
Tetsuo Shibuya: JSPS Kakenhi grants {23K28035, 23K18501, 21H05052}.

1 Introduction

One of the more elegant existence theorems in plane geometry is a circa 2008 result of Inaba [34, 35] that any arrangement of $n \leq 10$ points in $\mathbb{R}^2$ can be covered by pairwise interior-disjoint unit disks. This can be seen by noting that unit disks pack with an optimal density of $\rho = \frac{\pi}{2\sqrt{3}}$ in a hexagonal lattice arrangement [46], by using the union bound (i.e.,

[1] Corresponding author.

© Robert D. Barish and Tetsuo Shibuya;
licensed under Creative Commons License CC-BY 4.0

Boole's inequality) to determine the probability that n points in the plane are covered by a randomly rotated and translated hexagonal lattice packing of unit disks is at least $(1 - n \cdot (1 - \rho))$, and finally by observing that $(1 - n \cdot (1 - \rho)) > 0$ if $n < \frac{6}{(6-\sqrt{3}\pi)} \approx 10.7411$ (see, e.g., [39]). In the years since, this bound has been shown to hold for $n \leq 12$ [2], and the upper bound for the number of points in $\mathbb{R}^2$ that cannot always be covered by pairwise interior-disjoint unit disks, originally set at 60 due to a triangular lattice construction of Winkler [49], has fallen from 55 [21], to 53 [39], and most recently to 45 [2]. For many of these proofs, the authors either implicitly or explicitly assume the unit disks to be open.

Inaba's result [34, 35] immediately raises the question of the computational complexity of deciding whether pairwise interior-disjoint (i.e., non-overlapping) or pairwise disjoint unit disks can cover some specified set of points in the plane. It also beckons us beyond recreational mathematics into much deeper waters, where we encounter the frequency assignment problem [1, 15, 20, 29] – a well-known applied combinatorial optimization challenge, in which the objective is to assign a scarce and expensive set of well-separated frequencies (often the subject of government-led auctions [13, 32]) to a set of radio stations in order to avoid interference between any two signals. Here, we can interpret the points we are trying to cover with unit disks in Inaba's puzzle as the intended recipients of a signal emitted somewhere along the radio frequency spectrum (e.g., $\approx 10^6$ Hz for AM radio to $\approx 10^9$ Hz for modern Wi-Fi), and the disks used to cover the points as the broadcast ranges of radio frequency emitters with omnidirectional antennas. In this context, a coverage of the point set with pairwise disjoint unit disks becomes an explicit scheme for solving the special case of the frequency assignment problem where we have a finite number of points to cover and only one available broadcast frequency.

While we are unable to directly solve Inaba's puzzle generalized to $n \geq 10$ points, in this work we attempt to make progress towards its resolution by establishing the hardness of covering a specified set of points $\mathcal{P}$ in $\mathbb{R}^2$ with a fixed number of individually translatable and possibly rotatable geometric objects. More specifically, we concern ourselves with the problem of deciding the existence of and counting the following types of partitions:

▶ **Definition 1.** *2D Geometric Cover Partition (2D-GCP). Let $\mathcal{P}$ be a finite set of points in $\mathbb{R}^2$, and let S be a finite collection of geometric objects embedded in the plane. A 2D Geometric Cover Partition (2D-GCP) induced by S is a partitioning of $\mathcal{P}$ into subsets $\{X_1, \ldots, X_k\}$ (where $X_i \cap X_j = \emptyset$ if $i \neq j$) under the constraint that each subset $X_i \subseteq \mathcal{P}$ is the set of all points covered by the same geometric object $s \in S$.*

▶ **Definition 2.** *Pairwise Disjoint 2D Geometric Cover Partition (PD-2D-GCP). An instance of a 2D-GCP where all geometric objects in S are required to be pairwise disjoint (i.e., having no points in common even along their boundaries).*

We clarify that, per Definition 1, each point in $\mathcal{P}$ must belong to the closed set of points for exactly one geometric object in S. Furthermore, when we refer to a 2D-GCP induced by a set of geometric shapes S, we clarify that we are specifically and only referring to the partition of a point set $\mathcal{P}$ into subsets $\{X_1, \ldots, X_k\}$ for some $k \in \mathbb{N}$.

We remark that the task of finding an inducing set S for a 2D-GCP or PD-2D-GCP consisting of rigid-body isometries of a common geometric object is a natural generalization of the Minimum Size 1-Ply Set Cover of Unit Squares (MS1P-SC-US) problem, very recently introduced and proven to be NP-hard by Govindarajan et al. [27], where the objective is to find a minimum set of axis-aligned unit squares covering a set of points $\mathcal{P}$ under the constraint that the geometric system has ply-at-most-1. In particular, we can determine that the NP-hardness proof for MS1P-SC-US given by Govindarajan et al. [27] still holds under

a pairwise disjointness requirement. Extensive precedent for 2D-GCPs and PD-2D-GCPs also comes from the broader literature on geometric packing and coverage problems [3, 4, 9, 6, 8, 11, 18, 19, 22, 23, 24, 26, 25, 28, 31, 48]. Although a full review of this literature is outside the scope of the current work due to space constraints, we briefly note an interesting connection to point set labeling and cartography problems [23, 25, 48], where, for instance, the objective may be to place non-overlapping geometric markers on a map that uniquely label individual features of interest. Finally, we remark that, in contrast to the search or decision problems for 2D-GCPs and PD-2D-GCPs, there appears to be little precedent for the problem of counting these types of partitions.

2 Paper overview

After providing some brief preliminaries and clarifications in Section 3, we lay a foundation for our fine-grained complexity analysis in Section 4 (see Theorem 1 and Corollary 1) by giving a quadratic time parsimonious reduction from #SAT to #CPP-3-Conn-1-in-3-SAT (see Section 3.1 concerning parsimonious reductions and Section 3.4 concerning #CPP-3-Conn-1-in-3-SAT). We briefly remark that this result is of independent interest in tiling theory, as it strengthens an earlier finding of Pak and Yang [40, 41] by establishing the #P-completeness under parsimonious reductions of counting tilings of finite simply connected regions using at most 10^6 types of rectangles having integral edge lengths (see Corollary 2). This concomitantly establishes the completeness of the tiling problem for Ueda & Nagao's Another Solution Problem (ASP) complexity class [42, 47, 50], implying that the problem of finding a second tiling given an initial one is as difficult as finding a second solution for SAT given an initial satisfying assignment. We briefly remark that our proof of the #P-completeness of #CPP-3-Conn-1-in-3-SAT under parsimonious reductions also answers an open question in the literature (see, e.g., [30]) concerning the hardness of 1-in-3-SAT instances with planar 3-connected clause-variable incidence graphs.

Subsequently, letting $\mathcal{P}$ be a set of n points embedded in $\mathbb{R}^2$, we consider the fine-grained complexity of finding and counting 2D-GCPs or PD-2D-GCPs in cases where the set S of geometric objects used to cover $\mathcal{P}$ corresponds to: (case 1) unit disks that can be individually translated as desired; (case 2) fixed-area axis-aligned squares that can be individually translated as desired; and (case 3) fixed-area equilateral triangles that can be individually translated and rotated as desired. In particular, via Theorem 2 and Corollary 3, under the ETH and #ETH, we rule out the existence of a $2^{o(\sqrt{n})}$-time algorithm in each of these cases for finding and counting 2D-GCPs or PD-2D-GCPs.

Finally, in Section 5 we investigate density constraints on $\mathcal{P}$ that allow for tractable instances of the more general problem of covering $\mathcal{P}$ with a minimum cardinality set of translatable and/or rotatable instances of a semi-algebraic subset of a disk. Specifically, we consider two types of density constraints for $\mathcal{P}$: (constraint 1) that any 3 points have a minimum bounding disk of radius greater than 1, or (constraint 2) that any 5 points have a minimum bounding disk of radius at least 2 (where we earlier establish a bound in Theorem 2 for when (constraint 2) can lead to a polynomial-time algorithm under the ETH). Here, letting Υ be a subregion of a radius $\frac{1}{2}$ disk (where we can allow for translations and/or rotations of Υ) admitting a description as a semi-algebraic set, and letting n be the cardinality of $\mathcal{P}$, we show that this coverage problem – both with and without a pairwise disjointness constraint – is fixed-parameter tractable as a function of the size of the description of Υ.

3 Preliminaries and clarifications

3.1 Complexity theoretic notions and terminology

Letting $\phi, \psi : \Sigma^* \to \mathbb{N}$ (where we usually have that $\Sigma = \{0, 1\}$) be a pair of integer counting problems, and letting x be a string specifying an instance of ϕ, a many-one counting reduction from ϕ to ψ is a pair of polynomial time computable functions $f : \Sigma^* \to \Sigma^*$ and $h : \Sigma^* \times \mathbb{N} \to \mathbb{N}$ such that $\phi(x) = h(x, \psi(f(x)))$. Here, a *parsimonious* reduction is a many-one counting reduction where h is simply a projection onto its second argument. Recall that a problem is *fixed-parameter tractable* with respect to a parameter k, if for some computable function $f(k)$, letting x be a string specifying a problem instance, we have that the time complexity for the problem can be expressed as $f(k) \cdot poly(|x|)$.

Unless otherwise stated, we assume the standard word RAM model, where geometric primitives are encoded as bounded-precision rationals (or exact algebraic extensions where necessitated, e.g., for equilateral triangles) and rotations are restricted to exact algebraic extensions of the rationals. In Section 5, we additionally allow symbolic algebraic computation over the reals for intermediate algebraic operations.

3.2 Graph theoretic notions and terminology

All graphs in this work should be assumed to be undirected and simple (meaning free of loops and multi-edges). While we refer the reader to Bondy & Murty [10] or Diestel [17] concerning basic graph theoretic terminology and notions, briefly recall that a *geometric intersection graph* G for a set S of geometric objects embedded in the plane constructs a vertex set V having a bijection with S, and connects a pair of vertices $v_a, v_b \in V$ if and only if their associated geometric objects in S have a point in common.

3.3 Plane geometry notions and terminology

Recall that a *unit disk* is a disk of radius 1 embedded in the plane, that a *unit line segment* is a line segment of length 1 embedded in the plane, and that a *fixed-area axis-aligned square* is a square embedded in the plane in such a manner that its vertex coordinates are given by the points $\{(x, y), (x + k, y), (x + k, y + k), (x, y + k)\}$ for some $(x, y) \in \mathbb{R}^2$ and some specified $k \in \mathbb{N}_{>0}$. When we say that a geometric object *covers* a point p in the plane, unless otherwise specified, we mean that p belongs to the closed set of points defining the geometric object. In the context of this work, all motions of objects should be assumed to be *rigid-body transformations* which preserve the Euclidean distances between all pairs of points belonging to the object, and due to our focus on achiral shapes, we concern ourselves only with rigid transformations corresponding to *translations* and *rotations*. When we say that geometric objects are *pairwise disjoint*, we mean that no two objects have a point in common, regardless of whether any such common point falls internal to or along the boundary of one or both objects. Here, the *ply* [37] of a collection of geometric objects corresponds to the maximum number of objects sharing a common totally interior point.

3.4 Cubic Planar Positive 3-Connected 1-in-3 Satisfiability (CPP-3-Conn-1-in-3-SAT)

Letting ϕ be a Conjunctive Normal Form (CNF) formula (i.e., a conjunction of disjunctions) having exactly three literals per clause (where any literal can be either positive or negative), recall that the 1-in-3-SAT decision problem for input ϕ is the task of deciding whether ϕ admits a satisfying assignment in which each clause contains exactly one "True" literal and exactly two "False" literals. Recall also that the *clause-variable incidence graph* for ϕ is

the bipartite graph $G_\phi = (X \cup Y, E)$, where X (respectively, Y) is in bijection with the variables (respectively, clauses) for ϕ, and where we add an edge between a vertex $v \in X$ and $u \in Y$ if and only if the variable in ϕ corresponding to v occurs as a (positive or negative) literal in the clause in ϕ corresponding to u. We can now define the *Cubic Planar Positive* 3*-Connected* 1*-in-*3 *Satisfiability (CPP-*3*-Conn-*1*-in-*3*-SAT)* problem as a special case of 1-in-3-SAT in which the clause-variable incidence graph for any input formula ϕ is restricted to being a cubic (i.e., 3-regular) planar 3-connected graph, and where all literals in ϕ are positive. We also define #CPP-3-Conn-1-in-3-SAT as the #P problem of counting all satisfying assignments for instances of CPP-3-Conn-1-in-3-SAT. Finally, we define *Cubic Planar Positive* 1*-in-*3 *Satisfiability (CPP-*1*-in-*3*-SAT)* (respectively, #CPP-1-in-3-SAT) as a relaxation of CPP-3-Conn-1-in-3-SAT (respectively, #CPP-3-Conn-1-in-3-SAT) where we do not require the clause-variable incidence graph to be 3-connected.

3.5 Exponential Time Hypothesis (ETH) and Counting Exponential Time Hypothesis (#ETH)

The Exponential Time Hypothesis (ETH) of Impagliazzo & Paturi [33] is the conjecture that an n variable instance of 3-SAT ϕ cannot be solved in $2^{o(n)}$ time. In this context, using the Sparsification Lemma [33], we can assume that ϕ has at most $\mathcal{O}(n)$ clauses. The *Counting Exponential Time Hypothesis (#ETH)* of Dell et al. [16] is a relaxation of the ETH where we only assume that no $2^{o(n)}$-time algorithm exists for the #3-SAT problem of counting all solutions for a given 3-SAT formula ϕ. Here, the Sparsification Lemma can be adapted [16] to again allow us to assume that ϕ has at most $\mathcal{O}(n)$ clauses.

3.6 Orthogonal integer lattice embeddings

Letting H be a graph of maximum vertex degree ≤ 4 having vertex set V_H and edge set E_H, an *orthogonal integer lattice embedding* $\mathcal{Q}$ of H is an embedding (or "drawing") of H on a $\mathbb{Z}^2$ grid graph $\mathcal{G}$. To construct $\mathcal{Q}$, we map each vertex $v_i \in V_H$ to a distinct vertex in $\mathcal{G}$, where we can then say that this lattice point in $\mathcal{G}$ *hosts* or is *hosting* $v_i \in V_H$. Next, for every edge $v_a \leftrightarrow v_b \in E_H$, we create a non-self-intersecting *polyline* composed of end-to-end adjacent horizontal and vertical unit line segments falling along the edges of $\mathcal{G}$ and corresponding to a simple path between the lattice points hosting vertices $v_a, v_b \in V_H$ (note that the polyline must not otherwise intersect a vertex in $\mathcal{G}$ hosting a vertex in H). Here, the meeting of any horizontal and vertical unit line segments common to the same polyline is referred to as a *bend* in the embedding. As a clarification, polyline-polyline crossings are permitted, though we may refer to $\mathcal{Q}$ as a *planar orthogonal integer lattice embedding* if no such crossings occur. In this context, we define a *cell* in $\mathcal{Q}$ as a 1×1 square centered on a point $(x, y) \in \mathbb{Z}^2$ with vertex coordinates $\left(\left(x - \frac{1}{2}, y - \frac{1}{2}\right), \left(x + \frac{1}{2}, y - \frac{1}{2}\right), \left(x + \frac{1}{2}, y + \frac{1}{2}\right), \left(x - \frac{1}{2}, y + \frac{1}{2}\right)\right)$. Note that when we identify a cell at a coordinate $(x, y) \in \mathbb{Z}^2$ with some set of points S (e.g., as we do with various gadgets in the Theorem 2 proof argument), each point $p_i \in S$ should be assumed to be translated to the coordinate $p_i + (x, y)$. Finally, for some $k \in \mathbb{N}_{>0}$, we define a *k-fold enlargement* of $\mathcal{Q}$ as the embedding generated by moving all lattice points (x, y) hosting either vertices of an embedded graph or polyline segment endpoints to the coordinate $(k \cdot x, k \cdot y)$, then reconnecting polylines by adding horizontal and vertical unit line segments without the introduction of additional bends.

For the purposes of visual intuition, in Fig. 1(b) we give an explicit example of an orthogonal integer lattice embedding of the 3-cube from Fig. 1(a) on a 6×6 grid. Here, embedded vertices from Fig. 1(a) retain their original colorations in Fig. 1(b), (thick **black**) edges are used to indicate polylines, and a 1×1 cell is indicated using a (thick dashed **purple**) box.

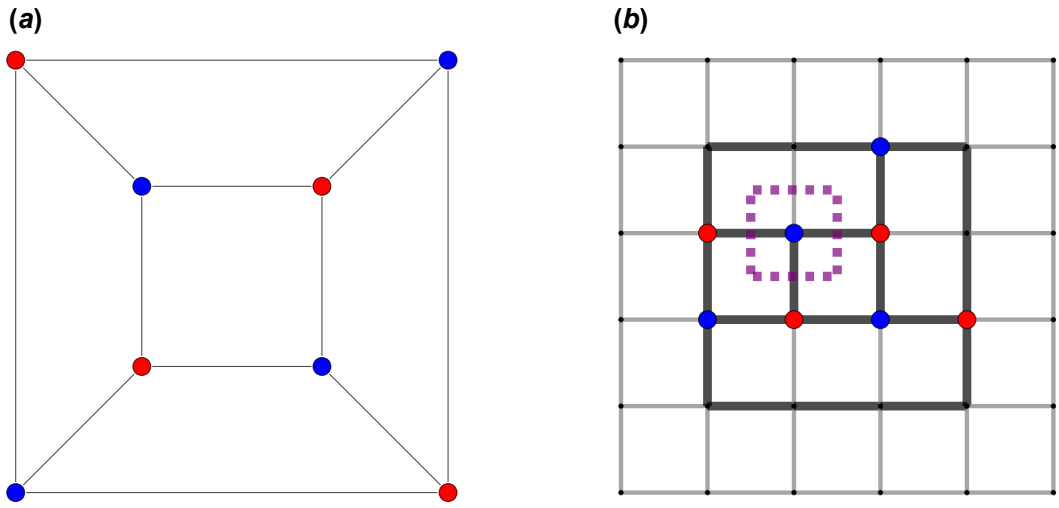


Figure 1 Example orthogonal integer lattice embedding; **(a)** planar embedding of the 3-cube, where vertices are colored (**blue**) and (**red**) to indicate their partite set membership; **(b)** orthogonal integer lattice embedding of the 3-cube from (a) in a 6×6 integer lattice, where the embedded vertices retain their coloration from (a), (thick **black**) edges belong to polylines, and the (thick dashed **purple**) box indicates an example 1×1 cell in the embedding.

3.7 Algebraic and semi-algebraic sets

An *algebraic set* is a subset of an affine space that can be represented as the locus of zeroes (i.e., zero-set) of a finite collection of polynomial equations. A *semi-algebraic set* is a subset of an affine space represented as the union, intersection, and/or complement of finite collections of polynomial equations and inequalities. In this work, we will only consider semi-algebraic sets in $\mathbb{R}^2$.

4 Covering points in the plane with pairwise disjoint geometric objects

In this section, we first lay a foundation for our fine-grained analysis by giving a parsimonious reduction from #SAT to #CPP-3-Conn-1-in-3-SAT via Theorem 1 and Corollary 1. In the following Corollary 2, we briefly address an implication of Theorem 1 in tiling theory.

▶ **Theorem 1.** *There exists an $\mathcal{O}(m)$-time parsimonious reduction from any Planar 1-in-3-SAT formula with a 2-connected clause-variable incidence graph and $m \in \mathbb{N}_{>0}$ clauses to a CPP-3-Conn-1-in-3-SAT formula.*

Proof. We proceed via a gadget-based reduction, where letting ϕ be an m clause instance of Planar 1-in-3-SAT with a 2-connected clause-variable incidence graph G_ϕ, we will construct the clause-variable incidence graph $H_{\phi'}$ for an $\mathcal{O}(m)$ clause instance ϕ' of CPP-3-Conn-1-in-3-SAT. Specifically, the gadgets shown in Fig. 2(a–d) will be connected together via vertex identification operations between what we refer to as *pole vertex pairs*, corresponding to gadget vertices with labels of the form $\{p_{(r,1)}, p_{(r,2)}\}$ for some $r \in \mathbb{N}_{>0}$. Here, for some $r, s \in \mathbb{N}_{>0}$, we identify pole vertex pairs $\{p_{(r,1)}, p_{(r,2)}\}$ and $\{p_{(s,1)}, p_{(s,2)}\}$ by identifying the degree 2 vertex $p_{(r,1)}$ with the degree 1 vertex $p_{(s,2)}$, and then identifying the degree 2 vertex $p_{(s,1)}$ with the degree 1 vertex $p_{(r,2)}$. We clarify that a pole vertex pair will be referred to as *unused* if it has not yet participated in such a vertex identification operation. We now build $H_{\phi'}$ via the following 6 steps:

- (step 1) For every degree $k = 3$ variable vertex v_i in G_ϕ, we construct an instance α_i of the "3-Output CPP-3-Conn-1-in-3-SAT variable gadget" shown in Fig. 2(a).
- (step 2) For every clause vertex u_i in G_ϕ, we construct an instance β_i of the "CPP-3-Conn-1-in-3-SAT clause gadget" shown in Fig. 2(b).
- (step 3) For every edge e_i corresponding to a negated literal in G_ϕ, we construct an instance γ_i of the "CPP-3-Conn-1-in-3-SAT inverter gadget" shown in Fig. 2(c).
- (step 4) For every degree $k = 2$ or $k \geq 4$ variable vertex v_i in G_ϕ, we construct instances $\{q_1, \ldots, q_k\}$ of the "3-Output CPP-3-Conn-1-in-3-SAT variable gadget" shown in Fig. 2(a), then – as illustrated in Fig. 2(d) – we identify an unused pole vertex pair on each gadget q_j with an unused pole vertex pair on the gadget $q_{((j \bmod k)+1)}$, yielding a "k-Output CPP-3-Conn-1-in-3-SAT variable gadget" δ_i with k unused pole vertex pairs.
- (step 5) For every edge between a variable vertex v_i and a clause vertex u_j in G_ϕ, where the variable corresponding to v_i occurs as a positive literal in the clause corresponding to u_j, we identify an unused pole vertex pair on α_i (or δ_i if v_i has degree 2 or degree ≥ 4) with an unused pole vertex pair on β_j.
- (step 6) For every edge e_k between a variable vertex v_i and a clause vertex u_j in G_ϕ, where the variable corresponding to v_i occurs as a negative literal in the clause corresponding to u_j, we identify unused pole vertex pairs on the gadgets α_i (or δ_i if v_i has degree 2 or degree ≥ 4) and β_j with unused pole vertex pairs on the gadget γ_k.

For the resulting graph $H_{\phi'}$, each pole vertex pair will correspond to a set of variable vertices in $H_{\phi'}$ that, due to the properties of the gadgets in Fig. 2(a–d), will necessarily have the same "True" or "False" assignment in any witness for ϕ', and thus serve as a means of specifying inputs to gadgets that simulate clauses and variables in the input instance ϕ of Planar 1-in-3-SAT. Accordingly, to see that the 1-in-3-SAT formula ϕ' corresponding to the clause-variable incidence graph $H_{\phi'}$ represents a formula logically equivalent to ϕ, it suffices to make the following three observations concerning the 1-in-3-SAT formulas induced by the gadgets used to construct $H_{\phi'}$: (obs. 1) there are exactly two feasible satisfying assignments for each formula induced by a "3-Output CPP-3-Conn-1-in-3-SAT variable gadget" or "k-Output CPP-3-Conn-1-in-3-SAT variable gadget", where one assignment sets all pole vertex pairs to "False" and the other sets all pole vertex pairs to "True" (simulating a variable vertex in G_ϕ); (obs. 2) there are exactly three feasible satisfying assignments for each formula induced by a "CPP-3-Conn-1-in-3-SAT clause gadget", where each assignment sets exactly one distinct pole vertex pair to "True" and all other pole vertex pairs to "False" (simulating a clause vertex in G_ϕ); and (obs. 3) there are exactly two feasible satisfying assignments for formulas induced by the "CPP-3-Conn-1-in-3-SAT inverter gadget", where each assignment sets exactly one of the two distinct pole vertex pairs to "True" (simulating an edge corresponding to a negated literal vertex in G_ϕ).

It remains to observe that $H_{\phi'}$ will necessarily be a cubic planar 3-connected graph corresponding to the clause-variable incidence graph of a CPP-3-Conn-1-in-3-SAT formula, and that (step 1) through (step 6) take $\mathcal{O}(m)$ time. ◀

▶ **Corollary 1.** *There exists an $\mathcal{O}(m^2)$-time parsimonious reduction from any 3-SAT formula with $m \in \mathbb{N}_{>0}$ clauses to a CPP-3-Conn-1-in-3-SAT formula with $\mathcal{O}(m^2)$ clauses.*

Proof. Observe first that we can parsimoniously reduce any m clause instance ψ of 3-SAT to an $\mathcal{O}(m)$-clause instance ϕ of 3-SAT with a 2-connected clause-variable incidence graph in $\mathcal{O}(m)$ time by: (1) constructing two instances ψ_1 and ψ_2 of ψ; (2) for every variable x_i in ψ_1 corresponding to a variable y_i in ψ_2, constructing the formula $Z_i = (x_i \vee \neg y_i) \wedge (\neg x_i \vee y_i)$; and (3) parsimoniously reducing the formula $\psi_1 \wedge \psi_2 \wedge \bigcup_i (Z_i)$ to a

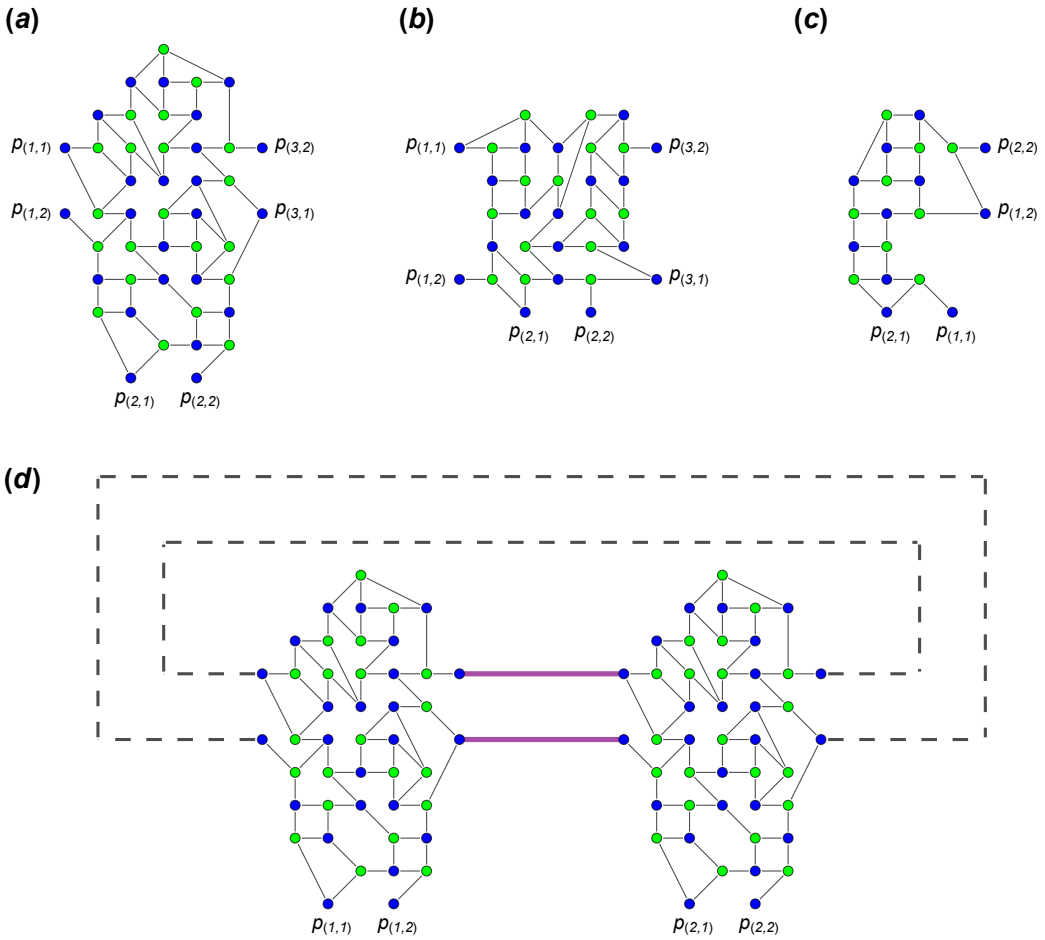


Figure 2 Gadgets used in reducing Planar 1-in-3-SAT with a 2-connected clause-variable incidence graph to CPP-3-Conn-1-in-3-SAT, where (green) vertices correspond to 1-in-3-SAT clauses and (blue) vertices correspond to variables; **(a)** "3-Output CPP-3-Conn-1-in-3-SAT variable gadget"; **(b)** "CPP-3-Conn-1-in-3-SAT clause gadget"; **(c)** "CPP-3-Conn-1-in-3-SAT inverter gadget"; **(d)** scheme for constructing a "k-Output CPP-3-Conn-1-in-3-SAT variable gadget" for arbitrary $k \in \mathbb{N}_{>1}$, where (purple) edges connect vertices to be identified via a pole vertex pair identification operation, and where (black dashed) edges indicate a gadget cycle containing k total copies of the "3-Output CPP-3-Conn-1-in-3-SAT variable gadget".

3-SAT formula ϕ via standard methods. Next, recall that the reduction of Lichtenstein [36] can be used to parsimoniously reduce the $\mathcal{O}(m)$ clause instance ϕ of 3-SAT to an $m' \in \mathcal{O}\left(m^2\right)$ clause instance ϕ' of Planar 3-SAT in $\mathcal{O}\left(m^2\right)$ time, and moreover, ensure that the clause-variable incidence graph of ϕ' is 2-connected if the clause-variable incidence graph of ϕ is 2-connected. Thus, by Theorem 1, it now suffices to show the existence of an $\mathcal{O}\left(m'\right)$-time reduction from ϕ' to an instance ϕ'' of Planar 1-in-3-SAT with a 2-connected clause-variable incidence graph. To proceed, let f be a 1-in-3-SAT clause function which accepts a set of 3 literals and returns "True" if and only if exactly one literal in the set is "True", and observe that the 3-SAT clause $(p_1 \vee p_2 \vee p_3)$ is logically equivalent to the 1-in-3-SAT formula $f\left(p_1, x_1, x_2\right) \wedge f\left(p_2, x_2, x_3\right) \wedge f\left(\neg p_3, x_2, x_4\right) \wedge f\left(x_1, x_3, x_5\right)$. Observe now that, after applying this reduction to an input Planar 3-SAT formula ϕ', the clause-variable incidence graph for the resulting Planar 1-in-3-SAT formula ϕ'' will be planar and 2-connected if the clause-variable incidence graph for ϕ' is planar and 2-connected. ◀

▶ **Corollary 2.** *The following problems are #P-complete under parsimonious reductions: (P1) #CPP-3-Conn-1-in-3-SAT; (P2) counting tilings of finite simply connected regions using at most* 10^6 *types of rectangles with integral edge lengths.*

Proof. Observe that (P1) and (P2) are trivially in #P. The claim for (P1) follows directly from Theorem 1 and Corollary 1. The claim for (P2) follows directly from a parsimonious reduction from #CPP-1-in-3-SAT to counting the stated type and number of rectangular tilings given in "Theorem 1.1" of Pak and Yang [40, 41]. ◀

We now proceed to establish Theorem 2 and Corollary 3.

▶ **Theorem 2.** *Letting* n *be the number of variables for an instance* $\#\phi$ *of #CPP-3-Conn-1-in-3-SAT, and letting* $\mathcal{P}$ *be a set of* $\mathcal{O}\left(n^2\right)$ *points embedded in* $\mathbb{R}^2$*, there exists an* $\mathcal{O}\left(n^2\right)$*-time parsimonious reduction from* $\#\phi$ *to counting either 2D-GCPs (or equivalently in the contexts we consider, PD-2D-GCPs) induced by a fixed number* k*, where* $k \in \mathcal{O}\left(n^2\right)$*, geometric objects corresponding to:*

- *(case 1) unit disks that can be individually translated;*
- *(case 2) fixed-area axis-aligned squares that can be individually translated;*
- *(case 3) fixed-area equilateral triangles that can be individually translated and rotated;*
- *(case 4) disks of radius* $\frac{6}{5}$ *under the additional constraint that any* 5 *points in* $\mathcal{P}$ *have a minimum bounding disk of radius at least* 2*.*

Proof. We proceed by first treating (case 1) through (case 3), and then treating (case 4) by establishing its equivalence to (case 1) for the point sets we consider.

To begin, we consider an instance $\#\phi$ of #CPP-3-Conn-1-in-3-SAT with n variables, $m \in \mathcal{O}(n)$ clauses, and a cubic planar bipartite clause-variable incidence graph G_ϕ, where V_{clause} and V_{var} are the partite sets of vertices in G_ϕ corresponding to clauses and variables, respectively. Using the algorithm of Biedl & Kant [7], we compute an orthogonal integer lattice embedding $\mathcal{Q}_1$ of G_ϕ having total area $\mathcal{O}\left(n^2\right)$ in $\mathcal{O}(n)$ time, and let $\mathcal{Q}_2$ be a 3-fold enlargement of $\mathcal{Q}_1$ (see Section 3.6 for an elaboration).

Next, we construct a set of points $\mathcal{P} \subset \mathbb{R}^2$ from the cells in $\mathcal{Q}_2$, such that both 2D-GCPs and PD-2D-GCPs for $\mathcal{P}$ in (case 1) through (case 3) will be in bijection with the satisfying assignments being counted by $\#\phi$. Specifically, we begin by generating a set of points $\Omega \subset \mathbb{R}^2$ via the following steps:

- (step 1) we identify every cell hosting a vertex $v \in V_{clause}$ with the "1-in-3-SAT clause gadget" having point set $\{(-1,0), \left(-\frac{3}{4},0\right), \left(-\frac{59}{100},-\frac{7}{40}\right), \left(-\frac{7}{20},-\frac{7}{40}\right), (0,0), \left(0,\frac{3}{4}\right), (0,1), \left(\frac{7}{40},\frac{7}{20}\right), \left(\frac{7}{40},\frac{59}{100}\right), \left(\frac{7}{20},-\frac{7}{40}\right), \left(\frac{59}{100},-\frac{7}{40}\right), \left(\frac{3}{4},0\right), (1,0)\}$ – corresponding to the graph shown in Fig. 3(a.1) – rotated to ensure that points on the cell boundary fall along all outgoing polylines.
- (step 2) We identify every cell hosting a vertex $v \in V_{var}$ with the "variable gadget" having point set $\{ (-1,0), \left(-\frac{3}{4},0\right), \left(-\frac{9}{16},0\right), \left(-\frac{1}{4},0\right), \left(0,\frac{1}{4}\right), \left(0,\frac{9}{16}\right), \left(0,\frac{3}{4}\right), (0,1), \left(\frac{1}{4},0\right), \left(\frac{9}{16},0\right), \left(\frac{3}{4},0\right), (1,0)\}$ – corresponding to the graph shown in Fig. 3(b.1) – rotated to ensure that points on the cell boundary fall along all outgoing polylines.
- (step 3) For every cell not hosting a vertex $v \in V_{clause} \cup V_{var}$ and incident to two horizontal or two vertical polylines, we identify the cell with the "horizontal / vertical connector gadget" having point set $\{(-1,0), \left(-\frac{3}{4},0\right), \left(-\frac{1}{2},0\right), \left(-\frac{1}{4},0\right), (0,0), \left(\frac{1}{4},0\right), \left(\frac{1}{2},0\right), \left(\frac{3}{4},0\right), (1,0)\}$ – corresponding to the graph shown in Fig. 3(c.1) – rotated to ensure that points on the cell boundary fall along all outgoing polylines.

- (step 4) For every cell not hosting a vertex $v \in V_{clause} \cup V_{var}$ and incident to a horizontal and a vertical polyline, we identify the cell with the "bend connector gadget" having point set $\{(0,-1), (0,-\frac{3}{4}), (\frac{1}{10},-\frac{113}{200}), (\frac{87}{400},-\frac{151}{400}), (\frac{151}{400},-\frac{87}{400}), (\frac{113}{200},-\frac{1}{16}), (\frac{3}{4},0), (1,0)\}$ – corresponding to the graph shown in Fig. 3(d.1) – rotated to ensure that points on the cell boundary fall along all outgoing polylines.
- (step 5) For every collection ζ of cells identified with a "horizontal / vertical connector gadget" or "bend connector gadget" and corresponding to a common polyline in $\mathcal{Q}_2$, to ensure that $(|\zeta| - 2) \equiv 0 \pmod 3$ we replace at most 2 of the 9 point "horizontal / vertical connector gadgets" (which will necessarily exist since $\mathcal{Q}_2$ is a 3-fold enlargement of $\mathcal{Q}_1$) with the 10 point "+1-shift horizontal / vertical connector gadget" having point set $\{(-1,0), (-\frac{3}{4},0), (-\frac{5}{8},0), (-\frac{3}{8},0), (-\frac{1}{8},0), (\frac{1}{8},0), (\frac{3}{8},0), (\frac{5}{8},0), (\frac{3}{4},0), (1,0)\}$ – corresponding to the graph shown in Fig. 3(e.1) – rotated to ensure that points on the cell boundary fall along all outgoing polylines.

We now let $\mathcal{P}$ be the set of points generated by moving every point $(x,y) \in \Omega$ to the coordinate $(4 \cdot x, 4 \cdot y)$, and make the following observations:

- (obs. 1) No set of 4 points in $\mathcal{P}$ can be covered by a unit disk in (case 1), a single fixed-area axis-aligned square in (case 2), or a fixed-area equilateral triangle in (case 3).
- (obs. 2) Letting φ be the number of point sets in $\mathcal{P}$ corresponding to a 4-fold enlargement of point sets in Ω arising from a "1-in-3-SAT clause gadget" instance, there must be at least the same number φ of geometric objects in (case 1) through (case 3) covering ≤ 2 distinct points each. Furthermore, if there are exactly φ instances of geometric objects in (case 1) through (case 3) covering ≤ 2 distinct points each, then each of these geometric objects must contain some translation – due to a cell identification operation when constructing Ω, and the subsequent 4-fold enlargement of Ω when generating $\mathcal{P}$ – of a point originally at coordinate $(0,0)$ in the "1-in-3-SAT clause gadget".
- (obs. 3) If we require that all but φ geometric objects from (case 1) through (case 3) cover three distinct points each in $\mathcal{P}$, then the only possible manner (up to the uniqueness of the induced 2D-GCP or PD-2D-GCP) of covering the point sets in $\mathcal{P}$ belonging to distinct instances (i.e., not falling along cell boundaries) of the "1-in-3-SAT clause gadget", "variable gadget", "horizontal / vertical connector gadget", "bend connector gadget", or "+1-shift horizontal / vertical connector gadget" are shown in Fig. 3(a.2–a.4), Fig. 3(b.2–b.3), Fig. 3(c.2–c.4), Fig. 3(d.2–d.4), and Fig. 3(e.2–e.4), respectively. Here, unit disks for (case 1) are colored (**blue**), fixed-area axis-aligned squares are drawn as (**black** hollow squares), and fixed-area equilateral triangles correspond to (**red** hollow triangles). Briefly, under a constraint that all but φ geometric objects cover 3 points each, it will otherwise be necessary in each case for some pair of geometric objects to cover the same point, where an example of this for the "variable gadget" is illustrated in Fig. 3(b.4) with the two (**gray**) disks sharing a common point in $\mathcal{P}$.
- (obs. 4) Due to (step 5) of constructing Ω, we can interpret the unit disk (case 1), fixed-area axis-aligned square (case 2), and fixed-area equilateral triangle (case 3) covers for the "1-in-3-SAT clause gadget" in Fig. 3(a.2), Fig. 3(a.3), and Fig. 3(a.4) as corresponding to clauses with literals evaluating to {"True", "False", "False"}, {"False", "True", "False"}, and {"False", "False", "True"}, respectively. In this context, the output of "True" is propagated along polylines entering the cell for the gadget from the North in Fig. 3(a.2), West in Fig. 3(a.3), and East in Fig. 3(a.4). Correspondingly, we can interpret the unit disk (case 1), fixed-area axis-aligned square (case 2), and fixed-area equilateral triangle (case 3) covers for the "variable gadget" in Fig. 3(b.2) and Fig. 3(b.3) as variable assignments of "True" and "False", respectively.

Putting (obs. 1) through (obs. 4) together, in (case 1) through (case 3) we can determine that each CPP-3-Conn-1-in-3-SAT satisfying assignment being counted by $\#\phi$ will correspond to the same number k of geometric objects inducing a specific 2D-GCP or PD-2D-GCP, and furthermore, that $\geq k+1$ geometric objects will be required to cover $\mathcal{P}$ if no satisfying assignment exists. In (case 1) through (case 3) we moreover have a bijection between the satisfying assignments counted by $\#\phi$ and 2D-GCPs, as well as a bijection between the satisfying assignments counted by $\#\phi$ and PD-2D-GCPs for the same sets of geometric objects, as our reduction ensures these objects will be pairwise disjoint.

It remains to address (case 4). Here, we can determine that the minimum bounding disk for any 5 points from the point sets in $\mathcal{P}$ will have radius at least $\frac{5}{3}$. Since $\left(\frac{5}{3}\right)\cdot\left(\frac{6}{5}\right)=2$, we can accordingly move each point $(x,y)\in\mathcal{P}$ to the coordinate $\left(\frac{6\cdot x}{5},\frac{6\cdot y}{5}\right)$ to create a new point set $\mathcal{P}'$ where every 5 points have a minimum bounding disk of radius at least 2. Finally, as we are proportionally increasing the radius of the unit disk from (case 1) to $\frac{6}{5}$, we can observe that (case 4) is equivalent to (case 1) in the context of our construction. ◀

▶ **Corollary 3.** *Letting $\mathcal{P}$ be a set of n points embedded in $\mathbb{R}^2$, letting Z be the set of 2D-GCPs (or equivalently in the contexts we consider, PD-2D-GCPs) of $\mathcal{P}$ induced by $k\in\mathcal{O}(n)$ geometric objects in (case 1) through (case 4) of Theorem 2, and under the assumption that at most 3 points can be covered by a geometric object, the following holds: (1) it is NP-hard and #P-hard under parsimonious reductions to decide whether $|Z|\geq 1$ and to determine $|Z|$, respectively; (2) no $2^{o(\sqrt{n})}$-time algorithm exists for deciding whether $|Z|\geq 1$ and for determining $|Z|$ under the ETH and #ETH, respectively.*

Proof. First, given the polynomial time reduction from #CPP-3-Conn-1-in-3-SAT to counting 2D-GCPs (or equivalently in the contexts we consider, PD-2D-GCPs), in (case 1) through (case 4) of Theorem 2, the fact that CPP-1-in-3-SAT is NP-complete [38] establishes the NP-hardness of deciding whether $|Z|\geq 1$, and Theorem 1 together with Corollary 1 establish that #CPP-3-Conn-1-in-3-SAT is #P-hard under parsimonious reductions. Next, establishing that no $2^{o(\sqrt{n})}$-time algorithm exists for deciding whether $|Z|\geq 1$ and determining $|Z|$ under the ETH and #ETH, respectively, requires a bit more care. Naively, letting $n=q^2$, it would appear that we require $\mathcal{O}(q^4)$ time to transitively parsimoniously reduce a q variable and $\mathcal{O}(q)$ clause instance of #3-SAT to counting 2D-GCPs or to counting PD-2D-GCPs, as there is a quadratic overhead to reduce #3-SAT to #Planar-3-SAT via the reduction of Lichtenstein [36], and then another quadratic overhead in computing an orthogonal integer lattice embedding when reducing #Planar-3-SAT to counting 2D-GCPs or PD-2D-GCPs via the method given in the Theorem 2 proof argument. However, we can nevertheless achieve the required $\mathcal{O}\left(q^2\right)$ overhead by making use of the fact that the orthogonal integer lattice embedding algorithm of Biedl & Kant [7] can compute, in $\mathcal{O}\left(q\right)$ time, both $\mathcal{O}\left(q^2\right)$ area non-planar orthogonal integer lattice embeddings (i.e., embeddings with crossing polylines) and $\mathcal{O}\left(q^2\right)$ area embeddings of planar graphs with a specified outer face.

Specifically, we can: (step 1) follow exactly the Theorem 1 and Corollary 1 reductions to reduce an input 3-SAT formula ϕ to a 1-in-3-SAT (not necessarily planar) formula ϕ' having a cubic clause-variable incidence graph with clauses containing only positive literals; (step 2) compute an orthogonal integer lattice embedding of ϕ' using the algorithm of Biedl & Kant [7]; (step 3) use the Theorem 1 and Corollary 1 reduction to modify the 3-SAT to Planar-3-SAT crossing gadget of Lichtenstein [36] to correspond to a CPP-3-Conn-1-in-3-SAT formula, then compute an orthogonal integer lattice embedding of this gadget having the same outer face as in Lichtenstein's original embedding [36]; and (step 4) for some constant factor enlargement of the clause-variable incidence graph for ϕ', use the gadget constructed in (step 3) to planarize the embedding from (step 2) while preserving its orthogonality. Here,

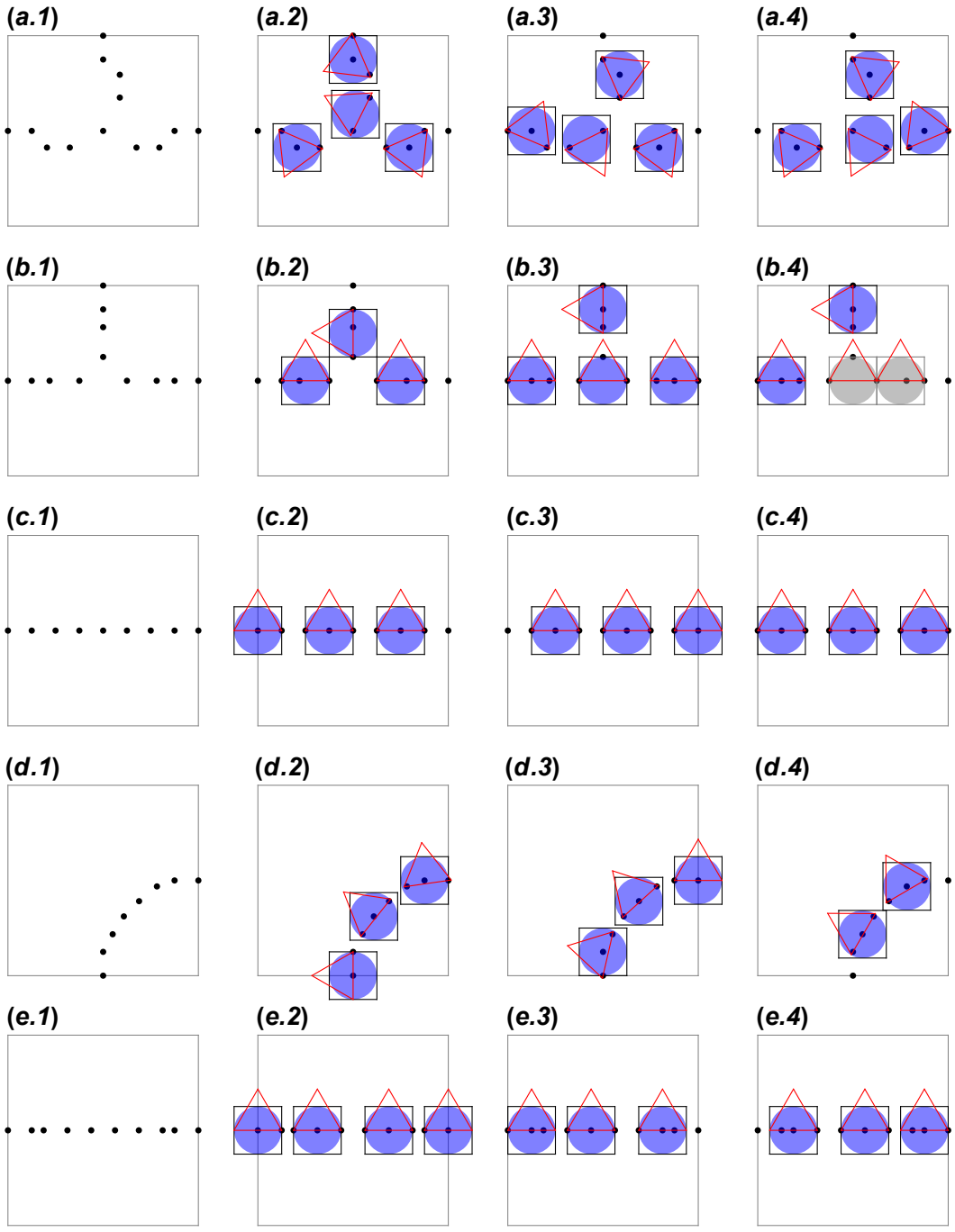


Figure 3 Illustrations for the Theorem 2 parsimonious reduction from #CPP-3-Conn-1-in-3-SAT to counting 2D-GCPs (or equivalently in the contexts we consider, PD-2D-GCPs) induced by translatable unit disks (**blue** or **gray** disks), translatable fixed-area axis-aligned squares (**black** hollow squares), or translatable and rotatable fixed-area equilateral triangles (**red** hollow triangles); **(a.1)**–**(a.4)** illustrations for the "1-in-3-SAT clause gadget" point set; **(b.1)**–**(b.4)** illustrations for the "variable gadget" point set; **(c.1)**–**(c.4)** illustrations for the "horizontal / vertical connector gadget" point set; **(d.1)**–**(d.4)** illustrations for the "bend connector gadget" point set; **(e.1)**–**(e.4)** illustrations for the "+1-shift horizontal / vertical connector gadget" point set; see the proof argument for Theorem 2 for further details.

(step 1) will take $\mathcal{O}(q)$ time, (step 2) will take $\mathcal{O}(q^2)$ time, (step 3) will take $\mathcal{O}(1)$ time, and finally, (step 4) will take at most $\mathcal{O}(q^2)$ time as there will be at most $\mathcal{O}(q^2)$ possible crossing polylines that require the placement of the gadget built in (step 3). Putting everything together, we can now rule out the existence of a $2^{o(\sqrt{n})}$-time algorithm for deciding whether $|Z| \geq 1$ and for determining $|Z|$ under the ETH and #ETH, respectively. ◀

5 Covering sparse sets of points with semi-algebraic subsets of a disk

In this section, we investigate density constraints on point sets $\mathcal{P}$ in $\mathbb{R}^2$ that allow for efficient algorithms to cover the points in $\mathcal{P}$ – with or without a pairwise disjointness constraint – using a minimum cardinality set of translatable and/or rotatable instances of a semi-algebraic subregion Υ of a disk (see Theorem 3). Due to the technical nature of the lemmas in this section (i.e., Lemma 1 through Lemma 3), as opposed to reading straight through, we recommend referring to them as they arise in the context of the proof argument for Theorem 3.

▶ **Lemma 1.** *Letting S be a set of n unit disks embedded in $\mathbb{R}^2$, under the assumption that at most a fixed number of disks in S can have center points in a minimum bounding disk of some fixed radius ≥ 1, the geometric intersection graph induced by S can be computed in $\mathcal{O}(n)$ time.*

Proof. Let $G = (V, E)$ be the geometric intersection graph induced by S, and observe that any requirement at most $k \in \mathbb{N}$ unit disk centers have a minimum bounding disk of some radius $r \in \mathbb{R}_{\geq 1}$ necessarily implies that $|E| \in \mathcal{O}(|V|)$. The current lemma now follows from the fact that we can construct a geometric intersection graph of unit disks having $\mathcal{O}(n)$ edges in $\mathcal{O}(n)$ time via a rounding and hash table approach [5]. ◀

▶ **Lemma 2.** *Let $\mathcal{P}$ be any finite set of points in $\mathbb{R}^2$, and let $\Upsilon \subset \mathbb{R}^2$ be a region admitting a description as a semi-algebraic set. In addition, let S be the set of polynomial inequalities and/or equalities specifying $\mathcal{P}$ and some fixed number of instances of Υ, let ϑ be the number of quantified variables in S, and let d be the maximum degree of any polynomial in S. With or without a pairwise disjointness constraint, there exists a constructive $\mathcal{O}\left((|S| \cdot d)^{2^{\vartheta}}\right)$-time algorithm for finding individual translations and/or rotations of each instance of Υ covering all points in $\mathcal{P}$ or deciding that this is impossible.*

Proof. Let (z_x, z_y) be a translation vector, let $\theta \in [0, 2\pi]$ be a rotation angle, let $R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix}$ be a rotation matrix with $\alpha, \beta \in \mathbb{R}$ satisfying the constraint that $\alpha^2 + \beta^2 = 1$, and let q be a point in a subregion $\Upsilon \subset \mathbb{R}^2$ admitting a description as a semi-algebraic set. Observe that we can express any translation and/or rotation of Υ as the function $f_{(z_x, z_y, \alpha, \beta)}(q) \coloneqq R(\theta)q + (z_x, z_y)$. Also observe that we can express the problem of translating and/or rotating Υ to cover a set of points $\mathcal{P}$ as a quantifier elimination problem for the first-order sentence over the reals "$\exists z_x, z_y, \alpha, \beta \left(\alpha^2 + \beta^2 = 1 \ \wedge \ \forall p \in \mathcal{P} \left(p \in f_{(z_x, z_y, \alpha, \beta)}(\Upsilon)\right)\right)$". This now generalizes in a straightforward manner to instances where we have at most a fixed number of individually translatable and/or rotatable instances of Υ, and where we can also choose to enforce a pairwise disjointness constraint by expressing in first-order logic that no point belongs to the closed set of points for any two translated and/or rotated instances of Υ.

We now appeal to Collins's (constructive) Cylindrical Algebraic Decomposition (CAD) algorithm for real quantifier elimination [12], which improves upon the original (also constructive) method of Tarski [44, 45]. Letting S, ϑ, and d be as stated, to establish the current lemma it remains to observe that Collins's algorithm [12] has time complexity $\mathcal{O}\left((|S| \cdot d)^{2^{\vartheta}}\right)$ [14]. ◀

▶ **Lemma 3.** *Letting G be a connected geometric intersection graph of at most 4 unit disks, the center points of the disks can always be covered by a pairwise interior-disjoint arrangement of 3 unit disks such that at least one unit disk is tangent to the other two.*

Proof. Let $\mathcal{P}$ be the set of center points for the unit disks inducing G. The case where $|\mathcal{P}| = 1$ is trivial. For cases where $2 \leq |\mathcal{P}| \leq 3$, place a unit disk at the center of one edge to cover a pair of points, and if necessary, use a second unit disk tangent to the first to cover the third point, which must always be possible as G is connected. In the case of 4 distinct points, $p_a, p_b, p_c, p_d \in \mathcal{P}$, assume without loss of generality that p_a and p_b induce a pair of adjacent vertices in G, and use a single unit disk D_{red} to cover both p_a and p_b. If D_{red} also covers p_c or p_d, then we can proceed as in the cases where $|\mathcal{P}| \leq 3$. Alternatively, if neither p_c nor p_d are covered by D_{red}, then we can attempt to cover p_c (respectively, p_d) with unit disks D_{blue} (respectively, D_{green}), such that D_{blue} and D_{green} are tangent to D_{red} and have no totally interior point in common.

Now assume that none of the aforementioned cases hold. To proceed, assume without loss of generality that D_{red} (illustrated by the (red) unit disk in Fig. 4(a)) has its center point at the coordinate $(0, 0)$, fixed in position by p_a at the coordinate $(-1, 0)$ and p_b at the coordinate $(1, 0)$ (illustrated by the diameter line between points $p_1 = p_a$ and $p_2 = p_b$ in Fig. 4(a)), and assume that p_c is further from p_a than p_b. Now cover p_c with a unit disk D_{blue} (illustrated by the (blue) disk in Fig. 4(a)), doing so in such a manner as to embed the center point of D_{blue} as far as possible from p_d while ensuring D_{red} and D_{blue} are tangent. Finally, embed a unit disk D_{green} (illustrated by the (green) disk in Fig. 4(a)) such that it is tangent to both D_{red} and D_{blue}, and has a center point as close as possible to p_d. Here, recalling that we are assuming no three points are interior to a common unit disk, we can observe that p_c will fall along the circumference of D_{blue} bordering the "hole" (i.e., the circular triangle) formed by the three mutually tangent disks, and p_d must be an interior point in this hole. However, as illustrated in Fig. 4(a), letting p_3 be the tangency point of D_{red} and D_{green} and p_4 be the tangency point of D_{blue} and D_{green}, we can observe that $\{p_2, p_3, p_4\}$ will form a triangle with side lengths of 1, 2, and at most $\sqrt{3}$. As this triangle fits in a unit disk and contains the points $\{p_b, p_c, p_d\}$, this contradicts our earlier assumption that no three points can be covered by a single unit disk, yielding the current lemma. ◀

▶ **Theorem 3.** *For any finite set of n points $\mathcal{P} \subset \mathbb{R}^2$, letting Υ be any semi-algebraic subset of a radius $\frac{1}{2}$ disk, under either of the following assumptions –*

- *(constraint 1) any 3 points have a minimum bounding disk of radius greater than 1;*
- *(constraint 2) any 5 points have a minimum bounding disk of radius at least 2;*

– it is fixed-parameter tractable in the size of the semi-algebraic description of Υ to find a minimum cardinality set (or determine that no such set exists) of translatable and rotatable instances of Υ covering all points in $\mathcal{P}$ both with and without a pairwise disjointness constraint.

Proof. Let $G = (V, E)$ be an order n geometric intersection graph of radius $\frac{1}{2}$ disks, each with a center at a distinct point in $\mathcal{P}$. Observe that, under both (constraint 1) and (constraint 2), Lemma 1 guarantees G can be constructed in $\mathcal{O}(n)$ time. Also observe that, because Υ is a subset of a radius $\frac{1}{2}$ disk, Υ can only cover a collection of points in $\mathcal{P}$ inducing a connected component in G.

To first treat the case of (constraint 1), as a result of a straightforward triangle inequality argument, observe that this assumption ensures all connected components in G have order at most 2. The assumption also ensures that a radius $\frac{1}{2}$ disk covering a pair of distinct points $p_a, p_b \in \mathcal{P}$ cannot have a point in common with another radius $\frac{1}{2}$ disk covering a

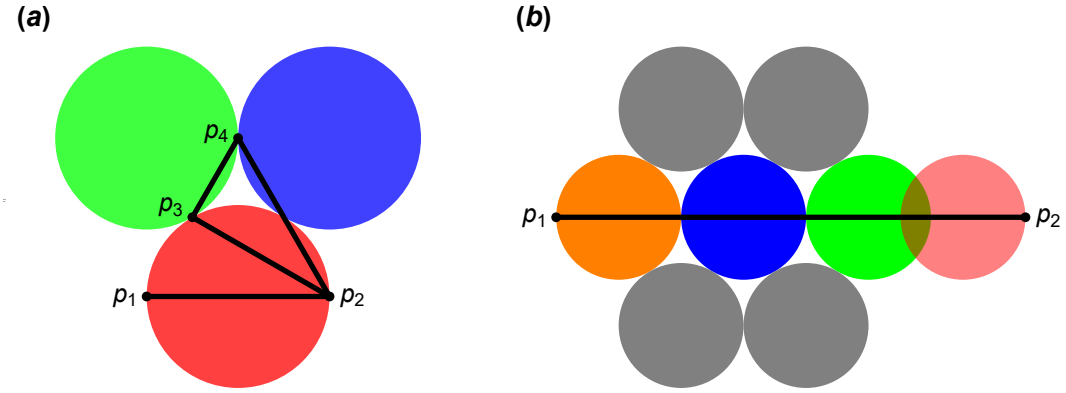


Figure 4 Illustrations to accompany the **(a)** Lemma 3 and **(b)** Theorem 3 proof arguments.

point $p_c \in \mathcal{P} \setminus \{p_a, p_b\}$. Accordingly, no method of covering the points in one order-at-most-2 connected component can prohibit a method of covering the points in another order-at-most-2 connected component. Thus, we have reduced our task to computing an optimal cover of at most n order-at-most-2 connected components using at most 2 translated and/or rotated instances of Υ, which, by Lemma 2, is fixed-parameter tractable in the size of the semi-algebraic description of Υ. This yields the required fixed-parameter tractable algorithm in the case of (constraint 1).

Concerning now (constraint 2), first observe that, under this assumption, any set of 5 points in $\mathcal{P}$ inducing a connected path subgraph H of G will have a maximum bounding disk of radius 2, with no larger radius possible, if and only if these points are embedded at unit intervals along a line. Here, the if direction is trivial to verify, and the only if direction follows from the uniqueness of the equality case for the triangle inequality. As any other embedding of H implies the existence of a minimum bounding disk of radius less than 2, this implies that any connected component in G of order at least 5 will correspond to points embedded at unit intervals along a line.

Next, observe that, under (constraint 2), Lemma 3 guarantees that any set of points in $\mathcal{P}$ inducing an order-at-most-4 connected component in G can always be covered using at most 3 radius $\frac{1}{2}$ disks (i.e., $\frac{1}{2}$ the radius of the original disks from Lemma 3) having no totally interior points in common, and where at least one of these disks is tangent to the other two. As a consequence, no method of covering the points in one order-at-most-4 connected component can prohibit a method of covering the points in another order-at-most-4 connected component. To see this, let D_1, D_2, and D_3 be a set of three radius $\frac{1}{2}$ disks covering some set of points $X \subset \mathcal{P}$ inducing an order-at-most-4 connected component in G, and let D_1 be tangent to D_2 and D_3. Observe that the existence of a radius $\frac{1}{2}$ disk D_4 covering a point $q \notin X$, and having a totally interior point in common with D_1, D_2, or D_3, will necessarily imply that the minimum bounding disk for $X \cup \{q\}$ has radius less than 2. For visual intuition we refer the reader to Fig. 4(b), where D_1 (colored (blue)), D_2 (colored (orange)), and D_3 (colored (green)) have collinear center points, maximizing the possible length of a line segment in their union. Nevertheless, when we embed D_4 (colored (red))

such that it has a totally interior point in common with D_3, the diameter of a minimum bounding disk for the center points of D_1 through D_4 – illustrated as the line between points p_1 and p_2 in Fig. 4(b) – will be less than 4, implying a radius less than 2.

Thus, we have reduced our task to computing an optimal cover by Υ of order-at-most-4 connected components, which Lemma 2 guarantees is fixed-parameter tractable in the size of the semi-algebraic description of Υ, or of points embedded at unit intervals along a line. In the latter case, in combination with Lemma 2, it suffices to observe that a rotation of Υ can cover at most 2 points when embedded at the center of the unit interval between any two points, that a maximum matching for a path graph can be computed in time linear in its order, and that there is sufficient space for each point to be covered by a distinct instance of Υ. This resolves (constraint 2), yielding the current theorem. ◀

6 Concluding remarks

Recall that, on the one hand, (case 4) of Theorem 2 rules out the existence of a sub-exponential algorithm (assuming the ETH) for finding a minimum cardinality set of disks of radius $\frac{6}{5}$ covering a point set $\mathcal{P}$ (with or without a pairwise disjointness constraint) under the additional constraint that any 5 points in $\mathcal{P}$ have a minimum bounding disk of radius at least 2. On the other hand, Theorem 3 tells us that this same problem can be solved in $\mathcal{O}(|\mathcal{P}|)$ time if we use disks of radius $\frac{1}{2}$ to cover the points in $\mathcal{P}$. Accordingly we ask the following question: for a point set $\mathcal{P}$ where any $k \in \mathbb{N}$ points have a minimum bounding disk of radius at least 2 (or some other constant), and letting Υ be some disk of radius $\frac{1}{2} < r < \frac{6}{5}$, when can one efficiently find a minimum number of translated copies of Υ covering all points in $\mathcal{P}$? Furthermore, given that all of the results we report in this work hold with and without a ply constraint, we pose the question of which specific conditions make a pairwise disjointness constraint significant.

References

1 K. I. Aardal, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for frequency assignment problems. *Ann. Oper. Res.*, 153(1):79–129, 2007. doi:10.1007/s10479-007-0178-0.

2 G. Aloupis, R. A. Hearn, H. Iwasawa, and R. Uehara. Covering points with disjoint unit disks. *Proc. 24th Canadian Conference on Computational Geometry (CCCG)*, pages 41–46, 2012.

3 P. Ashok, A. B. Roy, and S. Govindarajan. Local search strikes again: PTAS for variants of geometric covering and packing. *J. Comb. Optim.*, 39:618–635, 2020. doi:10.1007/s10878-019-00432-y.

4 S. Bandyapadhyay, W. Lochet, S. Saurabh, and J. Xue. Minimum-membership geometric set cover, revisited. *Proc. 39th Symposium on Computational Geometry (SoCG)*, pages 11:1–11:14, 2023. doi:10.4230/LIPIcs.SoCG.2023.11.

5 J. L. Bentley, D. F. Stanat, and E. H. Williams Jr. The complexity of finding fixed-radius near neighbors. *Inf. Process. Lett.*, 6(6):209–212, 1977. doi:10.1016/0020-0190(77)90070-9.

6 T. Biedl, A. Biniaz, and A. Lubiw. Minimum ply covering of points with disks and squares. *Comput. Geom.*, 94:101712:1–101712:11, 2021. doi:10.1016/j.comgeo.2020.101712.

7 T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998. doi:10.1016/S0925-7721(97)00026-6.

8 A. Biniaz and Z. Lin. Minimum ply covering of points with convex shapes. *Proc. 32nd Canadian Conference on Computational Geometry (CCCG)*, pages 2–5, 2020.

9 A. Biniaz, P. Liu, A. Maheshwari, and M. Smid. Approximation algorithms for the unit disk cover problem in 2D and 3D. *Comput. Geom.*, 60:8–18, 2017. doi:10.1016/j.comgeo.2016.04.002.

10 J. A. Bondy and U. S. R. Murty. *Graph theory with applications (1st edition)*. Macmillan Press, New York, NY, 1976.

11 J. H. Chun, C. Kipp, and S. Roch. On exact covering with unit disks. *Comput. Geom.*, 129:102193:1–102193:19, 2025. doi:10.1016/j.comgeo.2025.102193.

12 G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. *Proc. Automata theory and formal languages: 2nd GI conference*, pages 134–183, 1975. doi:10.1007/3-540-07407-4_17.

13 P. F. Cortese and M. Patrignani. Clustered planarity = flat clustered planarity. *Proc. 26th International Symposium on Graph Drawing (GD)*, pages 23–38, 2018. doi:10.1007/978-3-030-04414-5_2.

14 J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *J. Symbolic Computation*, 5(1–2):29–35, 1988. doi:10.1016/S0747-7171(88)80004-X.

15 D. de Werra and Y. Gay. Chromatic scheduling and frequency assignment. *Discrete Appl. Math.*, 49(1–3):165–174, 1994. doi:10.1016/0166-218X(94)90207-0.

16 H. Dell, T. Husfeldt, D. Marx, N. Taslaman, and M. Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014. doi:10.1145/2635812.

17 R. Diestel. *Graph theory (5th edition)*. Springer-Verlag, Heidelberg, 2017.

18 S. Durocher, J. M. Keil, and D. Mondal. Minimum ply covering of points with unit squares. *Proc. 17th International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 23–35, 2023. doi:10.1007/978-3-031-27051-2_3.

19 S. Durocher, J. M. Keil, and D. Mondal. Approximation algorithms for minimum ply covering of points with unit squares and unit disks. *Theoret. Comput. Sci.*, 1024:114906:1–114906:14, 2025. doi:10.1016/j.tcs.2024.114906.

20 A. Eisenblätter, M. Grötschel, and A. M. C. A. Koster. Frequency planning and ramifications of coloring. *Discuss. Math. Graph Theory*, 22(1):51–88, 2002.

21 V. Elser. Packing-constrained point coverings. *Geombinatorics Quarterly*, 21:73–80, 2011. arXiv:1101.3468.

22 T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. *Proc. 19th Symposium on Discrete Algorithms (SODA)*, pages 1267–1276, 2008. doi:10.5555/1347082.1347220.

23 M. Formann and F. Wagner. A packing problem with applications to lettering of maps. *Proc. 7th Symposium on Computational Geometry (SoCG)*, pages 281–288, 1991. doi:10.1145/109648.109680.

24 R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

25 S. P. Gomes, G. M. Ribeiro, and L. A. N. Lorena. Dispersion for the point-feature cartographic label placement problem. *Expert Syst. Appl.*, 40(15):5878–5883, 2013. doi:10.1016/j.eswa.2013.04.035.

26 T. F. Gonzalez. Covering a set of points in multidimensional space. *Inf. Process. Lett.*, 40(4):181–188, 1991. doi:10.1016/0020-0190(91)90075-S.

27 S. Govindarajan, M. Patle, and S. Sarkar. Minimum membership geometric set cover in the continuous setting. *Proc. 31st International Conference on Computing and Combinatorics (COCOON)*, pages 262–276, 2025. doi:10.1007/978-981-95-0215-8_20.

28 S. Govindarajan and S. Sarkar. Improved algorithms for minimum-membership geometric set cover. *Proc. 10th Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 103–116, 2024. doi:10.1007/978-3-031-52213-0_8.

29 W. K. Hale. Frequency assignment: theory and applications. *Proc. IEEE*, 68(12):1497–1514, 1980. doi:10.1109/PROC.1980.11899.

30 M. M. Hasan, D. Mondal, and M. S. Rahman. Positive planar satisfiability problems under 3-connectivity constraints. *Theoret. Comput. Sci.*, 917:81–93, 2022. doi:10.1016/j.tcs.2022.03.013.

31 D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

32 G. Illing and U. Klüh, editors. *Spectrum auctions and competition in telecommunications*. The MIT Press: Cambridge, MA, 2003. doi:10.7551/mitpress/6400.001.0001.

33 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

34 N. Inaba. "10 points answer" (translated from Japanese), 2008. URL: http://inabapuzzle.com/hirameki/suuri_ans4.html.

35 N. Inaba. "10 points" (translated from Japanese), 2008. URL: http://inabapuzzle.com/hirameki/suuri_4.html.

36 D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. doi:10.1137/0211025.

37 G. L. Miller, S. H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997. doi:10.1145/256292.256294.

38 C. Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete Comput. Geom.*, 26(4):573–590, 2001. doi:10.1007/s00454-001-0047-6.

39 Y. Okayama, M. Kiyomi, and R. Uehara. On covering of any point configuration by disjoint unit disks. *Proc. 23rd Canadian Conference on Computational Geometry (CCCG)*, pages 1–5, 2011.

40 I. Pak and J. Yang. Tiling simply connected regions with rectangles. *J. Comb. Theory. Ser. A*, 120(7):1804–1816, 2013. doi:10.1016/j.jcta.2013.06.008.

41 I. Pak and J. Yang. Tiling simply connected regions with rectangles, 2013. arXiv:1305.2796v1.

42 T. Seta. The complexities of puzzles, CROSS SUM and their another solution problems (ASP). Senior thesis, The University of Tokyo, Tokyo, Japan, 2002.

43 M. I. Shamos. Computational geometry. *Ph.D. dissertation, Yale University*, pages 1–236, 1978.

44 R. G. Swan. Tarski's principle and the elimination of quantifiers, 2005. Unpublished; available at https://www.math.uchicago.edu/~swan/expo/Tarski.pdf.

45 A. Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, RAND Corporation, Santa Monica, CA, 1948. Available at https://www.rand.org/content/dam/rand/pubs/reports/2008/R109.pdf.

46 L. F. Tóth. Über die dichteste kugellagerung. *Math. Z.*, 48:676–684, 1942. doi:10.1007/BF01180035.

47 N. Ueda and T. Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.

48 M. van Kreveld, T. Strijk, and A. Wolff. Point set labeling with sliding labels. *Proc. 14th Symposium on Computational Geometry (SoCG)*, pages 337–346, 1998. doi:10.1145/276884.276922.

49 P. Winkler. Puzzled: figures on a plane. *Commun. ACM*, 53(8):128, 2010. doi:10.1145/1810891.1810917.

50 T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Trans. Fundamentals*, E86-A(5):1052–1060, 2003. URL: http://search.ieice.org/bin/summary.php?id=e86-a_5_1052.

How Many Slopes Does Polynomial Area Cost?

Michael A. Bekos
University of Ioannina, Greece

Eleni Katsanou
National Technical University of Athens, Greece

Philipp Kindermann
Trier University, Germany

Maria Eleni Pavlidi
University of Ioannina, Greece

Abstract

In this work, we study the interplay between the number of slopes, the number of bends per edge, and the area requirements for planar drawings of bounded-degree graphs. Our motivation stems from the fact that, while numerous algorithms produce planar drawings with few slopes for graphs of relatively small degree in polynomial area, existing approaches for higher-degree graphs often require super-polynomial area. We address this gap in the literature by presenting new constructions that yield polynomial-area drawings with few bends per edge while slightly increasing the required number of slopes, thereby providing the first systematic study of slopes, bends and area trade-offs.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Human-centered computing → Graph drawings

Keywords and phrases k-bend planar drawings, planar slope number, area requirements

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.6

Related Version *Full Version*: https://arxiv.org/abs/2605.31098

Funding This work was partially funded by the IKYDA 2025 funding program of the DAAD (project 57775422 "Algorithms and Combinatorics for Drawing Graphs with Few Slopes").

1 Introduction

Producing drawings of graphs while minimizing the number of slopes is a well-studied problem in Graph Drawing, motivated by both theoretical interest and practical applications such as network visualization and VLSI design [2, 19, 41]. From an algorithmic perspective, given a graph, the goal is to determine its slope number, that is, the minimum number of pairwise distinct slopes used by its edge segments, taken over all polyline drawings of the graph with a prescribed number of bends per edge. Wade and Chu [42] were among the first to investigate this problem by showing that the slope number of the complete graph K_n in the straight-line setting is n. The problem has been extensively studied since then; several results are known for general graphs of bounded degree [15, 26, 34–36], and beyond-planar graphs [11, 28].

When the input graph is planar, the output drawing is additionally required to be crossing-free. The problem of finding planar drawings with few slopes has also been extensively studied [4, 6, 7, 12–14, 18, 25, 29–31, 33], as it traces back to orthogonal graph drawing [8, 17, 40], where edges are represented as polygonal chains consisting of alternating horizontal and vertical segments, that is, using only two slopes. In this context, a central result by Biedl and Kant [5] guarantees that every planar graph of maximum degree 4, except for the octahedron, admits an orthogonal drawing on an $n \times n$ grid in which each edge has at most two bends. This result is tight in the sense that there exist planar graphs of maximum degree 4 that do not admit planar orthogonal drawings with one bend per edge [40].

© Michael A. Bekos, Eleni Katsanou, Philipp Kindermann, and Maria Eleni Pavlidi;
licensed under Creative Commons License CC-BY 4.0

Table 1 Summary of our results for planar graph drawings with at most k bends per edge.

k	Degree	Connectivity	Drawing area	No. of Slopes	Ref.
1	$\Delta \geq 5$	3	$\mathcal{O}(\Delta n^2) \times \mathcal{O}(\Delta n^3)$	$3\Delta - 8$	Thm. 1
1	$\Delta \geq 5$	–	$\mathcal{O}(\Delta n^2) \times \mathcal{O}(\Delta n^3)$	$\lceil \frac{9}{2}\Delta \rceil + 1$	Cor. 2
1	5	3	$\mathcal{O}(n^3) \times \mathcal{O}(n^4)$	5	Thm. 3
2	$\Delta \geq 3$	2	$\mathcal{O}(n) \times \mathcal{O}(\Delta n^2)$	$\lceil \frac{\Delta}{2} \rceil$	Thm. 4
2	Δ	–	$\mathcal{O}(n) \times \mathcal{O}(\Delta n^2)$	$\lceil \frac{\Delta}{2} \rceil + 1$	Cor. 5
4	Δ	–	$\mathcal{O}(n) \times \mathcal{O}(n)$	Δ	Thm. 6

A natural extension of the orthogonal drawing model is the octilinear, which additionally supports diagonal segments at $\pm 45°$, yielding a total of four slopes. In this model, every planar graph with maximum degree at most 3 admits a bendless planar drawing [12] on a $O(n) \times O(n)$ grid, while every planar graph with maximum degree at most 4 (and 5, respectively) admits a planar drawing with at most one bend per edge on a $O(n^2) \times O(n)$ grid (and a super-polynomial grid, respectively) [3].

For graphs of higher degree, Keszegh, Pach, and Pálvölgyi [25] extended the algorithm of Biedl and Kant [5] and showed that every planar graph of maximum degree $\Delta \geq 3$, with the exception of the octahedron, admits a planar drawing with at most two bends per edge using segments of at most $\lceil \Delta/2 \rceil$ distinct equidistant slopes; this bound on the number of slopes is clearly optimal. Improving previous related results [25, 32], Angelini, Bekos, Liotta, and Montecchiani [1] demonstrated that every planar graph of maximum degree $\Delta \geq 4$ admits a planar drawing with at most one bend per edge using segments from any arbitrary set of $\Delta - 1$ pairwise distinct slopes. In the straight-line setting, Keszegh, Pach, and Pálvölgyi [25] showed that every planar graph with maximum degree Δ admits a straight-line drawing using segments of $2^{O(\Delta)}$ distinct slopes.

Our contribution. In this work, we identify and close a critical gap in the literature. While almost all aforementioned algorithms for planar graphs of small fixed degree produce drawings of polynomial area [3,5,12,17,40] (and are therefore practically applicable), the corresponding algorithms that have been proposed for planar graphs of higher maximum degree require super-polynomial area [1, 25, 32], which limits significantly their practical applicability. Thus, in this paper, we study for first time the interplay between the number of slopes, the number of bends per edge, and the corresponding area requirements; see Table 1. Our focus is on algorithms that produce planar drawings with few bends per edge on polynomial-size grids while only slightly increasing the number of slopes used. More precisely:

- In Section 3, we prove that every 3-connected n-vertex planar graph of maximum degree Δ admits a planar grid drawing with at most one bend per edge, using at most $3\Delta - 8$ slopes on a $O(\Delta n^2) \times O(\Delta n^3)$ grid. Our approach builds upon the incremental construction by Angelini, Bekos, Liotta and Montecchiani [1], which produces such drawings on *arbitrary* sets of $\Delta - 1$ slopes. In contrast, we fix the slope set in advance and increase its size from $\Delta - 1$ to $3\Delta - 8$ in order to guarantee polynomial area. As a consequence, for general planar graphs (i.e., not necessarily 3-connected) the number of slopes becomes $\lceil \frac{9}{2}\Delta \rceil + 1$.
- For the special case of planar graphs of maximum degree 5, we decrease the number of required slopes from $3\Delta - 8 = 7$ to 5 at the cost of increasing the drawing area by a factor of $O(n^2)$; see Theorem 3. Compared with the best-known algorithm in [3], our construction achieves polynomial-area while increasing the number of slopes by one.

- In Section 4, we prove that every planar graph G of maximum degree $\Delta \geq 3$ admits a 2-bend planar drawing on a $O(n) \times O(\Delta n^2)$ grid using at most $\lceil \Delta/2 \rceil$ slopes if G is biconnected, and at most $\lceil \Delta/2 \rceil + 1$ slopes otherwise. In contrast to the algorithm by Keszegh, Pach, and Pálvölgyi [25], which uses *equidistant* slopes to support rotations and scalings of biconnected components around cut vertices, our algorithm uses a different slope set to guarantee polynomial-drawing area. However, it requires one additional slope for general (i.e., non-biconnected) planar graphs, as it can rely neither on rotations (because of the non-equidistant slopes) nor on scaling (because of the area requirement).
- In Section 5, we prove that, regardless of the maximum degree Δ of the input planar graph, quadratic area in the number of vertices of the graph suffices to obtain a planar drawing with at most Δ slopes, where each edge has at most four bends. If the graph is additionally subhamiltonian, then three bends suffice.

Note that a fundamental requirement of our grid drawings is that both vertices and edge bends lie on grid points. Furthermore, we stress that the algorithms that we present in this work can be implemented to run in time linear in the size of the input graphs.

2 Preliminaries

In this section, we introduce preliminary definitions and notation used throughout the paper. Unless stated otherwise, all graphs considered are simple and undirected. The *degree* of a vertex is the number of its neighbors. A graph has *maximum degree* Δ if it contains a vertex of degree Δ and no vertex of degree greater than Δ. A graph is *connected* if every pair of vertices is joined by a path. More generally, for $k \geq 1$, a graph is *k-connected* if the removal of any set of at most $k-1$ vertices leaves the graph connected. In particular, 2- and 3-connected graphs are also referred to as *biconnected* and *triconnected*, respectively.

A *drawing* of a graph maps each vertex of the graph to a point of the Euclidean plane and each of its edges to a Jordan arc connecting its endpoints. A drawing is *planar* if no two edges intersect except possibly at common endpoints. Such a drawing partitions the plane into connected regions called *faces*; the unbounded one is the *outer face*. A graph is *planar* if it admits a planar drawing. A *planar embedding* of a planar graph is an equivalence class of planar drawings that define the same set of faces and the same outer face. A planar drawing is *k-bend* if each of its edges is a polygonal chain composed of at most $k+1$ straight-line segments. The point where two such segments meet is called a *bend*. Unless otherwise specified, we consider *grid drawings*, that is, drawings in which each vertex and each bend lies on a point of the Euclidean plane with integer coordinates. Given such a drawing Γ, we denote by $W(\Gamma)$ and by $H(\Gamma)$ the width and the height of the minimum rectangle enclosing Γ, respectively.

The *slope* of a line measures its steepness and direction. It is defined as the ratio of the vertical change (rise) to the horizontal change (run) between any two points on the line. This equivalently corresponds to the tangent of the counter-clockwise angle through which a horizontal line must be rotated to coincide with the given line. The *horizontal* (*vertical*) *slope* is the slope of a line parallel (perpendicular) to the x-axis. The slope of an edge segment is the slope of the line containing it. Given a set of slopes S, a k-bend planar drawing is said to be *on* S if each of its edge segments has a slope belonging to S. For a vertex v in a k-bend planar drawing, each slope s of S determines two distinct rays emanating from v with slope s, which we call *ports*. If s is the horizontal slope, these rays are called *horizontal*. The upward (downward) directed ports are called *top* (*bottom*) ports. We say that a port ρ_v incident to v is *free* if no edge incident to v is drawn along ρ_v; otherwise, ρ_v is *occupied*.

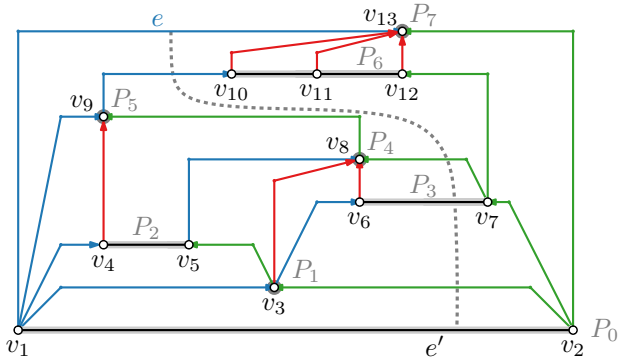


Figure 1 Illustration of a canonical order and of a 4-coloring of a 3-connected planar graph. The dotted curve is a cut (dashed) at edge e.

Given an edge e at the outer face of a k-bend planar drawing Γ, a *cut at edge* e is a strictly y-monotone curve that **(i)** starts at a point on a horizontal segment of e, **(ii)** ends at a point on a horizontal segment of an edge e' incident to the outer face of Γ, with $e' \neq e$, and **(iii)** intersects only horizontal segments of Γ; see Figure 1. Such a cut allows to *stretch* Γ horizontally by translating all vertices and edges on one side of the cut horizontally by an arbitrary distance $d > 0$, thereby increasing the horizontal distance between the two resulting parts without introducing crossings. Since the stretching is purely horizontal, the slopes of all non-horizontal segments remain unchanged, while the lengths of the horizontal segments crossed by the cut increase. Hence, if Γ is on S before the stretching, it remains on S afterward. When we say that we *stretch an edge* e, we refer precisely to this operation.

Let G be a 3-connected n-vertex plane graph and let $\Pi = (P_0, \ldots, P_m)$ be a partition of its vertex set into paths such that $P_0 = \{v_1, v_2\}$, $P_m = \{v_n\}$, the edges (v_1, v_2) and (v_1, v_n) exist and belong to the outer face of G. For $i = 0, \ldots, m$, let G_i be the subgraph induced by $P_0 \cup \ldots \cup P_i$ and denote by C_i the contour of G_k defined as follows: If $i = 0$, then C_0 is the edge (v_1, v_2) of P_0, while if $i > 0$, then C_i is the path from v_1 to v_2 obtained by removing (v_1, v_2) from the cycle delimiting the outer face of G_i. We say that Π is a *canonical order* [9, 21] of G if for each $i = 1, \ldots, m-1$ the following hold (see Figure 1): **(i)** G_i is biconnected, internally 3-connected and embedded with $C_i \cup \{(v_1, v_2)\}$ as its outer face; **(ii)** all neighbors of P_i in G_{i-1} are on C_{i-1}; **(iii)** P_i either consists of a single vertex (called *singleton*), or the degree of each of its vertices is 2 in G_i (called *chain*); **(iv)** every vertex in P_i has at least one neighbor in P_j with $j > i$. A canonical order of a 3-connected planar graph can be computed in linear time [21].

Given a 3-connected plane graph G and a canonical order Π of it, it is possible to compute a 4-edge-coloring of G similar to the one by Schnyder [16, 39]. The edge (v_1, v_2) of G_0 is colored black. For $i = 1, \ldots, m$, a 4-coloring of G_{i-1} is extended to one of G_i as follows (see, e.g, Figures 1 and 3). First, consider the edges of G_i that do not belong to G_{i-1} and lie on the contour C_i. The first (last) such edge encountered on a traversal of C_i from v_1 to v_2 is colored blue (green, respectively), while all remaining ones (i.e., those whose endpoints both belong to P_i, when P_i is a chain) are colored black. The remaining edges of G_i that do not belong to G_{i-1} are colored red; these are precisely the edges incident to P_i that are not part of C_i (i.e., when P_i is a singleton of the canonical order). Finally, we treat all black edges as undirected, and all remaining edges as directed where the orientation of an edge connecting a vertex $u \in P_i$ to a vertex $v \in P_j$ with $0 \leq i < j \leq m$ is from u to v.

For an n-vertex graph with two designated vertices s and t, an *st-ordering* $v_1, \ldots, v_n$ is a permutation of its vertices such that $v_1 = s$ and $v_n = t$, and every vertex v_j with $1 < j < n$ has at least two neighbors v_i and v_k with $i < j < k$. It is known that every biconnected graph admits such an st-ordering [38]. In the case of a planar input, one may additionally guarantee that the vertices v_1, v_2, v_n all lie on the outer face of the graph, such that the edge (v_1, v_2) is incident to this face [5].

In our algorithms, we occasionally augment the input graph to make it biconnected or triconnected. To augment a connected planar graph G to a simply biconnected planar graph G', we utilize an algorithm by Kant and Bodlaender [22]. This algorithm runs in linear time and ensures that the maximum degree $\Delta(G')$ of the resulting graph satisfies $\Delta(G') \leq \Delta(G) + 2$. Furthermore, we utilize an algorithm by Kant [20] to augment a biconnected planar graph G to a triconnected planar graph G'. This augmentation is performed in linear time, and the resulting maximum degree $\Delta(G')$ is bounded by $\Delta(G') \leq \max\{2, \lceil \frac{3}{2}\Delta(G) \rceil\}$.

3 1-bend planar drawings of 3-connected degree-Δ planar graphs

In this section, we seek to prove that every 3-connected n-vertex planar graph with maximum degree Δ admits a 1-bend planar grid drawing with at most $3\Delta - 8$ slopes on a $O(\Delta n^2) \times O(\Delta n^3)$ grid. Since for $\Delta \leq 4$ our results is superseded by [3, 10], we assume w.l.o.g. that $\Delta \geq 5$. Our approach builds upon the incremental construction by Angelini, Bekos, Liotta and Montecchiani [1], which uses a canonical ordering of the input 3-connected planar graph G to produce a 1-bend planar drawing Γ of it on an *arbitrary* set of $\Delta - 1$ slopes. In contrast, we fix the slope set in advance. Moreover, to obtain polynomial area, we enlarge the number of available slopes from $\Delta - 1$ to $3\Delta - 8$.

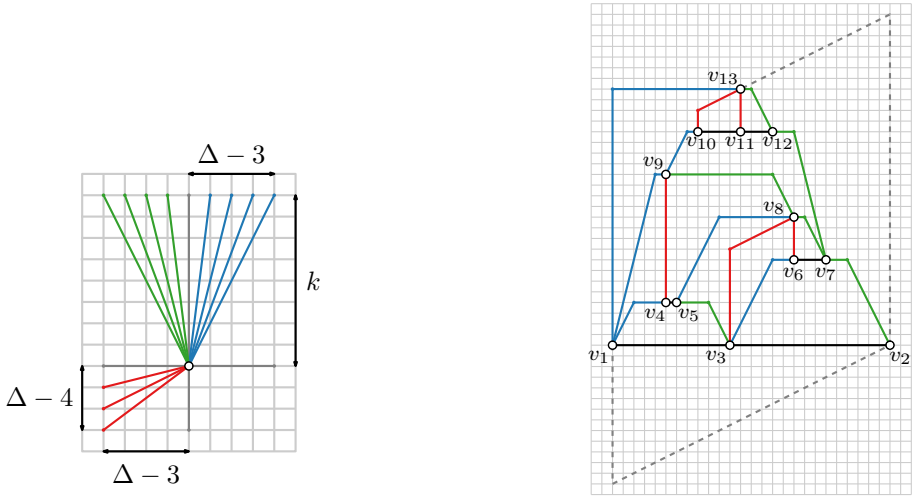
▶ **Theorem 1.** *Every 3-connected planar n-vertex graph G with maximum degree $\Delta \geq 5$ admits a 1-bend planar grid drawing with at most $3\Delta - 8$ slopes on a $12\Delta n^2 \times 18\Delta n^3$ grid.*

Proof. The slope set S used by our algorithm is defined with respect to a parameter $k > \Delta n^2$ that we will specify later, and is the union of the following sets. S_{v} consists only of the vertical slope. S_{h} consists only of the horizontal slope. S_{ls} consists of $\Delta - 3$ *left steep* slopes $-\frac{k}{1}, \ldots, -\frac{k}{\Delta-3}$ (green in Figure 2a). S_{rs} consists of $\Delta - 3$ *right steep* slopes $\frac{k}{1}, \ldots, \frac{k}{\Delta-3}$ (blue in Figure 2a). Finally, S_{f} consists of $\Delta - 4$ *flat* slopes $\frac{1}{\Delta-3}, \ldots, \frac{\Delta-4}{\Delta-3}$ (red in Figure 2a). Hence, the cardinality of S is $3\Delta - 8$, as desired.

Let $\Pi = (P_0, \ldots, P_m)$ be a canonical order of G. For the drawing algorithm, we first remove the edge (v_1, v_2) from the graph and color the edges incident to v_1 and v_2 in G_1 black. After the drawing algorithm is completed, we will insert the edge (v_1, v_2) below the constructed drawing with a vertical segment incident to v_1 and a segment with slope $\frac{1}{\Delta-3} \in S_{\text{f}}$ incident to v_2 (see Figure 2b). For now, we also assume that v_n has degree strictly less than Δ (note that if $\Delta \geq 6$, then this assumption is w.l.o.g. since we can choose v_n such that its degree is at most 5). We will describe how to handle the other case later.

Assume that for $0 < i \leq m$, we have already constructed a 1-bend planar grid drawing Γ_{i-1} of G_{i-1} on S that satisfies the following invariants.

- **I.1** The contour C_{i-1} of Γ_{i-1} is drawn strictly x-monotone.
- **I.2** There exists a cut at every edge belonging to the contour C_{i-1} of Γ_{i-1}.
- **I.3** Every vertex of C_{i-1} has at least as many unoccupied top ports in each of S_{rs} and S_{ls} incident to the outer face of Γ_{i-1} as it has neighbors in $G \setminus G_{i-1}$ minus one, and the port corresponding to S_{v} is unoccupied if the vertex has at least one neighbor in $G \setminus G_{i-1}$ or its degree is strictly less than Δ.
- **I.4** All vertices of G_{i-1} are at y-coordinates that are multiples of k in Γ_{i-1}.



(a) (b)

Figure 2 (a) Illustration of the slopes used in Theorem 1. (b) The drawing created by the algorithm in Theorem 1 with $k = 4$ for the graph in Figure 1.

I.5 Based on their colors, the edges of G_{i-1} have been drawn as follows in Γ_{i-1}:

- **a.** Each black edge of G_{i-1} consists of a single horizontal segment (i.e., its slope is in S_{h}).
- **b.** Each blue edge of G_{i-1} consists of two segments; the one incident to its source has a slope in $S_{\mathrm{v}} \cup S_{\mathrm{rs}}$, while the one incident to its target is in S_{h}.
- **c.** Each red edge of G_{i-1} consists of at most two segments; the one incident to its source is vertical (i.e., its slope is in S_{v}), while the one incident to its target has a slope in $S_{\mathrm{f}} \cup S_{\mathrm{v}}$, i.e, if the slope of the second segment is in S_{v}, then the edge consists of one segment.
- **d.** Each green edge of G_{i-1} consists of two segments; the one incident to its source has a slope in $S_{\mathrm{v}} \cup S_{\mathrm{ls}}$, while the one incident to its target is in S_{h}.

Note that Invariants I.1 and I.2 are inherited from [1]; Invariant I.3 is adapted to our setting, and Invariants I.4 and I.5 are specific to our construction. The base case of our recursive algorithm is the graph G_1, which consists of the vertices v_1, v_2, and all vertices in P_1. We place these vertices on a horizontal line: v_1 is positioned at $(0, 0)$, v_2 at $(|P_1|+1, 0)$, and the vertices of P_1 are placed between them at unit distance. All edges are then drawn as horizontal segments. This obviously satisfies the invariants. So, we may assume that $i > 1$. To derive a drawing Γ_i of G_i maintaining Invariants I.1–I.5, we introduce the vertices of path P_i into the drawing Γ_{i-1} by distinguish two cases depending on whether P_i is a chain or a singleton of degree 2 (Case 1) and a singleton of degree greater than 2 (Case 2). At a high level, Invariant I.4 will allow us to determine a grid point for each vertex of P_i and for each bend of the blue and green edges incident to it, while Invariant I.2 will support stretching the drawing Γ_{i-1} to create additional space to accomplish this placement, if needed.

Case 1: P_i is a chain or a singleton of degree 2. Suppose that $P_i = \{v_g, \ldots, v_h\}$ is a chain or a singleton of degree 2 in G_i. Note that in the latter case, $g = h$ holds. Let v_ℓ and v_r be the neighbors of v_g and v_h in G_{i-1}, respectively. Refer to Figure 3a. W.l.o.g., we may assume that v_ℓ appears before v_r along the contour C_{i-1} in a traversal of it from v_1 to v_2. Let also ρ_ℓ (ρ_r) be the first unoccupied port at v_ℓ (v_r) encountered in a counter-clockwise (clockwise) traversal

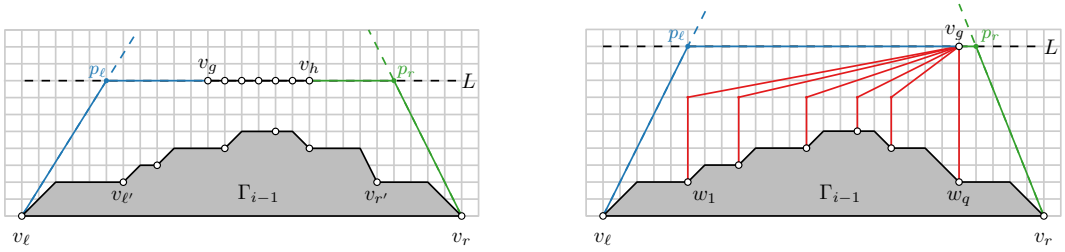


Figure 3 Illustration of the case where P_i is (a) a chain or a singleton of degree 2, and (b) a singleton of degree greater than 2 in Theorem 1.

of its top ports in $S_{\text{rs}} \cup S_{\text{v}}$ ($S_{\text{ls}} \cup S_{\text{v}}$) starting from the rightward (leftward) horizontal port, which exist by Invariant I.3. To satisfy Invariant I.4, we set the y-coordinate of each of $v_g, \ldots, v_h$ to $H(\Gamma_{i-1}) + k$. The x-coordinates of $v_g, \ldots, v_h$ will be determined by the constraints arising from the way that $(v_\ell, v_g), (v_g, v_{g+1}), \ldots, (v_{h-1}, v_h), (v_h, v_r)$ must be drawn. Since (v_ℓ, v_g) and (v_r, v_h) are incoming edges to v_g and v_h in G_i, we satisfy Invariants I.5.b and I.5.d by drawing (v_ℓ, v_g) and (v_r, v_h) with a horizontal segment incident to v_g and v_h and a second segment attached to ρ_ℓ and ρ_r, respectively, which maintains Invariant I.3 for v_ℓ and v_r. The remaining edges of P_i (if any) will be drawn as unit-length horizontal segments satisfying Invariant I.5.a. Each of the vertices $v_g, \ldots, v_h$ has at most $\Delta - 2$ neighbors in $G \setminus G_i$ and all $\Delta - 3$ ports unoccupied in each of S_{rs} and S_{ls} that are incident to the outer face of Γ_i, which ensures Invariant I.3 for them.

Let p_ℓ and p_r be the points where the rays that correspond to ρ_ℓ and ρ_r intersect the horizontal line L through v_g; see Figure 4. We have to ensure that p_ℓ lies at least $h - g + 2$ units to the left of p_r such that $v_g, \ldots, v_h$ can be placed between them in Γ_i. Furthermore, the rays at ρ_ℓ and ρ_r are not allowed to cross any part of Γ_{i-1}. To guarantee these conditions, we horizontally stretch Γ_{i-1} as follows. Let $(v_\ell, v_{\ell'})$ be the first edge of C_{i-1} that is encountered when traversing C_{i-1} from v_ℓ to v_2. Analogously, $(v_r, v_{r'})$ is the first edge of C_{i-1} that is encountered when traversing C_{i-1} from v_r to v_1. (Note that $v_{\ell'} = v_r$ and $v_{r'} = v_\ell$ is possible.) Since both these edges lie on C_{i-1}, by Invariant I.2 there is a cut at each of them. This allows us to stretch $(v_\ell, v_{\ell'})$ until p_ℓ lies to the left of $v_{\ell'}$, and symmetrically, stretch $(v_r, v_{r'})$ until p_r lies to the right of $v_{r'}$. This further guarantees that the horizontal distance between p_ℓ and p_r is at least 2. If needed, we further stretch one of the edges $(v_\ell, v_{\ell'})$ and $(v_{r'}, v_r)$, say w.l.o.g. the former, by up to $h - g$ additional units, so as to ensure that the horizontal distance between p_ℓ and p_r is at least $h - g + 2$. Hence, it is possible to place each of $v_g, \ldots, v_h$ at a grid point along L that lies between p_ℓ and p_r, and draw the edges (v_ℓ, v_g) and (v_r, v_h) with one bend each at p_ℓ and p_r, respectively. The remaining edges of P_i are drawn as unit-length horizontal edge segments, as we initially sought; see Figure 3. This completes the drawing of Γ_i.

It remains to show that Γ_i is planar and satisfies Invariant I.1. For the former, observe that the introduction of $v_g, \ldots, v_h$ into Γ_{i-1} yields $h - g + 4$ new edge segments in Γ_i. The two segments connecting p_ℓ with v_g and p_r with v_h, as well as all edge segments connecting internal vertices of P_i (if any) lie completely above Γ_{i-1}. So they cannot cross any edge of Γ_{i-1}. The port ρ_ℓ that is used by the edge segment connecting v_ℓ with p_ℓ is the next available port in S_{rs}, i.e., the one that follows the corresponding port used by the edge $(v_\ell, v_{\ell'})$. Furthermore, the only edge of C_{i-1} that lies between the x-coordinates of v_ℓ and p_ℓ is the edge $(v_\ell, v_{\ell'})$. So no edge of C_{i-1} and thus no edge of Γ_{i-1} can be crossed by the

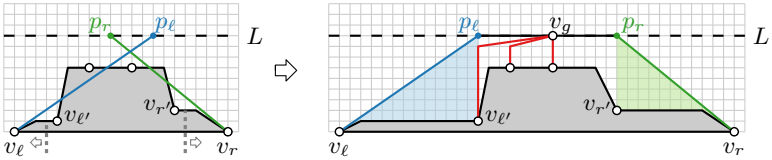


Figure 4 Illustration of using cuts to maintain planarity in Theorem 1.

edge segment connecting v_ℓ and p_ℓ. A symmetric argument applies to the edge segment connecting v_r and p_r. Since these two edge segments cannot cross each other, it follows that Γ_i is planar, as desired. Having ensured this property, the fact that the contour C_i of Γ_i is x-monotone is implied by the choice of ρ_ℓ and ρ_r and by the fact that $\rho_\ell \in S_{\text{rs}}$ and $\rho_r \in S_{\text{ls}}$. Hence, Γ_i satisfies Invariant I.1.

Before considering the case where P_i is a singleton of degree greater than 2 in G_i, we establish an upper bound on how much the drawing Γ_{i-1} must be stretched horizontally to accommodate $v_g, \ldots, v_h$ in Γ_i. To this end, consider the edge (v_ℓ, v_g). Let x_ℓ be the x-coordinate of p_ℓ before applying any stretching. Let also $s_\ell \in S_{\text{rs}}$ be the slope of the edge segment connecting v_ℓ with p_ℓ; symmetrically, $s_r \in S_{\text{ls}}$ is defined. It follows that $x_\ell = x(v_\ell) + \frac{y(v_g)-y(v_\ell)}{s_\ell} \leq x(v_\ell) + \frac{y(v_g)}{s_\ell}$. After the stretching of the edge $(v_\ell, v_{\ell'})$, the x-coordinate of p_ℓ should be at most $x(v_{\ell'}) - 1$. To achieve this, the edge $(v_\ell, v_{\ell'})$ must be stretched by at most $x_\ell - (x(v_{\ell'}) - 1) \leq x(v_\ell) + \frac{y(v_g)}{s_\ell} - x(v_\ell) = \frac{y(v_g)}{s_\ell}$ units of length. Symmetrically, the edge $(v_r, v_{r'})$ must be stretched by at most $\frac{y(v_g)}{|s_r|}$ units of length. Now the x-coordinate of p_ℓ is at most the old x-coordinate of v_ℓ and the x-coordinate of p_r is at least the old x-coordinate of v_r, so p_ℓ and p_r are at least one unit apart. To make space for $v_g, \ldots, v_h$ between them, we might have to stretch the edge $(v_\ell, v_{\ell'})$ by $h - g + 1$ more units. Since $|P_i| = h - g + 1$, it follows that the total stretch applied is at most

$$\frac{y(v_g)}{s_\ell} + \frac{y(v_g)}{|s_r|} + |P_i| \leq \frac{H(\Gamma_{i-1}) + k}{\frac{k}{\Delta-3}} + \frac{H(\Gamma_{i-1}) + k}{\frac{k}{\Delta-3}} + |P_i|$$
$$= 2 \cdot \frac{H(\Gamma_{i-1}) + k}{k} \cdot (\Delta - 3) + |P_i| \leq 2\Delta \cdot \frac{H(\Gamma_{i-1}) + k}{k} + |P_i|. \quad (1)$$

Case 2: P_i is a singleton of degree more than 2. Suppose that $P_i = \{v_g\}$ is a singleton of degree greater than 2 in G_i. Let $v_\ell, w_1, \ldots, w_q, v_r$, with $q \geq 1$, be the neighbors of v_g in Γ_{i-1} as they appear from left to right along C_{i-1}. We will place v_g above w_q, such that the edge connecting them is vertical; see Figure 3b. Equivalently, this corresponds to setting the x-coordinate of v_g to the one of w_q. The y-coordinates of v_g will be determined by the constraints arising from the way that $(v_\ell, v_g), (v_g, w_1), \ldots, (v_g, w_q), (v_g, v_r)$ must be drawn (see Invariants I.5.b, I.5.c and I.5.d). More precisely, each edge $(w_j, v_g), 1 \leq j \leq q$ will be drawn with a vertical segment incident to w_j, which is unoccupied by Invariant I.3, and, if $j \neq q$, with a second segment of slope $s_j = \frac{j}{\Delta-4} \in S_{\text{f}}$. Clearly, if $q \leq \Delta - 3$, then Invariant I.5.c is satisfied. This condition always holds when $i \neq m$, or when $i = m$ and the degree of v_n is strictly less than Δ. We will discuss the remaining case later. Furthermore, the edges (v_ℓ, v_g) and (v_g, v_r) will be drawn afterwards as in Case 1, thereby satisfying Invariants I.2, I.3, I.5.b, and I.5.d.

To ensure that the bend point of each edge $(w_j, v_g), 1 \leq j < q$ lies on a grid point, we first ensure that the horizontal distance between w_j and v_g is a multiple of $\Delta - 3$. To this end, let (w_j, w'_j) be the first edge of C_{i-1} that is encountered when traversing C_{i-1} from w_j to v_2 (possibly $w'_j = w_{j+1}$). We leverage Invariant I.2 to stretch (w_j, w'_j) by at most $\Delta - 4$

units so as to guarantee the required distance between w_j and v_g. Overall, this increases the width by at most $(q-1)\cdot(\Delta-4)$ units. To satisfy Invariant I.4, we set the y-coordinate of v_g to $H(\Gamma_{i-1})+\alpha k$, where $\alpha \in \mathbb{N}^*$ is a parameter that is chosen such that, for every neighbor w_j of v_g, the bend point of the edge (w_j, v_g) lies above Γ_{i-1}; see Figure 3b. Since the height of the flat segment of the edge (w_j, v_g) is $s_j \cdot (x(v_g) - x(w_j))$, where $x(v_g) - x(w_j)$ is its width, the y-coordinate of v_g must be at least $H(\Gamma_{i-1}) + s_j \cdot (x(v_g) - x(w_j))$. We choose $y(v_g)$ as the smallest multiple of k that fulfills all these bounds. Thus, we have

$$y(v_g) \;\leq\; H(\Gamma_{i-1}) \;+\; \max_{j\in\{1,\ldots,q-1\}} \{s_j\cdot(x(v_g)-x(w_j))\}+k \;\leq\; H(\Gamma_{i-1}) \;+\; W(\Gamma_{i-1})+k. \quad (2)$$

Having determined the x- and y-coordinates of v_g, the drawing Γ_i is completed by drawing each edge incident to v_g in G_i as described above, thereby guaranteeing Invariants I.2-I.5. By an argument analogous to that of Case 1, the edges (v_ℓ, v_g) and (v_g, v_r) are crossing-free in Γ_i. We still have to argue that the edges (w_j, v_g) are also crossing free in Γ_i. The flat slopes of the edges (w_j, v_g), with $j < q$, have been assigned in counterclockwise order around v_g. Also, the left-to-right order of the bend points of these edges matches the left-to-right order of $w_1, \ldots, w_{q-1}$ along the contour C_{i-1}; see Figure 3b. Therefore, no two edges (w_j, v_g) and $(w_{j'}, v_g)$ with $0 \leq j < j' \leq q$ can cross each other. Furthermore, the horizontal segments of the edges (v_ℓ, v_g) and (v_r, v_g) lie above the flat segments of the edges (w_j, v_g), while the stretching performed along the edge $(v_\ell, v_{\ell'})$ and $(v_r, v_{r'})$ ensures that the steep segments of (v_ℓ, v_g) and (v_r, v_g) lie completely to the left and to the right of these flat segments; see Figure 4. Since the flat segments of the edges (w_i, v_g) are drawn completely above Γ_{i-1}, they cannot cross any edge of G_{i-1}, thereby implying that Γ_i is planar, as desired. Having ensured this property, the fact that the contour C_i of Γ_i is x-monotone is implied by the fact that the slopes of the two segments of the edge (v_ℓ, v_g) are in $S_{\text{rs}} \cup S_{\text{h}}$, while the ones of (v_g, v_r) in $S_{\text{h}} \cup S_{\text{ls}}$ (i.e., symmetrically to Case 1). Hence, Γ_i also satisfies Invariant I.1.

We now establish an upper bound on how much the drawing Γ_{i-1} must be stretched horizontally to accommodate v_g in Γ_i. As in Case 1, the edges $(v_\ell, v_{\ell'})$ and $(v_r, v_{r'})$ have been stretched by at most $\frac{y(v_g)}{s_\ell}$ and $\frac{y(v_g)}{|s_r|}$ units, respectively. Furthermore, the edges (w_j, w'_j) have been stretched by at most $(q-1)\cdot(\Delta-4)$ unit in total. Thus, the total stretch applied is at most

$$\begin{aligned}
&\frac{y(v_g)}{s_\ell} + \frac{y(v_g)}{|s_r|} + (q-1)\cdot(\Delta-4)\\
&\overset{(2)}{\leq} \frac{H(\Gamma_{i-1})+W(\Gamma_{i-1})+k}{s_\ell} + \frac{H(\Gamma_{i-1})+W(\Gamma_{i-1})+k}{|s_r|} + (\deg(v_g)-1)\cdot(\Delta-4)\\
&\leq 2\cdot\frac{H(\Gamma_{i-1})+W(\Gamma_{i-1})+k}{\frac{k}{\Delta-3}} + \deg(v_g)\cdot\Delta\\
&\leq 2\Delta\cdot\frac{H(\Gamma_{i-1})+W(\Gamma_{i-1})+k}{k} + \Delta n. \qquad (3)
\end{aligned}$$

To complete the description of our drawing algorithm, it remains to consider the case in which v_n is of degree exactly Δ. Let $w_1, \ldots, w_\Delta$ be the neighbors of v_n as they appear along C_{m-1} from v_1 to v_2. Before applying our drawing algorithm as described so far, we remove the edge (w_Δ, v_n) from the graph and recolor the edge $(w_{\Delta-1}, v_n)$ green. After the last step of the drawing algorithm, we reinsert the edge (w_Δ, v_n). Since w_Δ lies on C_m and its degree is strictly less than Δ (in the absence of $(w_{\Delta-1}, v_n)$), the vertical top port of w_Δ is unoccupied by Invariant I.3. By Invariant I.1, we can draw the edge $(w_{\Delta-1}, v_n)$ with a vertical segment at w_Δ and a segment of slope $\frac{1}{\Delta-3}$ at v_n (see Figure 2b). This completes the drawing of the input graph G.

It remains to analyze the width and the height of the drawing. For ease of notation, we denote by W_i and H_i, $0 \leq i \leq m$ the width and height of the drawing Γ_i, that is, $W_i = W(\Gamma_i)$ and $H_i = H(\Gamma_i)$. In the base of our recursive algorithm, it holds that $W_0 = 1$ and $H_0 = 0$. By Equation (2), we obtain $H_i \leq H_{i-1} + W_{i-1} + k$. So,

$$H_m \leq \sum_{i=0}^{m-1} (W_i + k) \leq m \cdot (W_m + k) \leq n \cdot W_m + kn. \tag{4}$$

By Equation (1) and Equation (3), we obtain

$$\begin{aligned} W_i &\leq W_{i-1} + 2\Delta \cdot \frac{H_{i-1} + W_{i-1} + k}{k} + |P_i| + \Delta n \\ &= \frac{2\Delta}{k} \cdot H_{i-1} + (1 + \frac{2\Delta}{k}) \cdot W_{i-1} + (n+2)\Delta + |P_i|. \end{aligned}$$

By series expansion, we obtain

$$\begin{aligned} W_m &\leq \sum_{i=0}^{m-1} \left(1 + \frac{2\Delta}{k}\right)^{m-i} \left(\frac{2\Delta}{k} H_i + (n+2)\Delta + |P_i|\right) \\ &\leq \left(1 + \frac{2\Delta}{k}\right)^{m} \cdot \sum_{i=0}^{m-1} \left(\frac{2\Delta}{k} H_i + (n+2)\Delta + |P_i|\right) \\ &\leq \left(1 + \frac{2\Delta}{k}\right)^{m} \cdot \left(m \cdot \frac{2\Delta}{k} \cdot H_m + (n+2)\Delta m + n\right) \qquad m \leq n, n \geq 6 \\ &\leq \left(1 + \frac{2\Delta}{k}\right)^{n} \cdot \left(n \cdot \frac{2\Delta}{k} \cdot H_m + 2\Delta n^2\right). \end{aligned}$$

If we choose $k = 4\Delta n^2$, then we have $\left(1 + \frac{2\Delta}{k}\right)^n = \left(1 + \frac{1}{2n^2}\right)^n$. Since $\ln(1+x) \leq x$ for $0 < x < 1$, we obtain $\left(1 + \frac{1}{2n^2}\right)^n \leq e^{1/(2n)} \leq e^{1/6} < 1.19$, so

$$W_m \leq 1.19 \left(n \cdot \frac{2\Delta}{k} \cdot H_m + 2\Delta n^2\right). \tag{5}$$

Plugging Equation (5) into Equation (4), we obtain

$$\begin{aligned} H_m &\leq 1.19 \cdot \left(n^2 \cdot \frac{2\Delta}{k} \cdot H_m + 2\Delta n^3\right) + kn \leq \frac{2 \cdot 1.19 \cdot k\Delta n^3 + k^2 n}{k - 2 \cdot 1.19 \cdot \Delta n^2} \\ &= \frac{2 \cdot 1.19 \cdot 4\Delta^2 n^5 + 16\Delta^2 n^5}{4\Delta n^2 - 2 \cdot 1.19 \cdot \Delta n^2} = \frac{25.52\Delta n^3}{1.62} \leq 15.76\Delta n^3 \in O(\Delta n^3). \end{aligned}$$

Plugging this back into Equation (5), we can bound the width by

$$\begin{aligned} W_m &\leq 1.19 \left(n \cdot \frac{2\Delta}{4\Delta n^2} \cdot H_m + 2\Delta n^2\right) \leq \frac{1.19}{2n} \cdot H_m + 2.38\Delta n^2 \leq \frac{18.76\Delta n^3}{2n} + 2.38\Delta n^2 \\ &\leq 11.76\Delta n^2 \in O(\Delta n^2). \end{aligned}$$

Reinserting (v_1, v_2) and (w_Δ, v_n) at the end increases the height by at most $2W_m/(\Delta - 3) \leq 11.76\Delta n^2 \leq 1.96\Delta n^3$, since $\Delta \geq 5$ and $n \geq 6$. Thus, our drawing has area $O(\Delta^2 n^5)$. ◀

Using the algorithms in [20, 22] (see Section 2), we can augment any planar graph with maximum degree $\Delta \geq 5$ to a triconnected planar graph of maximum degree at most $\lceil 3\Delta/2 \rceil + 3$.

▶ **Corollary 2.** *Every planar n-vertex graph with maximum degree $\Delta \geq 5$ admits a 1-bend planar grid drawing with at most $\lceil \frac{9}{2}\Delta \rceil + 1$ slopes on a $O(\Delta n^2) \times O(\Delta n^3)$ grid.*

For the special case of planar graphs of maximum degree 5, we can slightly improve the number of slopes while increasing the required area by a factor of $O(n^2)$.

▶ **Theorem 3.** *Every 3-connected planar graph G with maximum degree $\Delta = 5$ admits a 1-bend planar grid drawing Γ using a fixed set of 5 slopes. Such a drawing can be constructed on a grid of size $O(n^3) \times O(n^4)$.*

Sketch. To reduce the number of slopes, we relax the requirement that the contour of the drawing is x-monotone. Instead, at vertices which do not have remaining outgoing edges non-monotone parts are allowed. This allows to construct a drawing using only five slopes, namely, $\{0, \infty, 1, k, -k\}$, where $k = 5n^2$. The horizontal stretch needed at each step increases to $O(k) + O(W/k) + O(H/k)$, while the height to $O(k) + O(W) + O(H/k)$. Solving the resulting recurrences yields width $O(n^3)$ and height $O(n^4)$. ◀

4 2-bend planar drawings of degree-Δ graphs

In this section, we extend a result of Keszegh, Pach, and Pálvölgyi [24, 25] by showing that every planar graph G with maximum degree $\Delta \geq 3$ admits a 2-bend planar drawing on a grid of polynomial size using at most $\lceil \Delta/2 \rceil$ slopes, if G is biconnected and at most $\lceil \Delta/2 \rceil + 1$ slopes otherwise. We first address the former case.

▶ **Theorem 4.** *Every biconnected planar graph G with maximum degree $\Delta \geq 3$ admits a 2-bend planar drawing with at most $\lceil \Delta/2 \rceil$ distinct slopes on a grid of size $O(n) \times O(n^2\Delta)$. The only exception is the octahedron graph, which requires 3 slopes.*

Proof. Without loss of generality, we may assume that Δ is even. Since planar graphs with $\Delta = 4$ always admit an orthogonal drawing with at most two bends per edge on a grid of size $O(n) \times O(n)$ [5, 37], we may further assume $\Delta \geq 6$. We choose a vertex v_n of G with degree at most 5 (which exists in every planar graph) and consider a planar embedding $\mathcal{E}$ of G with v_n on its outer face. The fact that G is biconnected implies that it admits an st-ordering $v_1, \ldots, v_n$, such that (v_1, v_2) is an edge of the outer face of $\mathcal{E}$. We will use this st-ordering to construct incrementally a 2-bend planar drawing Γ of G on the following set of $\frac{\Delta}{2}$ slopes: $S = \{-\lfloor \frac{\Delta}{4} \rfloor + 1, \ldots, 0, \ldots, \lceil \frac{\Delta}{4} \rceil - 1, \infty\}$; see Figure 5a. In particular, each edge (v_i, v_j) with $i < j$ will be drawn with a *first non-vertical segment* incident to v_i (of possibly zero length), followed by a vertical segment (of non-zero length) and a *last non-vertical segment* incident to v_j (of possibly zero length); see Figure 5b.

We first compute a *sketch* drawing Γ' of G using the algorithm of [24, 25] and taking $v_1, \ldots, v_n$ as the input st-ordering. In the resulting drawing Γ', each edge is drawn with at most two bends and contains a vertical segment. These vertical segments lie in consecutive columns of Γ' that also contain the vertices of G. Given a vertex v of G, we denote by $x(v)$ the index of the column of Γ' that contains v in the left-to-right ordering of the columns of Γ'. Accordingly, given an edge e, with slight abuse of notation, we denote by $x(e)$ the index of the column of Γ' that contains the vertical segment of e in the left-to-right ordering of the columns of Γ'; see Figure 5a. Let $S' = \{s'_{-\lfloor \Delta/4 \rfloor + 1}, \ldots, s'_0, \ldots, s'_{\lceil \Delta/4 \rceil - 1}, s'_\infty\}$ be the slopes used in Γ' such that s'_∞ is the vertical slope and $s'_i < s'_j$ if $i < j$. We produce a drawing Γ of G on a grid of size $O(n) \times O(n^2\Delta)$ such that every edge segment that is drawn with slope s'_i in Γ' is drawn with slope i in Γ. Hence, Γ and Γ' share the same planar embedding. Furthermore, the x-coordinate of each vertex v of G will be $x(v)$ in Γ, while the x-coordinate of the vertical segment of each edge e of G will be $x(e)$. Hence, the width of Γ will be $O(n)$.

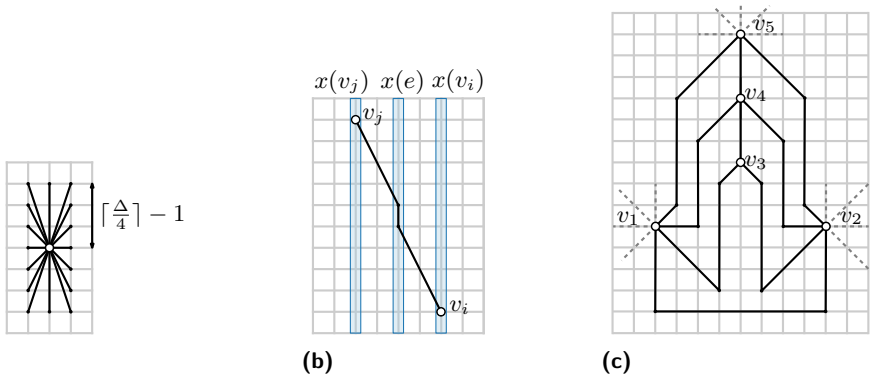


Figure 5 (a) The set of $\lceil \Delta/2 \rceil$ slopes used in Theorem 4. (b) Example construction of a 2-bend edge $e = (v_i, v_j)$. (c) A sketch of our drawing.

Let G_i with $1 \leq i \leq n$ be the subgraph of G induced by $\{v_1, \ldots, v_i\}$, and let $\mathcal{E}_i$ be the restriction of $\mathcal{E}$ to G_i. We say that an edge (v_j, v_k) of G is a *pending* edge of G_i if and only if $j \leq i < k$. For $2 \leq i \leq n$, we denote by Γ_i a drawing of G_i which additionally contains the first non-vertical segment (if any) of each pending edge of G_i. In the following, we will first describe how to compute drawing Γ_2. Then, assuming that we have recursively computed a drawing Γ_{i-1} with $2 \leq i < n$, we describe how to compute drawing Γ_i.

To obtain drawing Γ_2, we begin by placing v_1 at $(0, 0)$ and v_2 at $(x(v_2), 0)$, as in the original algorithm. We connect v_1 to v_2 with a 2-bend edge drawn below both vertices. This edge consists of two vertical segments, each of length $\lfloor \frac{\Delta}{4} \rfloor (2m - n) + 1$, incident to v_1 and v_2, and a horizontal segment connecting their lower endpoints; see Figure 5c. To complete drawing Γ_2, we next draw the first non-vertical segment of each pending edge that is incident to v_1 and v_2 preserving $\mathcal{E}_2$ as follows. For $j \in \{1, 2\}$, consider a pending edge $e = (v_j, v_k)$ with $k > 2$ and suppose that this edge has a non-vertical segment incident to v_j in Γ'. Let s'_ℓ be the slope of this segment in Γ'. We draw a segment starting at v_j with slope ℓ until its x-coordinate equals $x(e)$. This point serves as the first bend of e and is a grid point, since S consists of integer slopes. Furthermore, the vertical extent of this segment is at most $\lfloor \Delta/4 \rfloor \cdot |x(e) - x(v_j)|$. This guarantees that this segment does not cross (v_1, v_2), since the width of the drawing is bounded by $2m - n$, as we will shortly show, and thus, $|x(e) - x(v_j)| \leq 2m - n$.

We now describe how to compute drawing Γ_i, assuming that we have recursively constructed drawing Γ_{i-1}. We place v_i at x-coordinate $x(v_i)$ and y-coordinate $(i - 1) \cdot (\Delta/2 \cdot (2m - n) + 1)$, such that v_i lies above one of its median predecessors in G_i, as in the original algorithm. Then, we complete the drawing of the edges connecting v_i to its neighbors in G_{i-1} while preserving $\mathcal{E}_i$. Consider such an edge $e = (v_j, v_i)$ with $i > j$ and assume that s'_ℓ is the slope of its last non-vertical segment in Γ'. In drawing Γ_i, we draw a segment starting at v_i with slope ℓ until its x-coordinate equals $x(e)$ followed by a vertical segment that connects it to the endpoint of the first non-vertical segment of the edge e (which has already been drawn, when Γ_j was computed). To complete the drawing of Γ_i, we draw the first non-vertical segments of the pending edges incident to v_i in the same way as described for Γ_2. It follows that the vertical extent of any non-vertical segment incident to v_i belonging to edge e is bounded by $\lfloor \Delta/4 \rfloor \cdot |x(e) - x(v_i)|$.

Since each vertex v_i and its incident edges occupy at most $\deg(v_i) - 1$ columns, the total width of the resulting drawing Γ of G is bounded by $\sum_{i=1}^{n} (\deg(v_i) - 1) = 2m - n \in O(n)$. To estimate the height of Γ, let H_i be the maximum y-coordinate in Γ_i. For each $i > 2$, H_i is

determined by the y-coordinate of v_i plus the vertical extent of the first non-vertical segment of each of the pending edges of v_i plus the vertical extent of edge (v_1, v_2). Thus,

$$\begin{aligned} H_{i-1} &\leq y(v_{i-1}) + \lfloor\frac{\Delta}{4}\rfloor \cdot W_{i-2} + \lfloor\frac{\Delta}{4}\rfloor(2m-n)+1 \\ &\leq (i-2)\cdot\left(\frac{\Delta}{2}\cdot(2m-n)+1\right) + \frac{1}{2}\cdot\frac{\Delta}{2}\cdot(2m-n) + \lfloor\frac{\Delta}{4}\rfloor(2m-n)+1 \\ &< \left(\frac{\Delta}{2}\cdot(2m-n)+1\right)\left(i-\frac{3}{2}\right) + \lfloor\frac{\Delta}{4}\rfloor(2m-n)+1. \end{aligned}$$

The lowest point of any segment of a pending edge of v_i in Γ_i is at least

$$y(v_i) - \lfloor\frac{\Delta}{4}\rfloor \cdot W_{i-1} + \lfloor\frac{\Delta}{4}\rfloor(2m-n)+1 > \left(\frac{\Delta}{2}\cdot(2m-n)+1\right)\left(i-\frac{3}{2}\right) + \lfloor\frac{\Delta}{4}\rfloor(2m-n)+1.$$

This ensures that all segments of the pending edge of v_i lie above drawing Γ_{i-1}. Hence Γ_i is planar. The total height of the drawing is $O(n \cdot \Delta \cdot n) = O(\Delta n^2)$. ◀

Since any simply connected planar graph can be augmented to a biconnected planar graph by adding auxiliary edges such that every vertex receives at most two augmenting incident edges [20, 22], we obtain the following corollary.

▶ **Corollary 5.** *Every planar graph G with maximum degree Δ admits a 2-bend planar drawing with at most $\lceil\Delta/2\rceil + 1$ distinct slopes on a grid of size $O(n) \times O(\Delta n^2)$.*

5 4-bend planar drawings of degree-Δ graphs

In this section, we seek to prove that every planar graph of maximum degree Δ admits a 4-bend planar grid drawing with at most Δ slopes on an $O(n) \times O(n)$ grid. Our approach is based on an algorithm by Kaufmann and Wiese [23], which given a planar graph $G = (V, E)$ and a set of points P in the plane such that no two points have the same x-coordinate, it computes a 2-bend planar drawing of G that maps each vertex in V to a point in P. Let the *spine* be the x-monotone polyline whose bend-points are exactly the points in P. In the produced drawing, every edge is one of the following (see Figure 6a): **(i)** a *top edge*: it is drawn completely above the spine with one bend; **(ii)** a *bottom edge*: it is drawn completely below the spine with one bend; or **(iii)** a *spine-crossing edge*: it crosses the spine exactly once and has two bend points: one below and one above the spine. Note that the drawings produced by their algorithm require a linear number of slopes and, as stated in [23, Lemma 3.2], exponential area (in the number of vertices).

▶ **Theorem 6.** *Every planar graph of maximum degree Δ admits a 4-bend planar drawing with Δ slopes on a grid of size $O(n) \times O(n)$.*

Proof. We will create a drawing that is conceptually similar to the one by Kaufmann and Wiese. We first apply their algorithm on a set of points that lie on the diagonal $y = x$ to obtain a drawing Γ of G. Let Γ' be the drawing of the graph $G' = (V', E')$ obtained from Γ by subdividing every spine-crossing edge e by a dummy vertex placed on the crossing between e and the spine. We will show how to obtain a 3-bend planar drawing with Δ slopes for G' on a grid of size $O(n) \times O(n)$, in which the two edges incident to each dummy vertex use at most two bends and are drawn such that they use opposite (horizontal or vertical) ports at the incident dummy vertex. Thus, smoothing the dummy vertices yields a 4-bend drawing of G with Δ slopes on a grid of size $O(n) \times O(n)$.

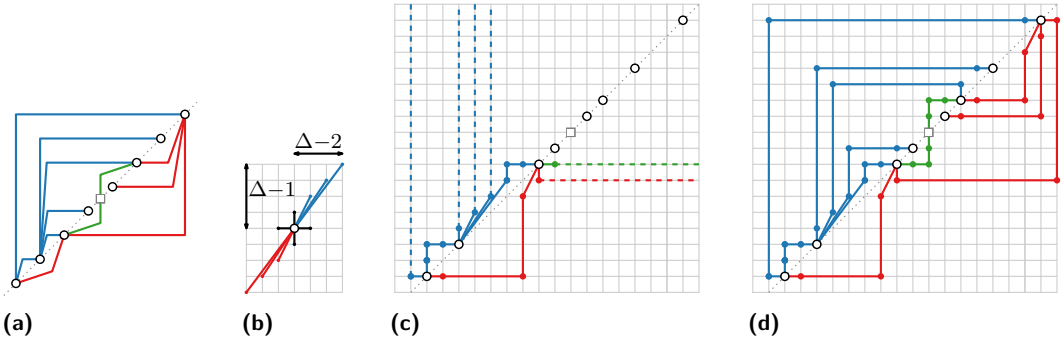


Figure 6 Illustration for our algorithm in Theorem 6 for the graph in Figure 6. (a) A 2-bend planar drawing obtained by the algorithm of Kaufmann and Wiese [23]. Top edges are drawn blue, bottom edges red, and spine-crossing edges green. (b) The vectors used around each vertex in Theorem 6. (c) The drawing after processing v_3 and (d) the final drawing.

Let $v_1, \ldots, v_{n'}$ be the vertices of V' in the order that they appear along the spine in Γ' and assume that every edge (v_i, v_j) with $i < j$ is oriented from v_i to v_j. Let t_i and b_i be the number of top and bottom edges of v_i, respectively. We place v_1 at point $(0, 0)$ and every other vertex v_i with $1 < i \leq n'$ at point $p_{v_i} = p_{v_{i-1}} + (d_{i-1}, d_{i-1})$, where $d_{i-1} = \max\{1, t_{i-1}, b_i\}$. Furthermore, the j-th top (bottom, respective) edge incident to v_i in clockwise order around v_i starting from the spine will have a bend at point at $p_{v_i} + \mathbf{v}_j$ ($p_{v_i} - \mathbf{v}_j$, respectively), where $\mathbf{v}_j = (j - 2, j - 1)$. This ensures that Δ distinct slopes are used by the edge segments incident to each vertex; all the edge segments not directly incident to a vertex are drawn either horizontally or vertically.

To achieve the above properties, we process the vertices in the order $v_1, \ldots, v_{n'}$. Assume that we have already processed the vertices $v_1, \ldots, v_{i-1}$, and consider the next vertex v_i. We call an edge (v_k, v_ℓ) with $k < \ell$ *open* if we have already processed v_k but not yet v_ℓ, that is, $k < i \leq \ell$. As an invariant of our algorithm, each open top edge is assigned to a grid column that will contain its vertical segment, and each open bottom edge is assigned to a grid row that will contain its horizontal segment; see Figure 6c.

Let $e^{\mathrm{t}}_1, \ldots, e^{\mathrm{t}}_{t_i}$ denote the top edges of v_i in clockwise order around v_i, starting from the spine. As already mentioned, for each such edge e^{t}_j, we place a bend at point $p_{v_i} + \mathbf{v}_j$. If e^{t}_j is open, we assign it to the grid column $x(v_i) + j - 2$. Otherwise, we draw a horizontal segment to the left until reaching the grid column previously assigned to e^{t}_j, and then a vertical segment downward to the bend point located in the neighborhood of the other endpoint of e^{t}_j. In this way, each edge e^{t}_j is drawn using at most three bends. Symmetrically, let $e^{\mathrm{b}}_1, \ldots, e^{\mathrm{b}}_{b_i}$ be the bottom edges of v_i in clockwise order around v_i, starting from the spine. For each such edge e^{b}_j, we place a bend at the point $p_{v_i} - \mathbf{v}_j$. If e^{b}_j is open, we assign it to the grid row $y(v_i) - j + 1$. Otherwise, we draw a vertical segment to the bottom and then a horizontal segment leftward to the bend point located in the neighborhood of the other endpoint of e^{b}_j. Hence, each edge e^{b}_j is also drawn using at most three bends and the invariant of our algorithm is satisfied.

Once all vertices have been processed, he have obtained a 3-bend drawing of G'; see Figure 6d for an illustration. The obtained drawing is planar because (i) no two edge segments are assigned to the same horizontal or vertical grid column, and (ii) throughout the incremental drawing construction, the planar embedding of the algorithm by Kaufmann and Wiese is maintained due to the choice of the bend points around each vertex.

We now argue that after smoothing the dummy vertices, we obtain a 4-bend drawing of G. If v_i is a dummy vertex of G', then it has exactly two edges: one incoming edge (v_h, v_i) and one outgoing edge (v_i, v_j) with $h < i < j$, one of which is a top edge, while the other one is a bottom edge. If (v_h, v_i) is a top edge, then both edges use a horizontal segment around v_i. Hence, after smoothing v_i, the spine-crossing edge (v_h, v_j) will be drawn with a short segment at v_h, followed by a long vertical segment, a horizontal segment that crosses the spine at the old position of v_i, a long vertical segment, and a short segment at v_j, so it has four bends in total. The case that (v_h, v_i) is a bottom edge is symmetric.

For the drawing area, recall that the difference in x-coordinates (and y-coordinates) between two consecutive vertices v_i and v_{i+1} is $d_i = \max\{1, t_i, b_{i+1}\} \leq 1 + t_i + b_{i+1}$, so we have $x(v_{n'}) = y(v_{n'}) \leq n' + \sum_{i=1}^{n'-1} t_i + \sum_{i=2}^{n'} b_i = n' + 2m - t_{n'} - b_1$. Some edges can extend to the bottom-left of v_1 and to the top-right of $v_{n'}$. Namely, one top edge might occupy one column to the left of v_1, $b_1 - 2$ bottom edges might occupy one column to the left of v_1 each, and $b_1 - 1$ bottom edges might occupy one row to the bottom of v_1 each. Furthermore, to the right of $v_{n'}$, one column might be occupied by a bottom edge, and $t_{n'} - 2$ columns might be occupied by a top edge; above $v_{n'}$, $t_{n'} - 1$ rows might be occupied by top edges. Thus, the total width and height of the drawing is at most

$$\begin{aligned} x(v_{n'}) - x(v_1) + b_1 + 1 + t_{n'} + 1 &\leq n' + 2m - t_{n'} - b_1 - 0 + b_1 + t_{n'} + 2 \\ &= n' + 2m + 2 \leq (n + m) + 2m + 2 \leq 10n - 16 \in O(n). \end{aligned}$$ ◀

Since every *subhamiltonian* graph (i.e., a subgraph of a planar Hamiltonian graph) admits an embedding consisting exclusively of top and bottom edges, the following is a direct consequence of Theorem 6.

▶ **Corollary 7.** *Every subhamiltonian graph of maximum degree Δ admits a 3-bend planar drawing with Δ slopes on a grid of size $O(n) \times O(n)$.*

6 Conclusions and Open Problems

In this work, we studied trade-offs between the number of slopes, the number of bends per edge, and the area requirements in planar graph drawing. Our results show that allowing only a small number of bends per edge suffices to obtain polynomial-area drawings while maintaining a relatively small slope set. Our results narrow a gap between previous approaches that achieved few slopes at the expense of super-polynomial area and those that focused primarily on low-degree graphs. However, several questions remain open. Besides tightening the trade-offs between slopes and bends and extending the study to non-planar graph classes, we identify the following open problems.

- The algorithm presented in Section 3 yields drawings in which almost all edges have a bend. The only exception are the black edges and a red incoming edge per vertex. Following an argument of [27], we can choose a canonical order such that there are at most $(2n + 1)/3$ vertices without an incoming red edge. This gives us an upper bound of $(8n - 17)/3$ on the total number of bends for the constructed drawing. Adjusting the drawing algorithms, so as to reduce the total number of bends in the resulting 1-bend drawing is an interesting open problem for future consideration.
- We were unable to derive an extension of the algorithm presented in Section 3 to handle biconnected (and simply connected) planar graphs without increasing the number of slopes. Such an extension appears to be non-trivial and may require additional properties on the produced drawings in order to handle rigid 3-connected components of the graph.

- The drawings produced by our algorithms have low *angular resolution*, that is, the minimum angle between any two edge segments incident to the same vertex is small. Developing techniques that also account for angular resolution is an interesting direction for future work.
- It is known that the straight-line slope number of outerplanar graphs and partial 2-trees is $\Delta - 1$ and 2Δ, respectively [31, 33]. However, the existing algorithms for these results require superpolynomial area. A natural question is whether we can achieve polynomial area drawings for these graph classes by allowing a slight increase in the number of slopes.

References

1 Patrizio Angelini, Michael A. Bekos, Giuseppe Liotta, and Fabrizio Montecchiani. Universal slope sets for 1-bend planar drawings. *Algorithmica*, 81(6):2527–2556, 2019. doi:10.1007/S00453-018-00542-9.

2 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

3 Michael Bekos, Martin Gronemann, Michael Kaufmann, and Robert Krug. Planar octilinear drawings with one bend per edge. *J. Graph Algorithms Appl.*, 19(2):657–680, 2015. doi:10.7155/jgaa.00369.

4 Michael A. Bekos, Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. Universal slope sets for upward planar drawings. *Algorithmica*, 84(9):2556–2580, 2022. doi:10.1007/S00453-022-00975-3.

5 Therese Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998. doi:10.1016/S0925-7721(97)00026-6.

6 Guido Brückner, Nadine Davina Krisam, and Tamara Mchedlidze. Level-planar drawings with few slopes. *Algorithmica*, 84(1):176–196, 2022. doi:10.1007/S00453-021-00884-X.

7 Steven Chaplick, Giordano Da Lozzo, Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Planar drawings with few slopes of Halin graphs and nested pseudotrees. *Algorithmica*, 86(8):2413–2447, 2024. doi:10.1007/S00453-024-01230-7.

8 Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *J. Graph Algorithms Appl.*, 16(3):635–650, 2012. doi:10.7155/JGAA.00265.

9 Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Comb.*, 10(1):41–51, 1990. doi:10.1007/BF02122694.

10 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. The planar slope number of subcubic graphs. In Alberto Pardo and Alfredo Viola, editors, *Theoretical Informatics (LATIN 2014)*, pages 132–143. Springer, 2014. doi:10.1007/978-3-642-54423-1_12.

11 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Drawing outer 1-planar graphs with few slopes. *J. Graph Algorithms Appl.*, 19(2):707–741, 2015. doi:10.7155/JGAA.00376.

12 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Drawing subcubic planar graphs with four slopes and optimal angular resolution. *Theor. Comput. Sci.*, 714:51–73, 2018. doi:10.1016/J.TCS.2017.12.004.

13 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. 1-bend upward planar slope number of SP-digraphs. *Comput. Geom.*, 90:101628, 2020. doi:10.1016/J.COMGEO.2020.101628.

14 Vida Dujmovic, David Eppstein, Matthew Suderman, and David R. Wood. Drawings of planar graphs with few slopes and segments. *Comput. Geom.*, 38(3):194–212, 2007. doi:10.1016/J.COMGEO.2006.09.002.

15 Vida Dujmovic, Matthew Suderman, and David R. Wood. Graph drawings with few slopes. *Comput. Geom.*, 38(3):181–193, 2007. doi:10.1016/J.COMGEO.2006.08.002.

16 Stefan Felsner. *Geometric Graphs and Arrangements*. Advanced Lectures in Mathematics. Vieweg, 2004. doi:10.1007/978-3-322-80303-0.

17 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.

18 Vít Jelínek, Eva Jelínková, Jan Kratochvíl, Bernard Lidický, Marek Tesar, and Tomás Vyskocil. The planar slope number of planar partial 3-trees of bounded degree. *Graphs Comb.*, 29(4):981–1005, 2013. doi:10.1007/S00373-012-1157-Z.

19 Michael Jünger and Petra Mutzel, editors. *Graph Drawing Software*. Springer, 2004. doi:10.1007/978-3-642-18638-7.

20 Goos Kant. *Algorithms for Drawing Planar Graphs*. PhD thesis, Utrecht University, Utrecht, Netherlands, 1993. URL: https://dspace.library.uu.nl/bitstream/handle/1874/842/full.pdf.

21 Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996. doi:10.1007/BF02086606.

22 Goos Kant and Hans L. Bodlaender. Triangulating planar graphs while minimizing the maximum degree. In Otto Nurmi and Esko Ukkonen, editors, *Algorithm Theory (SWAT 1992)*, volume 621 of *LNCS*, pages 258–271. Springer, 1992. doi:10.1007/3-540-55706-7_22.

23 Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms Appl.*, 6(1):115–129, 2002. doi:10.7155/jgaa.00046.

24 Balázs Keszegh, János Pach, and Dömötör Pálvölgyi. Drawing planar graphs of bounded degree with few slopes. In Ulrik Brandes and Sabine Cornelsen, editors, *Graph Drawing (GD 2011)*, LNCS, pages 293–304. Springer, 2011. doi:10.1007/978-3-642-18469-7_27.

25 Balázs Keszegh, János Pach, and Dömötör Pálvölgyi. Drawing planar graphs of bounded degree with few slopes. *SIAM J. Discr. Math.*, 27(2):1171–1183, 2013. doi:10.1137/100815001.

26 Balázs Keszegh, János Pach, Dömötör Pálvölgyi, and Géza Tóth. Drawing cubic graphs with at most five slopes. *Comput. Geom.*, 40(2):138–147, 2008. doi:10.1016/J.COMGEO.2007.05.003.

27 Philipp Kindermann, Tamara Mchedlidze, Thomas Schneck, and Antonios Symvonis. Drawing planar graphs with few segments on a polynomial grid. In Daniel Archambault and Csaba D. Tóth, editors, *Graph Drawing (GD 2019)*, volume 11904 of *LNCS*, pages 416–429. Springer, 2019. doi:10.1007/978-3-030-35802-0_32.

28 Philipp Kindermann, Fabrizio Montecchiani, Lena Schlipf, and André Schulz. Drawing subcubic 1-planar graphs with few bends, few slopes, and large angles. *J. Graph Algorithms Appl.*, 25(1):1–28, 2021. doi:10.7155/JGAA.00547.

29 Jonathan Klawitter and Tamara Mchedlidze. Upward planar drawings with two slopes. *J. Graph Algorithms Appl.*, 26(1):171–198, 2022. doi:10.7155/JGAA.00587.

30 Jonathan Klawitter and Johannes Zink. Upward planar drawings with three and more slopes. *J. Graph Algorithms Appl.*, 27(2):49–70, 2023. doi:10.7155/JGAA.00617.

31 Kolja B. Knauer, Piotr Micek, and Bartosz Walczak. Outerplanar graph drawings with few slopes. *Comput. Geom.*, 47(5):614–624, 2014. doi:10.1016/J.COMGEO.2014.01.003.

32 Kolja B. Knauer and Bartosz Walczak. Graph drawings with one bend and few slopes. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *Theoretical Informatics (LATIN 2016)*, volume 9644 of *LNCS*, pages 549–561. Springer, 2016. doi:10.1007/978-3-662-49529-2_41.

33 William J. Lenhart, Giuseppe Liotta, Debajyoti Mondal, and Rahnuma Islam Nishat. Drawing partial 2-trees with few slopes. *Algorithmica*, 85(5):1156–1175, 2023. doi:10.1007/S00453-022-01065-0.

34 Padmini Mukkamala and Dömötör Pálvölgyi. Drawing cubic graphs with the four basic slopes. In Marc J. van Kreveld and Bettina Speckmann, editors, *Graph Drawing and Network Visualization (GD 2011)*, volume 7034 of *LNCS*, pages 254–265. Springer, 2011. doi:10.1007/978-3-642-25878-7_25.

35 Padmini Mukkamala and Mario Szegedy. Geometric representation of cubic graphs with four directions. *Comput. Geom.*, 42(9):842–851, 2009. doi:10.1016/J.COMGEO.2009.01.005.

36 János Pach and Dömötör Pálvölgyi. Bounded-degree graphs can have arbitrarily large slope numbers. *Electron. J. Comb.*, 13(1), 2006. doi:10.37236/1139.

37 Achilleas Papakostas and Ioannis G. Tollis. A pairing technique for area-efficient orthogonal drawings. In Stephen C. North, editor, *Graph Drawing (GD 1996)*, volume 1190 of *LNCS*, pages 355–370. Springer, 1996. doi:10.1007/3-540-62495-3_60.

38 Pierre Rosenstiehl and Robert Endre Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discret. Comput. Geom.*, 1:343–353, 1986. doi:10.1007/BF02187706.

39 Walter Schnyder. Embedding planar graphs on the grid. In *ACM-SIAM Symposium on Discrete Algorithms (SODA 1990)*, pages 138–148. SIAM, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320191.

40 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.

41 Roberto Tamassia, editor. *Handbook on Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013. URL: https://www.crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

42 Greg A. Wade and Jiang-Hsing Chu. Drawability of complete graphs using a minimal slope set. *The Computer Journal*, 37(2):139–142, 1994. doi:10.1093/COMJNL/37.2.139.

On the Doubling Dimension and the Perimeter of Geodesically Convex Sets in Fat Polygons

Mark de Berg
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Leonidas Theocharous
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Abstract

Many algorithmic problems can be solved (almost) as efficiently in metric spaces of bounded doubling dimension as in Euclidean space. Unfortunately, the metric space defined by points in a simple polygon equipped with the geodesic distance does not necessarily have bounded doubling dimension. We therefore study the doubling dimension of fat polygons, for two well-known fatness definitions. We prove that locally-fat simple polygons *do not* always have bounded doubling dimension, while any (α, β)-covered polygon *does* have bounded doubling dimension (even if it has holes). We also study the perimeter of geodesically convex sets in (α, β)-covered polygons (possibly with holes), and show that this perimeter is at most a constant times the Euclidean diameter of the set.

Using these two results, we obtain new results for several problems on (α, β)-covered polygons, including an algorithm that computes the closest pair of a set of m points in an (α, β)-covered polygon with n vertices that runs in $O(n + m \log n)$ expected time.

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

Keywords and phrases Fat polygons, doubling dimension

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.7

Funding *Mark de Berg*: MdB is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.
Prosenjit Bose: PB is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).
Leonidas Theocharous: LT is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1 Introduction

Motivation and background. In computational geometry, the standard way to model planar objects or regions is by polygons. Because polygons can have very complicated shapes and/or interact with each other in complicated ways, developing efficient solutions for algorithmic problems involving polygons is not always easy. In practice, however, the objects or regions under consideration are usually fairly well-behaved. The *realistic input models* paradigm [13] suggests to make this precise by defining properties of the input that are expected to be satisfied in practice and that allow for easier and/or more efficient algorithmic solutions.

One such property that has been studied extensively is *fatness*, where the underlying assumption is that real-world objects or regions are not extremely thin; see below for more precise definitions. For example, there have been results on motion-planning in environments where the obstacles are fat [26, 27] on various graphics-related problems involving fat objects [1, 21, 11, 10], on flow problems on terrains consisting of fat triangles [9], on various data-structure problems for fat objects [16, 20, 23], and more. Many of these results use one

© Mark de Berg, Prosenjit Bose, and Leonidas Theocharous;
licensed under Creative Commons License CC-BY 4.0

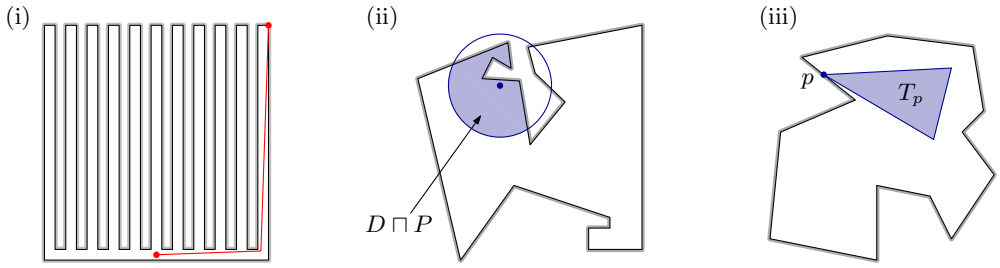


Figure 1 (i) A γ-fat polygon that is not locally γ-fat. (ii) A locally γ-fat polygon. (iii) An (α, β)-covered polygon.

of the following two fundamental results: (i) the union complexity of n constant-complexity fat polygons in the plane is near-linear; see [3] and the references therein; and (ii) any set of n constant-complexity disjoint fat objects in the plane admits a hierarchical decomposition of the plane into constant-complexity cells that each intersect $O(1)$ objects [7]. Note that both results concern constant-complexity objects and that they essentially say something about the part of the space outside the objects. We are interested in what happens *inside* a fat polygon. In particular, we are interested in properties of the metric space (P, dist_g) where P is a fat polygon (possibly with holes) and dist_g is the geodesic distance. In other words, $\mathrm{dist}_g(p, q)$ is the length of a shortest path between two points p and q in P. There has been work on guarding and triangulating fat polygons [2], but the special properties that the metric space (P, dist_g) may or may not have for fat polygons have not been studied much, to the best of our knowledge. This is surprising as problems involving shortest paths in polygons have received widespread attention, and fatness has been studied and used extensively as well. The only paper that is closely related to our work is by Bose, Cheong, and Dujmovic [5], who study the perimeter of fat objects, as discussed below.

Fatness variants. The study of fat polygons was initiated by Matoušek et al. [22], who studied the union complexity of α-*fat triangles*, which are triangles whose angles are at least α, for some fixed constant $\alpha > 0$. Since then, the concept of fatness has been generalized in different ways, both to convex and to non-convex objects. For convex objects, all definitions are equivalent in the sense that a convex object that is fat under one definition is also fat under the other definitions [25, 28]. Our interest lies in non-convex objects, since inside a convex object the geodesic distance is identical to the Euclidean distance. For non-convex objects, different fatness definitions have been proposed as well, but these are no longer equivalent. We discuss the most popular ones, focusing on definitions that apply to polygons.

Overmars, Halperin, and Van der Stappen [26] define a polygon P to be γ-*fat* if, for any Euclidean disk D centered inside P and not fully containing P, we have $\mathrm{area}(P \cap D) \geqslant \gamma \cdot \mathrm{area}(D)$, for some fixed constant $\gamma > 0$. This definition still allows the polygon to have very thin parts, as shown in Figure 1(i). As a result, a collection of n γ-fat, constant-complexity polygons can have quadratic union complexity. Hence, De Berg [8] introduced *locally* γ-*fat polygons*; here we require that for any Euclidean disk D centered inside P and not fully containing P, we have $\mathrm{area}(P \sqcap D) \geqslant \gamma \cdot \mathrm{area}(D)$, where $D \sqcap P$ denotes the connected component of $D \cap P$ that contains the center of D; see Figure 1(ii).

A third definition, by Efrat [15], is the following: a polygon is (α, β)-*covered* if for every p on the boundary ∂P, there exists an α-fat triangle $T_p \subset P$ with p as a vertex such that each side of T_p has length at least $\beta \cdot \mathrm{diam}(P)$, where α and β are fixed positive constants.

We will refer to such a triangle T_p as a *witness triangle* for the point p; see Figure 1(i). This is a stronger condition than local fatness: any (α, β)-covered polygon is locally γ-fat for a constant γ that depends only on the constants α and β, but the reverse is not true [8].

The three definitions above also apply to non-polygonal domains, even if they have holes.

Our contribution. Recall that a metric space $(\mathcal{X}, d)$ is *c-doubling* if every ball of radius r in the space $\mathcal{X}$ can be covered by c balls of radius $r/2$. The *doubling dimension* of the metric space is defined to be $\log_2 c$. If c is a constant, then we say that the metric space has *bounded doubling dimension*. Spaces of bounded doubling dimension generalize Euclidean space of constant dimension, and many problems can be solved (almost) as efficiently on spaces of bounded doubling dimension as they can in Euclidean space. Unfortunately, if P is an arbitrary polygon then the metric space (P, dist_g) may not have bounded doubling dimension. The first problem we study is therefore: does the metric space (P, dist_g) necessarily have bounded doubling dimension when P is fat?

For the fatness definition of Overmars et al. [26], the answer is easily seen to be no, as exemplified by the polygon in Figure 1(i): the ball centered at the red point in the middle of the polygon and whose radius r is the geodesic distance to the top-right vertex, needs $\Omega(n)$ geodesic balls of radius $r/2$ to be covered. Our first two results, presented in Section 2, concern the doubling dimension of locally-fat polygons and (α, β)-covered polygons.

- We show that locally-fat polygons do not necessarily have bounded doubling dimension. More precisely, we show that for any $n \geqslant 3$ there exists a simple polygon P with n vertices and a geodesic disk D in P such that $\Omega(n^{1/3})$ geodesic disks of radius r are needed to cover D, where $r = \mathrm{radius}(D)$.
- On the positive side, we prove that the doubling dimension of any (α, β)-covered polygon is upper bounded by a constant that only depends on the constants α and β and not on the number of vertices of the polygon. This result even holds for (α, β)-covered domains with curved boundaries and holes.

The fact that (α, β)-covered polygonal regions have bounded doubling dimension immediately implies a plethora of results that improve upon known results for non-fat polygons; we mention several of them at the end of Section 2.

After studying the doubling dimension of fat polygons, we turn our attention in Section 3 to the following problem: can the perimeter $\mathrm{per}(R)$ of a geodesically convex polygonal region R be bounded in terms of its geodesic diameter $\mathrm{diam}_g(R)$, or perhaps even in terms of its Euclidean diameter $\mathrm{diam}(R)$? For arbitrary polygons the answer is clearly no. In the polygon of Figure 1(i) the perimeter $\mathrm{per}(P)$ of P, which is geodesically convex, is $\Omega(n \cdot \mathrm{diam}_g(P))$, and hence arbitrarily larger than $\mathrm{diam}(P)$. Bose, Cheong, and Dujmović [5] showed that the same can happen for locally-fat polygons, via the Koch snowflake. For (α, β)-covered simple polygons, on the other hand, they showed that $\mathrm{per}(P) = O(\mathrm{diam}(P))$, where the constant of proportionality only depends on the constants α and β and not on the number of vertices of the polygon. The applicability of this result is limited, because it only bounds the perimeter of the polygon itself and not of geodesically convex sets inside the polygon. We show that the result also holds in this much more general setting.

- For any geodesically convex polygonal set R in an (α, β)-covered polygonal domain P, we have $\mathrm{per}(R) = O(\mathrm{diam}(R))$.

A fairly immediate consequence of this result is the existence of an ε-coreset of size $O(1/\varepsilon)$ for furthest-neighbor queries in an (α, β)-covered simple polygon P. This improves on the $O(1/\varepsilon^2)$ bound recently proved by De Berg and Theocharous [12] for general simple polygons. The result can also be used to obtain an algorithm for the closest-pair problem on a set S of m points in P whose expected running time is linear in m, as explained in Section 4.

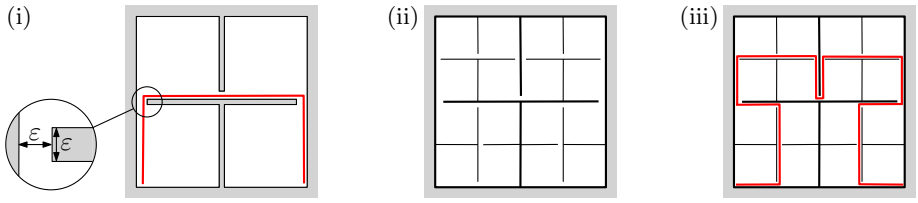


Figure 2 (i) The basic building block P_1 and the shortest path from $(0,0)$ to $(1,0)$ in P_1. (ii) Schematic illustration of P_2. (iii) The shortest path from $(0,0)$ to $(1,0)$ in P_2.

Notation. We denote the Euclidean disk of radius r centered at a point $p \in \mathbb{R}^2$ by $D(p,r)$ and we denote the Euclidean distance between two points p, q by $|pq|$. For a polygon P, which will always be clear from the context, and two points $p, q \in P$, we use $\pi(p,q)$ to denote the shortest path between p and q inside P, and we denote the length of this path by $\|\pi(p,q)\|$. Thus, $\mathrm{dist}_g(p,q) = \|\pi(p,q)\|$. The geodesic disk of radius r centered at a point $p \in P$ is denoted by $D_g(p,r)$. Recall that $\mathrm{diam}(P)$ denotes the Euclidean diameter of P and that $\mathrm{diam}_g(P)$ denotes the geodesic diameter. From now on, when we speak of the doubling dimension of P, we always refer to the doubling dimension of the metric space (P, dist_g).

2 The doubling dimension of fat polygons

In this section we investigate the doubling dimension of fat polygons. We first show that locally-fat polygons do not necessarily have bounded doubling dimension, and then we show that (α,β)-covered polygons do have bounded doubling dimension.

2.1 Locally-fat polygons do not have bounded doubling dimension

To prove that locally-fat polygons do not have bounded doubling dimension, we first construct a family of locally-fat polygons P_m, for $m \geqslant 1$, such that $\mathrm{diam}(P_m) = \sqrt{2}$ and $\mathrm{diam}_g(P_m) \approx 2^m$. The polygon P_m will be constructed recursively inside the unit square $[0,1]^2$, in such a way that the shortest path from the lower-left corner $(0,0)$ to the lower-right corner $(1,0)$ resembles the order-m Hilbert curve.

The initial polygon P_1, illustrated in Figure 2(i), is constructed as follows. Take the unit square $[0,1]^2$ and a small value $\varepsilon > 0$, and add the horizontal segment $[\varepsilon, 1-\varepsilon] \times \frac{1}{2}$ and the vertical segments $\frac{1}{2} \times [0, \frac{1}{2}]$ and $\frac{1}{2} \times [\frac{1}{2} + \varepsilon, 1]$. Then slightly inflate these segments to give them width ε, thus obtaining a non-degenerate simple polygon. Note that the length of the shortest path from $(0,0)$ to $(1,0)$ approaches 2 as $\varepsilon \to 0$; see the red path in Figure 2(i). This shortest path resembles the order-1 Hilbert curve, except that the middle link has twice the length of the first and last link, whereas in the Hilbert curve all links have the same length.

We will recursively construct P_m, for $m \geqslant 2$, such that the shortest path $\pi((0,0),(1,0))$ from $(0,0)$ to $(1,0)$ resembles the order-m Hilbert curve as $\varepsilon \to 0$. We construct P_m from P_{m-1} as follows. We scale P_{m-1} by $\frac{1}{2}$ and place four copies of it in a 2×2 grid whose cells have size $\frac{1}{2^{m-1}}$, where the south-west copy is rotated clockwise by $\frac{\pi}{2}$ and the south-east copy is rotated counter-clockwise by $\frac{\pi}{2}$. Finally, we create small openings in the top-left corner of the south-west copy, in the bottom-right corner of the north-west copy, and in the top-right corner of the south-east copy. This ensures that P_m is a simple polygon and that the only way to go from $(0,0)$ to $(1,0)$ is to pass through these openings and visit all four copies See Figure 2(ii) for an illustration of P_2. Note that the Euclidean diameter of P_m equals $\sqrt{2}$.

Furthermore, the length L_m of the shortest path $\pi((0,0),(1,0))$ in P_m (as $\varepsilon \to 0$) satisfies

$$L_m = 4 \cdot \left(\frac{1}{2} L_{m-1}\right) = 2 \cdot L_{m-1},$$

where $L_1 = 2$. Thus, $L_m = 2^m$. We obtain the following observation.

▶ **Observation 1.** *The polygon P_m has $O(4^m)$ vertices and is such that* $\mathrm{diam}(P_m) = \sqrt{2}$ *and $L_m \to 2^m$ as $\varepsilon \to 0$, where L_m is the length of the shortest path in P_m from $(0,0)$ to $(1,0)$.*

The next lemma shows that P_m is locally fat.

▶ **Lemma 2.** *The polygon P_m is locally γ-fat for $\gamma = \frac{1}{8\pi}$.*

Proof. For simplicity we assume in the computations below that ε is infinitesimally small. Let $D(p,r)$ be a disk centered at a point $p \in P_m$ such that $D(p,r)$ does not contain P_m in its interior. Consider a hierarchical grid inside the unit square $[0,1]^2$, where the cells at level ℓ have side length $\frac{1}{2^\ell}$ and diameter $\frac{\sqrt{2}}{2^\ell}$. Let $S_0, S_1, ..., S_m$ be the squares of level $0, 1, ..., m$ containing p. Since $P_m \not\subset D(p,r)$, we have $r < \sqrt{2}$. We consider three cases.

- Case I: $r < \frac{\mathrm{diam}(S_m)}{2}$. Then $D(p,r)$ does not contain S_m. Moreover, since S_m is a square at the deepest level, it doesn't contain any parts of ∂P in its interior by construction. This implies that $\mathrm{area}(D(p,r) \sqcap P_m) \geqslant \frac{\pi r^2}{4}$. Hence,

$$\mathrm{area}(D(p,r) \sqcap P_m) \geqslant \mathrm{area}(D(p,r) \sqcap S_m) \geqslant \frac{\pi r^2}{4} \geqslant \frac{1}{4} \cdot \mathrm{area}(D(p,r)).$$

- Case II: $\frac{\mathrm{diam}(S_m)}{2} \leqslant r < \mathrm{diam}(S_m)$. Then $D(p,r)$ contains more than $1/4$ of the area of S_m, and so

$$\mathrm{area}(D(p,r) \sqcap P_m) > \frac{1}{4} \cdot \left(\frac{1}{2^m}\right)^2 = \frac{1}{8\pi} \cdot \pi \left(\frac{\sqrt{2}}{2^m}\right)^2 > \frac{1}{8\pi} \cdot \mathrm{area}(D(p,r)).$$

- Case III: $r \geqslant \mathrm{diam}(S_m)$. Then there exists an ℓ with $0 \leqslant \ell < n$ such that $\mathrm{diam}(S_{\ell+1}) \leqslant r \leqslant \mathrm{diam}(S_\ell)$. Since $p \in S_{\ell+1}$, we thus have $S_{\ell+1} \subset D(p,r)$. Moreover, by construction, all points of P_m within $S_{\ell+1}$ are connected via a path that stays inside $S_{\ell+1}$. This means $S_{\ell+1} \subset D(p,r) \sqcap P_m$. Since $r \leqslant \mathrm{diam}(S_\ell)$, we also have $\mathrm{area}(D(p,r)) \leqslant \pi \left(\frac{\sqrt{2}}{2^\ell}\right)^2$. Hence,

$$\mathrm{area}(D(p,r) \sqcap P_m) \geqslant \left(\frac{1}{2^{\ell+1}}\right)^2 = \frac{1}{8\pi} \cdot \pi \left(\frac{\sqrt{2}}{2^\ell}\right)^2 \geqslant \frac{1}{8\pi} \cdot \mathrm{area}(D(p,r)),$$

  which concludes the proof. ◀

One may think that we now only need to show that P_m does not have bounded doubling dimension but, even though $\lim_{m\to\infty} \frac{\mathrm{diam_g}(P_m)}{\mathrm{diam}(P_m)} = \infty$, this is not the case.

▶ **Lemma 3.** *The doubling dimension of the polygon P_m is $O(1)$.*

Proof. As in the proof of Lemma 2, we will assume that ε is infinitesimally small. It will also be convenient to scale the polygon P_m by a factor 2^m. Thus P_m consists of $2^m \times 2^m$ cells that are unit squares. We number these cells $s_1, \ldots, s_k$, where $k = 4^m$, in the order in which they are visited by the shortest path from the lower-left corner of P_m to its lower-right corner. Observe that the only way to enter a cell s_i is from the previous cell s_{i-1} or the next cell s_{i+1}.

Now consider a geodesic disk $D_g(p,r)$.

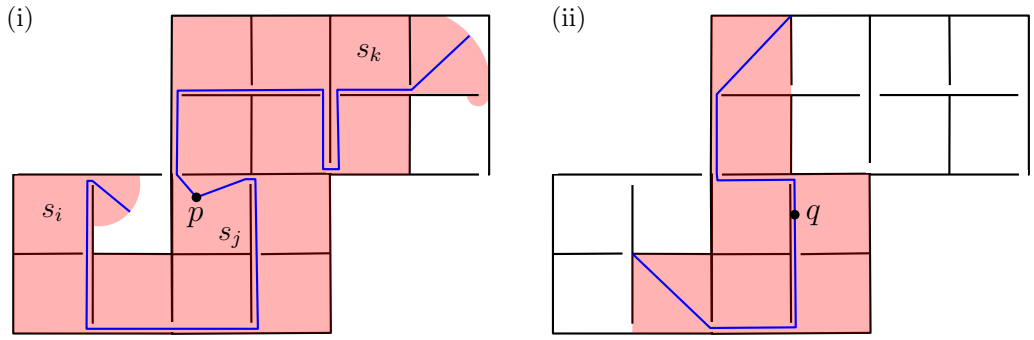


Figure 3 Illustration for the proof of Lemma 3. Note that only the relevant part of P_m is shown.

We must show that $D_g(p,r)$ can be covered by $O(1)$ geodesic disks of radius $r/2$. Let s_j be the cell containing the point p. Then $D_g(p,r)$ consists of $z \geqslant 0$ consecutive cells $s_i, \ldots, s_{i+z-1}$ that are fully contained in $D_g(p,r)$ plus at most four cells that are partially contained in $D_g(p,r)$; see Figure 3 (i). Because any Euclidean disk can be covered by seven Euclidean disks of half its radius and the cells s_t are convex, we know that $D_g(p,r) \cap s_t$ can be covered by at most seven geodesic disks of radius $r/2$, for any cell s_t. Hence, the part of $D_g(p,r)$ inside the four partially covered cells can be covered by 28 geodesic disks of radius $r/2$.

Now consider the z cells $s_i, \ldots, s_{i+z-1}$ that are fully contained in $D_g(p,r)$. Assume that $z > 0$, otherwise we are done. Since $\mathrm{diam}_g(s_i \cup \cdots \cup s_{i+z-1}) \geqslant z$, we must have $r \geqslant \frac{z}{2} \geqslant \frac{1}{2}$. We now have two cases.

- If $z \leqslant 7$ then we can cover each of the z cells by seven geodesic disks of radius $r/2$, using at most 49 disks in total.
- If $z \geqslant 8$ then we partition the sequence $s_i, \ldots, s_{i+z-1}$ into eight subsequences, each consisting of at most $\lceil z/8 \rceil$ cells. Note that any sequence of z^* cells can be covered by a geodesic disk of radius $(z^*-2)/2 + \sqrt{2}$. This is done by placing such a disk at the midpoint q of a longest path between the first and last cell of that sequence; see Figure 3(ii). Hence, a sequence of $\lceil z/8 \rceil$ cells can be covered by a geodesic disk of radius

$$\frac{\lceil z/8 \rceil - 2}{2} + \sqrt{2} < \frac{z}{16} + (\sqrt{2} - \tfrac{1}{2}) \leqslant \frac{z}{16} + \frac{z}{8} < \frac{z}{4} \leqslant r/2$$

Thus, in this case we use eight disks to cover the full sequence of z cells.

We conclude that $D_g(p,r)$ can be covered by at most 28+49 disks of radius $r/2$. Hence, the doubling constant is at most 77, and the doubling dimension is at most $\log 77 \approx 6.27$. ◀

The lemma above shows that by itself, the polygon P_m does not show that there are locally-fat polygons with unbounded doubling dimension. However, we can glue copies of P_m together to form a locally-fat polygon with unbounded doubling dimension.

▶ **Theorem 4.** *For any $n \geqslant 3$ there exists a simple polygon P with n vertices that is locally fat and whose doubling constant is $\Omega(n^{1/3})$.*

Proof. Let $Q_1, Q_2, \ldots, Q_k$ be $k := 2^m$ copies of P_m scaled by a factor of $\frac{1}{2^m}$. By Observation 1 each Q_i has geodesic diameter 1. We place the Q_i next to each other such that the right side of Q_i coincides with the left side of Q_{i+1}, for $1 \leqslant i \leqslant k-1$; see Figure 4. We make a small opening at the lower-left corner of each Q_i. Let u be the lower-left corner of Q_1 and v be the lower-right corner Q_k. Clearly, $|uv| = 1$. Consider the point c below uv such that

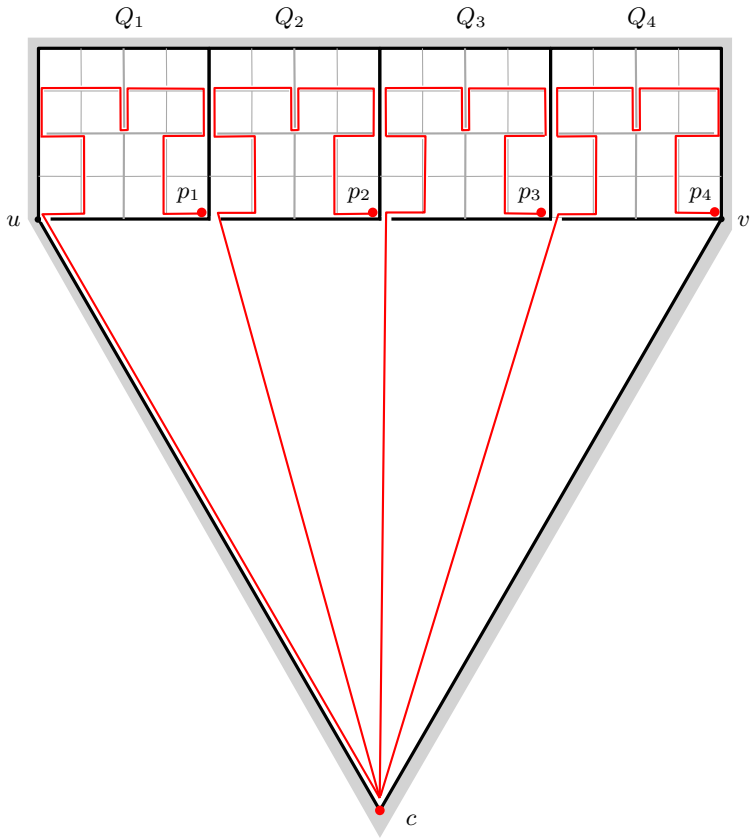


Figure 4 A chain of scaled copies $Q_1, \ldots, Q_{2^m}$ of the polygon P_m – in this example $m = 2$ – and the final polygon P_m^* obtained by attaching an equilateral triangle with apex c. The red curves show the paths from c (slightly displaced for clarity) to the points p_i.

$\triangle cuv$ is equilateral. The polygons $Q_1, \ldots, Q_k$, together with the segments cu and cv define a polygon, which we denote by P_m^*. Since the Q_i are locally fat, the polygon P_m^* is fat as well. However, the geodesic disk $D(c, 2)$ cannot be covered by a constant number of geodesic disks of radius 1. Indeed, if p_i is the point of Q_i at furthest geodesic distance from c, then any two p_i, p_j have geodesic distance $\text{dist}_g(p_i, p_j) > 2$ and can thus never be covered by the same geodesic disk of radius one. Hence, the doubling constant of P_m^* is at least 2^m.

Note that each Q_i has $O(4^m)$ vertices. Hence, if n denotes the number of vertices of P_m^* then $n = O(8^m)$. Thus, the doubling constant of P_m^* is $\Omega(n^{1/3})$. ◀

2.2 (α, β)-covered domains have bounded doubling dimension

Recall that a domain P, possibly with holes and a curved boundary, is (α, β)-covered if for every $p \in \partial P$ there exists a triangle $T_p \subset P$ with p as a vertex such that T_p is α-fat – each angle is at least α – and such that each side of T_p has length at least $\beta \cdot \text{diam}(P)$.

▶ **Theorem 5.** *Any (α, β)-covered domain P has bounded doubling dimension. In particular, it is at most $\log c(\alpha, \beta)$, where $c(\alpha, \beta) := \max\left(\left(\lceil \frac{48}{\sin\alpha} \rceil + 1\right)^2, \left(\left\lceil \frac{16}{\beta \sin\alpha} \right\rceil + 1\right)^2\right)$.*

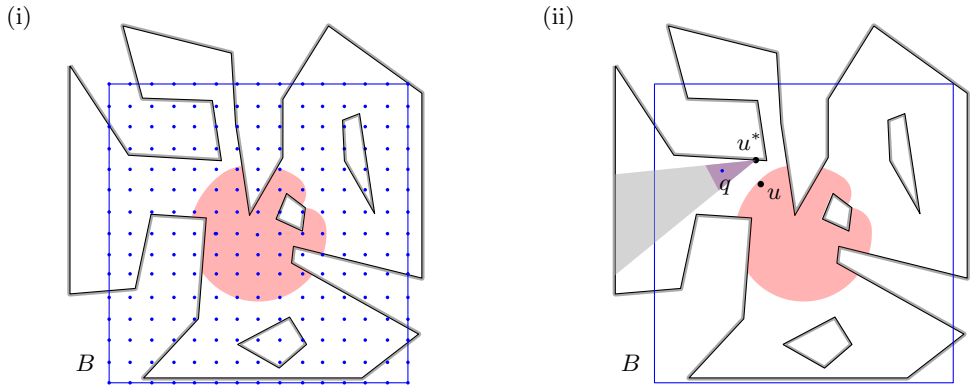


Figure 5 Construction of the grid G. The geodesic disk D is shown in red. The figure is schematic and not to scale.

Proof. Consider a geodesic disk $D := D_g(p, r)$ with center p and radius r in P. Let B be the square of side $3r$ centered at p. We partition B into a regular square grid G with $g \times g$ cells, where $g := \max(\lceil \frac{48}{\sin\alpha} \rceil, \lceil \frac{16}{\beta \sin\alpha} \rceil)$; see Figure 5. Let s denote the side length of a cell. Then we can see that $s = \frac{3r}{g} \leqslant \frac{m \cdot \sin\alpha}{16}$, where $m := \min(r, \beta \cdot \mathrm{diam}(P))$.

Now consider the set $\mathcal{D} = \{D_g(q, r/2) : q \in G \cap P\}$ of geodesic balls of radius $r/2$ centered at the grid points inside P. Note that $|\mathcal{D}| \leqslant c(\alpha, \beta)$. To prove Theorem 5, it suffices to show that the disks in $\mathcal{D}$ cover D. In other words, we need to show that for any point $u \in D$ there exists a grid point $q \in G \cap P$ with $\mathrm{dist}_g(u, q) \leqslant r/2$. We need the following well-known fact. For completenes, we give a proof.

▶ **Fact 6.** *Let G be a regular grid in $\mathbb{R}^2$ whose cells have side length x. Let T be an isosceles triangle whose two equal sides have length t and meet at angle ϕ. If $t \geqslant \frac{4x}{\sin\phi}$, then T contains a grid point of G.*

Proof. Every point of $\mathbb{R}^2$ is within Euclidean distance x – in fact, within distance $\frac{\sqrt{2}}{2}x$ – of some grid point. It therefore suffices to show that r_{in}, the radius of the incircle of T, is at least x. We have $r_{\mathrm{in}} = \frac{2 \cdot \mathrm{area}(T)}{\mathrm{per}(T)}$. Using $\mathrm{area}(T) = \frac{1}{2}t^2 \sin\phi$ and $\mathrm{per}(T) = 2t + 2t\sin(\phi/2)$ we obtain

$$r_{\mathrm{in}} = \frac{t^2 \sin\phi}{2t(1 + \sin(\phi/2))} \geqslant \frac{t \sin\phi}{4}.$$

Thus, if $t \geqslant 4x/\sin\phi$ then $r_{\mathrm{in}} \geqslant x$, which finishes the proof. ◀

We now consider two cases.

- *Case 1: $D(u, r/4) \subset P$.* Since $s \leqslant r \cdot (\sin\alpha/16) \leqslant r/4$, the disk $D(u, r/4)$ contains a grid point q. Since $D(u, r/4) \subset P$ we therefore have $\mathrm{dist}_g(u, q) = |uq| \leqslant r/4 \leqslant r/2$.
- *Case 2: $D(u, r/4)$ intersects ∂P.* This case is illustrated in Figure 5(ii). Let $u^* \in \partial P$ be a point closest to u. Then $|uu^*| < r/4$ and $uu^* \subset P$. Let $T_{u^*} \subset P$ be an α-fat triangle with vertex u^* and all of whose edges have length at least $\beta \cdot \mathrm{diam}(P)$, which is guaranteed to exist because P is (α, β)-covered.

  If $r < \beta \cdot \mathrm{diam}(P)$ then every side of T_{u^*} has length at least r. Thus, T_{u^*} contains an isosceles α-fat triangle T with apex u^* and side length $t = r/4$. Because $T \subset P \cap B$ and $r/4 = m/4 \geqslant 4s/\sin\alpha$, the triangle T contains a grid point q by Fact 6. Since $|u^*q| \leqslant r/4$, we thus have

$$\mathrm{dist}_g(u, q) \leqslant |uu^*| + |u^*q| \leqslant \frac{r}{4} + \frac{r}{4} = \frac{r}{2}.$$

On the other hand, if $r \geqslant \beta \cdot \text{diam}(P)$ then T_{u^*} contains an isosceles α-fat triangle with side length $t' = \beta \cdot \text{diam}(P)/4 = m/4 \geqslant 4s/\sin\alpha$. Again by Fact 6 it contains a grid point q', and since $t' \leqslant r/4$ we also have $\text{dist}_g(u, q') \leqslant r/2$.

We conclude that every $u \in D$ lies in some disk of $\mathcal{D}$, which finishes the proof of Theorem 5. ◀

Applications. The fact that (α, β)-covered domains have bounded doubling dimension immediately gives a plethora of results that improve on the state-of-the-art for arbitrary (non-fat) domains. We mention results on spanners and WSPDs, because they form the basis of many other results.

▶ **Corollary 7.** *Let P be an (α, β)-covered domain and let $S \subset P$ be a set of m points.*

(i) *Let $G = (S, E)$ be the complete graph on S such that the weight of an edge $(p, q) \in S \times S$ is its geodesic distance $\text{dist}_g(p, q)$. Then there exists a $(1 + \varepsilon)$-spanner of G consisting of $m(1/\varepsilon)^{O(1)}$ edges.*

(ii) *For any fixed $s > 1$, there exists an s-well seperated pair decomposition (WSPD) of S of size $m \cdot s^{O(1)}$.*

Proof. Part (i) follows from the result of Gao, Guibas, and Nguyen [17], who showed that a set of m points in a space of doubling dimension *dim* has a $(1+\varepsilon)$-spanner of size $m(1/\varepsilon)^{O(dim)}$ edges. For part (ii), we use the WSPD construction by Har-peled and Mendel [19], which has size $m \cdot s^{O(dim)}$. ◀

Note that the bound on the spanner size in part (i) of Corollary 7 is linear in m and does not depend on the size of the polygon. A similar result is not possible for non-fat polygons: for any $\varepsilon > 0$ and any m there is a polygon P and a set of m points such that any spanner of subquadratic size has spanning ratio at least $2 - \varepsilon$. (Take the polygon of Figure 1(i) and put a point in each of the spikes.) Also note that there are more results on spanners in spaces of bounded doubling dimension – for example on the lightness of the spanner [4] – and that any such result immediately applies to point sets in an (α, β)-covered polygonal domain.

3 The perimeter of geodesically convex regions in fat polygonal domains

Recall that Bose et al. [5] showed that for any (α, β)-covered simple polygon P there exists a constant $\mu(\alpha, \beta)$, depending only on α and β, such that $\text{per}(P) \leqslant \mu(\alpha, \beta) \cdot \text{diam}(P)$. Theorem 8 generalizes this result to arbitrary geodesically convex polygonal sets in P, and extends it from simple polygons to general polygonal domains. As we will see in the next section, this is useful in several applications. Note that we cannot hope to generalize the result to locally-fat polygons. Indeed, there are locally-fat polygons for which we do not even have $\text{per}(P) = O(\text{diam}(P))$ – the polygon P_n from Section 2 is an example.

▶ **Theorem 8.** *Let P be an (α, β)-covered polygonal domain, possibly with holes. Let $R \subseteq P$ be a geodesically convex set in P such that $\partial R \setminus \partial P$ is polygonal. Then there exists a constant $\nu(\alpha, \beta)$ such that $\text{per}(R) \leqslant \nu(\alpha, \beta) \cdot \text{diam}(R)$.*

To prove Theorem 8, define $\Gamma_1 := \partial R \cap \partial P$ and $\Gamma_2 := \partial R \setminus \Gamma_1$, and note that $\text{per}(R) = \|\Gamma_1\| + \|\Gamma_2\|$. Refer to Figure 6 for an illustration of these sets. We will bound $\|\Gamma_1\|$ and $\|\Gamma_2\|$ separately.

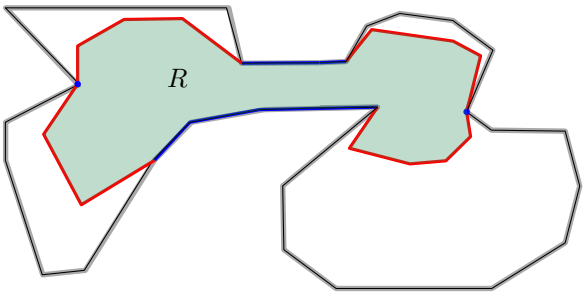


Figure 6 The shaded region R is geodesically convex. The blue curves (and points) are the connected components of Γ_1, while the red curves are the connected components of Γ_2.

Bounding $\|\Gamma_1\|$. Let S be an axis-aligned square of side length $\mathrm{diam}(R)$ that contains R. We partition S into a regular $g \times g$ grid $\mathcal{G}$, where $g := \left\lceil \frac{2}{\beta \sin\alpha} \right\rceil$. Let $s := \mathrm{diam}(R)/g$ be the side length of a grid cell. Then $s \leqslant \frac{\beta \sin\alpha}{2} \cdot \mathrm{diam}(R)$, and each cell $C \in \mathcal{G}$ satisfies

$$\mathrm{diam}(C) = \sqrt{2}\, s < \beta \sin\alpha \cdot \mathrm{diam}(R). \tag{1}$$

Moreover,

$$\sum_{C \in \mathcal{G}} \mathrm{per}(C) \;=\; g^2 \cdot 4s \;=\; 4g \cdot \mathrm{diam}(R) \;=\; O\Big(\tfrac{1}{\alpha\beta} \cdot \mathrm{diam}(R)\Big). \tag{2}$$

We will need the following lemma.

▶ **Lemma 9.** *Let $p \in \partial P \cap C$ for some cell $C \in \mathcal{G}$. Then any ray ρ emanating from p and going into the witness triangle T_p, will hit ∂C before exiting T_p.*

Proof. It suffices to prove that the side e of T_p opposite to p lies fully outside C. Assume for a contradiction that there exists a point $q \in e$ that is contained in C. Since $p \in C$ and $q \in C$, we have that $\mathrm{diam}(C) \geqslant |pq|$. Moreover, $|pq| \geqslant h_p$, where h_p is the height of T_p from p. Since $h_p \geqslant \sin\alpha \cdot \beta \cdot \mathrm{diam}(R)$, we have

$$\mathrm{diam}(C) \geqslant |pq| \geqslant h_p \geqslant \sin\alpha \cdot \beta \cdot \mathrm{diam}(R),$$

which contradicts Inequality (1). ◀

Let $\mathcal{D} = \{d_i := i\frac{\alpha}{4} \mid i \in \mathbb{Z} \text{ and } 0 \leqslant i \leqslant \frac{8\pi}{\alpha}\}$ be a set of *canonical directions*.[1] For a point $p \in P$ and a direction d, let $\rho(p, d)$ be the ray starting from p and going into the direction d. Let C be a cell in $\mathcal{G}$ and consider a point $p \in \partial P \cap C$ that lies in the relative interior of an edge e of ∂P. We say a direction $d_i \in \mathcal{D}$ is *good* for p if the following holds for the ray $\rho(p, d_i)$:

(i) $\rho(p, d_i)$ makes an angle of at least $\alpha/4$ with the edge e, and
(ii) $\rho(p, d_i)$ hits ∂C before hitting ∂P.

The following lemma will allow us to bound $\|\Gamma_1\|$.

▶ **Lemma 10.** *Let C be a cell in $\mathcal{G}$ and let $X \subset \partial P \cap C$ be a point set that consists of finitely many connected components. Assume that every $p \in X$ that is not a vertex of P, has at least one good direction in $\mathcal{D}$. Then*

$$\|X\| \leqslant c(\alpha) \cdot \mathrm{per}(C) \qquad \textit{where} \qquad c(\alpha) := \tfrac{8\pi}{\alpha \sin(\alpha/4)}.$$

[1] With a slight abuse of terminology, we identify an angle $d_i \in \mathcal{D}$ with the direction of a vector whose counterclockwise angle with the positive x-axis is d_i.

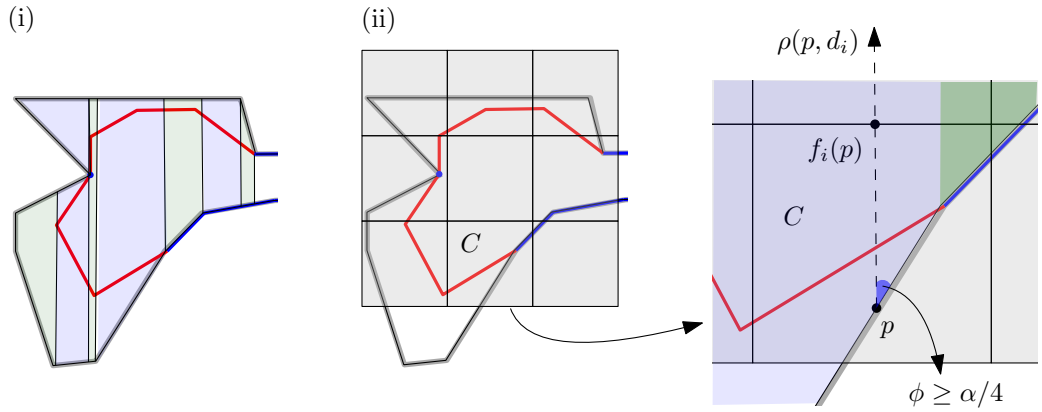


Figure 7 (i) The vertical decomposition of part of the polygon from Figure 6. (ii) Part of the grid $\mathcal{G}$. In the cell C, the direction d_i is good for p (and in this example it is good for all points of $\partial P \cap C$).

Proof. For each $d_i \in \mathcal{D}$, let $X_i \subseteq X$ be the set of points where d_i is good. Note that $\|X\| \leqslant \sum_{d_i \in \mathcal{D}} \|X_i\|$.

Fix a direction d_i and consider X_i. Assume wlog that d_i is the vertically upward direction. We define a mapping $f_i : X_i \to \partial C$ such that $f_i(p)$ is the point where $\rho(p, d_i)$ hits ∂C; see Figure 7.

Observe that f_i is injective. Indeed, if $q = f_i(p)$ then p is the unique point in X_i that is hit by a vertically downward ray from q. Let $\text{VD}(P)$ be the vertical decomposition of P, and consider a trapezoid $\Delta \in \text{VD}(P)$. Let $X_i(\Delta)$ be the part of X_i contained in the bottom side of Δ – the top side cannot contain any part of X_i – and assume that $X_i(\Delta) \neq \emptyset$. Observe that $f_i(p) \in \Delta \cap \partial C$ for all $p \in X_i(\Delta)$. Since the angle between $\rho(p, d_i)$ and the bottom side of Δ is at least $\alpha/4$, we thus have $\|X_i(\Delta)\| \leqslant \frac{1}{\sin(\alpha/4)} \cdot \|\partial C \cap \Delta\|$. Hence,

$$\|X_i\| = \sum_{\Delta} \|X_i(\Delta)\| \leqslant \tfrac{1}{\sin(\alpha/4)} \cdot \|\partial C \cap \Delta\| = \tfrac{1}{\sin(\alpha/4)} \cdot \|\partial C\|.$$

Finally, we bound $\|X\|$ by summing over all canonical directions:

$$\|X\| \leqslant \sum_{d_i \in \mathcal{D}} \|X_i\| \leqslant |\mathcal{D}| \cdot \tfrac{1}{\sin(\alpha/4)} \cdot \|\partial C\| \leqslant \tfrac{8\pi}{\alpha \sin(\alpha/4)} \cdot \|\partial C\|.$$

This proves the lemma. ◀

We can now bound $\|\partial P \cap S\|$. Note that this immediately implies the same bound for $\|\Gamma_1\|$, since $\Gamma_1 \subseteq \partial P \cap S$.

▶ **Lemma 11.** $\|\partial P \cap S\| = O\Big(\frac{1}{\beta\alpha^3} \cdot \text{diam}(R)\Big)$.

Proof. Fix a grid cell $C \in \mathcal{G}$ and consider $X := \partial P \cap C$. Let $p \in X$ be a point that lies in the relative interior of an edge e of ∂P, and let T_p be its witness triangle. Since the angle of T_p at p is at least α, there are at least three canonical directions d_{i-1}, d_i, d_{i+1} such that the rays $\rho(p, d_{i-1})$, $\rho(p, d_i)$, and $\rho(p, d_{i+1})$ go into T_p. The ray $\rho(p, d_i)$ must therefore make an angle of at least $\alpha/4$ with the edge e. Lemma 9 implies that the ray $\rho(p, d_i)$ will hit ∂C before exiting T_p. As a result, the direction d_i is good for p. Hence, we can apply Lemma 10 to X and conclude that

$$\|\partial P \cap C\| = \|X\| \leqslant \tfrac{8\pi}{\alpha \sin(\alpha/4)} \cdot \text{per}(C).$$

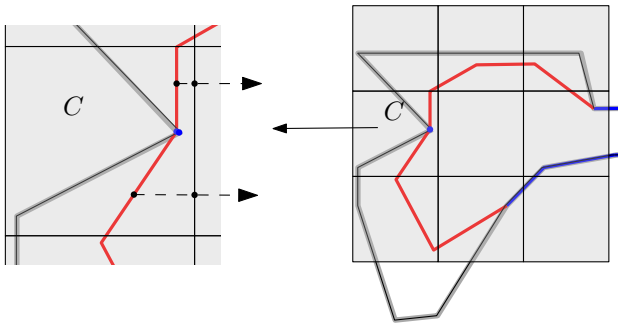


Figure 8 The two red edges in cell C bound R from the left and are mostly vertical. The two shown horizontal rays hit ∂C before hitting ∂P.

Summing over all cells $C \in \mathcal{G}$ and using Equation (2) gives

$$\|\partial P \cap S\| \leqslant \tfrac{8\pi}{\alpha \sin(\alpha/4)} \cdot \sum_{C \in G} \mathrm{per}(C) = O\Big(\tfrac{1}{\beta\alpha^3} \cdot \mathrm{diam}(R)\Big). \qquad \blacktriangleleft$$

As mentioned, the fact that $\Gamma_1 \subset \partial P \cap S$ immediately gives us the following corollary.

▶ **Corollary 12.** *The total length of* Γ_1 *is* $O(\frac{1}{\beta\alpha^3} \cdot \mathrm{diam}(R))$.

Bounding $\|\Gamma_2\|$. We now bound the contribution to the perimeter coming from Γ_2, the part of ∂R that lies in the interior of P.

For a grid cell $C \in \mathcal{G}$, let $E_{\mathrm{v}}(C)$ be the set of edges of $\Gamma_2 \cap C$ that are *mostly vertical*, in the sense that their angle with the x-axis is at least $\pi/4$. Define $E_{\mathrm{h}}(C)$ analogously for *mostly horizontal* edges. Note that any edge is mostly vertical or mostly horizontal (or both, when it's slope is exactly 1).

▶ **Lemma 13.** *For any cell* $C \in \mathcal{G}$ *we have*

$$\|\Gamma_2 \cap C\| = O\big(\mathrm{per}(C) + \|\partial P \cap C\|\big).$$

Proof. We prove the bound for $E_{\mathrm{v}}(C)$; the bound for $E_{\mathrm{h}}(C)$ is symmetric. Since $\|\Gamma_2 \cap C\| = \|E_{\mathrm{v}}(C)\| + \|E_{\mathrm{h}}(C)\|$, the lemma then follows.

Consider the edges of $E_{\mathrm{v}}(C)$ that bound R from the left, that is, the interior of R lies locally to their right. Let p be a point on such an edge and shoot a ray ρ from p horizontally to the right. We claim that ρ cannot hit another edge of $E_{\mathrm{v}}(C)$ that bounds R from the left before hitting ∂P, and thus it either hits ∂P first or exits C.

Indeed, if ρ would hit another such edge at a point q before hitting ∂P, then the horizontal segment pq is contained in P. Hence, pq is a shortest path in P between p and q, and because R is geodesically convex, we must have $pq \subseteq R$. This contradicts that q lies on an edge that bounds R from the left.

We now charge any such point p on a left-bounding edge to where the ray ρ hits ∂P (if it hits ∂P first) or to where it exits C (if it does not hit ∂P inside C); see Figure 8. This is analogous to the method used in Lemma 10: since we consider mostly vertical edges, we know that ρ forms an angle of at least $\pi/4$ with the corresponding edge of ∂P (whereas in Lemma 10 this bound was $\alpha/4$). Therefore, by using a horizontal decomposition of P, we can similarly argue that the total length charged to ∂P is $O(\|\partial P \cap C\|)$ and the total length charged to ∂C is $O(\mathrm{per}(C))$. In total we have

$$\|E_{\mathrm{v}}(C)\| = O\big(\mathrm{per}(C) + \|\partial P \cap C\|\big).$$

A similar argument applies to the edges of $E_{\mathrm{v}}(C)$ that bound R from the right. $\blacktriangleleft$

Summing the bound from Lemma 13 over all grid cells yields

$$\|\Gamma_2\| = \sum_{C\in\mathcal{G}} \|\Gamma_2 \cap C\| = O\left(\sum_{C\in\mathcal{G}} \mathrm{per}(C) + \|\partial P \cap S\|\right) = O\left(\frac{1}{\beta\alpha^3}\cdot \mathrm{diam}(R)\right),$$

where the last step uses Lemma 11 and the bound on $\sum_{C\in\mathcal{G}} \mathrm{per}(C)$ given in Equation (2).

Since $\mathrm{per}(R) = \|\Gamma_1\| + \|\Gamma_2\|$, we conclude that

$$\mathrm{per}(R) = O\left(\tfrac{1}{\beta\alpha^3}\cdot \mathrm{diam}(R)\right).$$

This proves Theorem 8 with $\nu(\alpha,\beta) = O\left(\frac{1}{\beta\alpha^3}\right)$.

Application to coresets for furthest-neighbor queries. Let S be a point set in a simple polygon P. Recall that an ε-coreset of S for furthest-neighbor queries is a set $C \subseteq S$ such that for any query point $q \in P$ we have $\mathrm{dist}_g(q, \mathrm{FN}(q,C)) \geqslant (1-\varepsilon)\cdot \mathrm{dist}_g(q, \mathrm{FN}(q,S))$, where $\mathrm{FN}(p,Q)$ denotes the furthest neighbor of q in a set Q. By applying Theorem 8 for $R = \mathrm{RCH}(S)$, we obtain the following result.

▶ **Corollary 14.** *Let S be a point set of size m in an (α,β)-covered simple polygon P with n vertices. For any $0 < \varepsilon \leqslant 1$, there exists an ε-coreset $C \subseteq S$ of size $O(1/\varepsilon)$ for furthest-neighbor queries. The coreset C can be constructed in $O(n + m\log(n+m))$ time.*

Proof. For any query point q, its furthest neighbor $\mathrm{FN}(q,S)$ is a vertex of the relative convex hull $\mathrm{RCH}(S)$ of S inside P [12]. Moreover, $\mathrm{dist}_g(q, \mathrm{FN}(q,S)) \geqslant \frac{1}{2}\mathrm{diam}_g(\mathrm{RCH}(S))$. Hence, we can compute an ε-coreset as follows. First, we compute $\mathrm{RCH}(S)$ in $O(n + m\log(n+m))$ time [24, 6, 18]. Then we traverse $\partial\,\mathrm{RCH}(S)$ to select a subset C of $O(1/\varepsilon)$ vertices of $\partial\,\mathrm{RCH}(S)$ such that for any vertex p' of $\mathrm{RCH}(S)$, there is a vertex $p \in C$ whose distance to p' along $\partial\,\mathrm{RCH}(S)$ is at most $\frac{\varepsilon}{2\nu(\alpha,\beta)}\cdot \mathrm{per}(\mathrm{RCH}(S))$, where $\nu(\alpha,\beta)$ is the constant in Theorem 8.

To see that C is an ε-coreset, recall that $\mathrm{per}(\mathrm{RCH}(S)) \leqslant \nu(\alpha,\beta)\cdot \mathrm{diam}(\mathrm{RCH}(S))$ by Theorem 8. Now consider a query point q. Let p be a point in C whose distance to $\mathrm{FN}(q,S)$ along $\partial\,\mathrm{RCH}(S)$ is at most $\frac{\varepsilon}{2\nu(\alpha,\beta)}\cdot \mathrm{per}(\mathrm{RCH}(S))$. Then $\mathrm{dist}_g(p, \mathrm{FN}(q,S)) \leqslant \frac{\varepsilon}{2\nu(\alpha,\beta)}\cdot \mathrm{per}(\mathrm{RCH}(S))$, and so

$$\begin{aligned}
\mathrm{dist}_g(q, \mathrm{FN}(q,S)) &\leqslant \mathrm{dist}_g(q,p) + \mathrm{dist}_g(p, \mathrm{FN}(q,S)) \\
&\leqslant \mathrm{dist}_g(q, \mathrm{FN}(q,C)) + \tfrac{\varepsilon}{2\nu(\alpha,\beta)}\cdot \mathrm{per}(\mathrm{RCH}(S)) \\
&\leqslant \mathrm{dist}_g(q, \mathrm{FN}(q,C)) + \varepsilon\cdot\tfrac{1}{2}\cdot \mathrm{diam}(\mathrm{RCH}(S)) \\
&\leqslant \mathrm{dist}_g(q, \mathrm{FN}(q,C)) + \varepsilon\cdot \mathrm{dist}_g(q, \mathrm{FN}(q,S)),
\end{aligned}$$

which proves that C is an ε-coreset. ◀

4 Closest pair in an (α,β)-covered polygon

Let Q be a set of m points contained in an (α,β)-covered polygon P. The closest-pair problem asks for a pair of points $p,q \in Q$ minimizing the geodesic distance $\mathrm{dist}_g(p,q)$. Using Theorem 8 we prove the following lemma, which will allow us to adapt the classical linear-time closest-pair algorithm for the plane [14] to the geodesic metric in P.

▶ **Lemma 15.** *Let P be an (α,β)-covered simple polygon, and let S be a square that intersects the interior of P. Let $C_1,\ldots,C_m$ be the connected components of $S \cap P$. Then there exists a partition of $\{C_1,\ldots,C_m\}$ into $t = O(1)$ classes $\mathcal{C}_1,\ldots,\mathcal{C}_t$ such that the following holds:*

There is constant M such that for every class $\mathcal{C}_j$ and every pair of components $C_a, C_b \in \mathcal{C}_j$ we have $\mathrm{dist}_g(x,y) \leqslant M\cdot \mathrm{diam}(S)$ for all points $x \in C_a$ and $y \in C_b$.

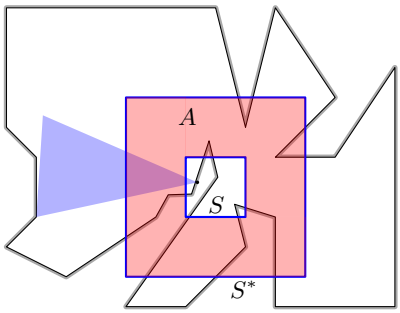


Figure 9 Illustration for the proof of Lemma 15.

Proof. If $S \subset P$ then $m = 1$ and the statement is trivial since any two points $x, y \in S \cap P$ can be connected by a segment of length at most $\mathrm{diam}(S)$. Hence, we assume that ∂P intersects the interior of S. Let ℓ be the side-length of S. We have two cases.

Case 1: $\ell \geqslant \frac{\beta \cdot \mathrm{diam}(P)}{8}$. Then, for any $x, y \in S \cap P$,

$$\mathrm{dist}_{\mathrm{g}}(x, y) \leqslant \mathrm{diam}_{\mathrm{g}}(P) \leqslant \mathrm{per}(P) \leqslant \mu(\alpha, \beta) \cdot \mathrm{diam}(P) \leqslant \tfrac{8\mu(\alpha,\beta)}{\beta} \cdot \ell \leqslant \tfrac{8\mu(\alpha,\beta)}{\beta} \cdot \mathrm{diam}(S).$$

So in this case we may put all components C_i into a single class and the lemma holds.

Case 2: $\ell < \frac{\beta \cdot \mathrm{diam}(P)}{8}$. Observe that any connected component C_i of $S \cap P$, is a geodesically convex region. Indeed, if there were $x, y \in C_i$ such that a shortest path $\pi(x, y)$ leaves C_i, then $\pi(x, y)$ must exit S at a point a and re-enter S at a point b. Let $\pi(a, b) \subset \pi(x, y)$ be the subpath of $\pi(x, y)$ from a to b. Since C_i is connected, there is a path $\pi'(a, b) \subset C_i$. Since P is a simple polygon, the cycle $\pi'(a, b) \cup \pi(a, b)$ does not contain a part of ∂P. But then we can replace $\pi(a, b)$ by the part of ∂S that lies inside the cycle, and obtain a shorter path from x to y, which is a contradiction. Hence, C_i must be a geodesically convex, as claimed. Applying Theorem 8 with $R := C_i$ we have $\mathrm{per}(C_i) \leqslant \nu(\alpha, \beta) \cdot \mathrm{diam}(C_i)$. Since $C_i \subseteq S$, we have $\mathrm{diam}(C_i) \leqslant \mathrm{diam}(S)$, and hence for any $x, y \in C_i$

$$\mathrm{dist}_{\mathrm{g}}(x, y) \leqslant \mathrm{per}(C_i) \leqslant \nu(\alpha, \beta) \cdot \mathrm{diam}(C_i) \leqslant \nu(\alpha, \beta) \cdot \mathrm{diam}(S). \tag{3}$$

Since ∂P meets the interior of S, every component C_i satisfies $C_i \cap \partial P \neq \emptyset$. For each i choose a point $p_i \in C_i \cap \partial P$ and let T_i be its witness triangle. Let S^* be the square with the same center as S and with side length 3ℓ, and let $A := S^* \setminus \mathrm{int}(S)$ be the annulus between the two squares. The maximum Euclidean distance from a point in S to a point in A is $2 \cdot \mathrm{diam}(S)$; see Figure 9. An edge of T_i incident to p_i has length at least $\beta \cdot \mathrm{diam}(P) \geqslant 8\ell \geqslant 4 \cdot \mathrm{diam}(S)$. This implies that the two sides of T_i incident to p_i exit S^*, and because the angle between them is at least α we can conclude that $\mathrm{area}(T_i \cap A) = \Omega(\mathrm{area}(A))$. This implies that we can stab the collection of objects $\{T_i \cap A\}_{i=1}^{m}$, with a constant number of points $q_1, q_2, ..., q_t$. Our classes $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_t$ are now defined by assigning each component C_j to the class $\mathcal{C}_i$ such that i is the smallest index for which q_i stabs $T_j \cap A$.

Now fix a class $\mathcal{C}_j$ and components $C_a, C_b \in \mathcal{C}_j$. Let $x \in C_a$ and $y \in C_b$. To bound $\mathrm{dist}_{\mathrm{g}}(x, y)$, consider the path $\pi = \pi(x, p_a) \cup p_a q_j \cup q_j p_b \cup \pi(p_b, y)$. By Inequality (3) we have

$$\|\pi(x, p_a)\| + \|\pi(p_b, y)\| = \mathrm{dist}_{\mathrm{g}}(x, p_a) + \mathrm{dist}_{\mathrm{g}}(y, p_b) = 2\nu(\alpha, \beta) \cdot \mathrm{diam}(S).$$

Moreover the points p_a, p_b, q_j lie in S^*, which implies that $|p_a q_j| \leqslant \mathrm{diam}(S^*) = 3 \cdot \mathrm{diam}(S)$ and $|q_j p_b| \leqslant 3 \cdot \mathrm{diam}(S)$. Therefore $\mathrm{dist}_{\mathrm{g}}(x, y) \leqslant (2\nu(\alpha, \beta) + 6) \cdot \mathrm{diam}(S)$.

Combining the two cases, the lemma holds for all squares S. ◀

The algorithm. We adapt the algorithm by Dietzfelbinger et al. [14] that solves the problem in expected linear time in the Euclidean plane. We process the points of Q in random order, and maintain the current minimum distance δ and a grid of squares of side length $s := \delta/(2M)$, stored in a hash table. As long as δ is the smallest geodesic distance among the processed points, Lemma 15 implies that each grid cell contains at most $t = O(1)$ points. Indeed, if $|X(S)| > t$ then two points of $X(S)$ would lie in components belonging to the same class of Lemma 15, implying $\mathrm{dist}_g(x, y) \leqslant M \cdot \mathrm{diam}(S) = \frac{\sqrt{2}}{2}\delta < \delta$, a contradiction.

Now let $q_1, \ldots, q_m$ be the points from Q, in random order, and suppose that the insertion of some point q_j changes the closest pair. Let S be the cell of the current grid that contains the new point q_j, and let q_i, q_j be the new closest pair. Since $\mathrm{dist}_g(q_i, q_j) < \delta$ we know that q_i must lie in cell S' such that the Euclidean distance between S and S' is smaller than δ. Thus, to find q_i we can restrict our attention to the $O(M^2)$ cells within distance δ of S (including S itself). Since each cell contains at most t points, only $O(1)$ candidates must be checked, which we can do in $O(\log n)$ time using a shortest-path query. When a smaller distance is found, we update δ and rebuild the grid. The analysis of the expected running time is identical to the Euclidean case [14], except that computing the distance between two given points incurs a logarithmic overhead (after preprocessing P for shortest-path queries in $O(n)$ time [18]). We obtain the following theorem.

▶ **Theorem 16.** *Let P be an (α, β)-covered simple polygon with n vertices and let Q be a set of m points in P. A closest pair of Q under the geodesic distance can be found in expected time $O(n + \kappa(\alpha, \beta)m \log n)$, where $\kappa(\alpha, \beta)$ is a constant depending only on α and β.*

References

1. Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom.*, 5:187–206, 1995. doi:10.1016/0925-7721(95)00005-8.
2. Greg Aloupis, Prosenjit Bose, Vida Dujmovic, Chris Gray, Stefan Langerman, and Bettina Speckmann. Triangulating and guarding realistic polygons. *Comput. Geom.*, 47(2):296–306, 2014. doi:10.1016/J.COMGEO.2013.03.005.
3. Boris Aronov, Mark de Berg, Esther Ezra, and Micha Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM J. Comput.*, 43(2):543–572, 2014. doi:10.1137/120891241.
4. Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2371–2379, 2019. doi:10.1137/1.9781611975482.145.
5. Prosenjit Bose, Otfried Cheong, and Vida Dujmović. A note on the perimeter of fat objects. *Computational Geometry*, 44(1):1–8, 2011. doi:10.1016/j.comgeo.2010.06.002.
6. Bernard Chazelle. Triangulating a simple polygon in linear time. *Discret. Comput. Geom.*, 6:485–524, 1991. doi:10.1007/BF02574703.
7. Mark de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, 28(3):353–366, 2000. doi:10.1007/S004530010047.
8. Mark de Berg. Improved bounds on the union complexity of fat objects. *Discret. Comput. Geom.*, 40(1):127–140, 2008. doi:10.1007/S00454-007-9029-7.
9. Mark de Berg, Otfried Cheong, Herman J. Haverkort, Jung Gun Lim, and Laura Toma. The complexity of flow on fat terrains and its i/o-efficient computation. *Comput. Geom.*, 43(4):331–356, 2010. doi:10.1016/J.COMGEO.2008.12.008.
10. Mark de Berg and Chris Gray. Vertical ray shooting and computing depth orders for fat objects. *SIAM J. Comput.*, 38(1):257–275, 2008. doi:10.1137/060672261.

11 Mark de Berg and Chris Gray. Computing the visibility map of fat objects. *Comput. Geom.*, 43(4):410–418, 2010. doi:10.1016/J.COMGEO.2008.12.010.

12 Mark de Berg and Leonidas Theocharous. A coreset for approximate furthest-neighbor queries in a simple polygon. In *40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 16:1–16:16, 2024. doi:10.4230/LIPIcs.SOCG.2024.16.

13 Mark de Berg, A. Frank van der Stappen, Jules Vleugels, and Matthew J. Katz. Realistic input models for geometric algorithms. *Algorithmica*, 34(1):81–97, 2002. doi:10.1007/S00453-002-0961-X.

14 Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997. doi:10.1006/jagm.1997.0873.

15 Alon Efrat. The complexity of the union of (alpha, beta)-covered objects. *SIAM J. Comput.*, 34(4):775–787, 2005. doi:10.1137/S0097539702407515.

16 Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Comput. Geom.*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0.

17 Jie Gao, Leonidas J. Guibas, and An Thai Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006. doi:10.1016/J.COMGEO.2005.10.001.

18 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.

19 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

20 Matthew J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Comput. Geom.*, 8:299–316, 1997. doi:10.1016/S0925-7721(96)00027-2.

21 Matthew J. Katz, Mark H. Overmars, and Micha Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom.*, 2:223–234, 1992. doi:10.1016/0925-7721(92)90024-M.

22 Jirí Matousek, János Pach, Micha Sharir, Shmuel Sifrony, and Emo Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23(1):154–169, 1994. doi:10.1137/S009753979018330X.

23 Mark H. Overmars and A. Frank van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21(3):629–656, 1996. doi:10.1006/JAGM.1996.0063.

24 Godfried Toussaint. An optimal algorithm for computing the relative convex hull of a set of points in a polygon. In *Proceedings of EURASIP, Signal Processing III: Theories and Applications, Part 2*, volume 293, pages 853–856, 1986.

25 A. Frank van der Stappen. *Motion planning amidst fat obstacles.* PhD thesis, University Utrecht, 1994.

26 A. Frank van der Stappen, Dan Halperin, and Mark H. Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom.*, 3:353–373, 1993. doi:10.1016/0925-7721(93)90007-S.

27 A. Frank van der Stappen, Mark H. Overmars, Mark de Berg, and Jules Vleugels. Motion planning in environments with low obstacle density. *Discret. Comput. Geom.*, 20(4):561–587, 1998. doi:10.1007/PL00009402.

28 J. Vleugels. *On Fatness and Fitness—Realistic Input Models for Geometric Algorithms.* PhD thesis, University Utrecht, 1997.

Submodular Max-Min Allocation Under Identical Valuations

Kimon Boehmer
DIENS, École normale supérieure, PSL University, Paris, France
LIP, ENS Lyon, France

Abstract

In the problem of Submodular Max-Min Allocation, we are given a set of items, a set of players, and monotone submodular valuation functions that represent the satisfaction of a player with a certain subset of items. The goal is to find an allocation of the items to the players that maximizes the lowest satisfaction among all players.

We study this problem in the special case where all players have the same valuation function. We devise a greedy algorithm which gives a 0.4-approximation, improving the previously best factor of $\frac{10}{27} \approx 0.37$ by Uziahu and Feige.

Furthermore, we study the integrality gap of the *configuration LP* when players have identical valuations. By constructing a variable assignment to the dual from a primal integral solution, we give the first constant upper bound on the integrality gap for submodular valuations. Generalizing the result to the case where players' allocations must be independent in k given matroids, we derive a $\mathcal{O}(k)$-estimation algorithm for max-min allocation subject to k matroid constraints under identical valuations.

2012 ACM Subject Classification Theory of computation → Submodular optimization and polymatroids; Theory of computation → Linear programming; Theory of computation → Packing and covering problems

Keywords and phrases Submodularity, Approximation algorithms, Allocation, Configuration LP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.8

Related Version *Full Version*: https://arxiv.org/abs/2604.12417

Acknowledgements I want to thank Chien-Chung Huang for introducing me to the topic and for many helpful dicussions. Also, I thank the anonymous reviewers for their valuable suggestions.

1 General context

Item allocation is a fundamental problem in combinatorial optimization. Its task consists of distributing a set of *items* to a set of *players*, each equipped with a valuation function over subsets of items, with the goal of optimizing a specific objective function. The most well-studied objective functions are *max-sum*, i.e. maximizing the average satisfaction across all players, and *max-min*, maximizing the satisfaction of the least satisfied player. This work focuses on the latter objective function. Applications of max-min allocation range from assigning classrooms to charter schools [19] to sensor placement and scheduling [18].

Usually, research focuses on restricted classes of valuation functions that are both expressive and computationally tractable. For example, a common assumption is that each player has an *additive* valuation function; that is, each item is associated with a certain weight, and the satisfaction of a player is the sum of the weights of the received items. Another important and more general class of valuation functions are *submodular* functions; besides their useful algorithmic properties, they naturally model the law of *diminishing returns* in economics. This makes them especially interesting in the context of allocation.

© Kimon Boehmer;
licensed under Creative Commons License CC-BY 4.0

In this work, we will restrict our attention to the case where all players share the same submodular valuation function. It is interesting to note that the strongest hardness results of approximating submodular max-min allocation to a factor of $1 - \frac{1}{e}$ already hold even for identical valuations [17, 21].

1.1 Our results

We design a simple greedy algorithm, which we call *Truncated Max-Sum Greedy*, that achieves a 0.4-approximation for identical submodular valuations, improving the previously best-known factor of $\frac{10}{27} \approx 0.37$ by Uziahu and Feige [28]. Essentially, given a fixed threshold α, our algorithm greedily builds an allocation while ignoring players who already have items of value $\alpha \cdot \mathsf{OPT}$. Our main result is the following:

▶ **Theorem 1.1.** *There is a $\frac{2}{5}$-approximation for submodular max-min allocation under identical valuations.*

While showing a guarantee of $\frac{1}{3}$ is rather straightforward, the improvement to $\frac{2}{5}$ uses some graph-theoretic tools such as Hall's theorem and certain arguments about the digraph of reallocations needed to transform the greedy solution into an optimal solution. Our algorithm also provides guarantees in more general settings. As an example, we consider the problem of submodular max-min allocation under identical valuations with an additional *global cardinality constraint*: The objective remains the same, but we are allowed to use at most k items in total. We show the following:

▶ **Proposition 1.2.** *There is a $\frac{1}{3}$-approximation for submodular max-min allocation under identical valuations with a global cardinality constraint.*

Furthermore, we study the integrality gap of the *configuration LP* for max-min allocation, an important LP relaxation for max-min allocation.

We bound the integrality gap of the configuration LP by transforming certain primal integral solutions of the configuration LP to satisfying assignments of a dual formulation of this LP. Unlike to similar approaches when rounding configuration LPs [25], our analysis only uses the fact that the primal integral solution is *at most* OPT. As a result, our proof does not provide an algorithm (not even a non-trivial exponential one) for constructing an integral solution that demonstrates the integrality gap.

Using this technique, we give the first constant upper bound on the integrality gap of the configuration LP for identical submodular valuations.

▶ **Theorem 1.3.** *The integrality gap of the configuration LP with identical submodular valuations is at most* 3*.*

Additionally, we give a lower bound on this integrality gap.

▶ **Proposition 1.4.** *The integrality gap of the configuration LP with identical submodular valuations is at least* $4/3$*, even with* 6 *items and* 3 *players.*

We believe that the technique of Theorem 1.3 can be used to show constant integrality gaps for submodular max-min allocation in more general settings. As an example, we also consider the case where we are given k matroids on the items and the sets that players receive must all be independent in all matroids. For this case, we also give a constant upper bound on the integrality gap, independent of k.

▶ **Theorem 1.5.** *The integrality gap of the configuration LP with identical submodular valuations and any number of matroid constraints is at most* 5*.*

Accounting for the loss incurred when solving the separation oracle of the dual LP, this yields a $\mathcal{O}(k)$-estimation algorithm for max-min allocation subject to k matroid constraints.

1.2 Related work

Additive valuations. Additivity is one of the simplest assumptions on the valuation functions of the players. In the general version where players may have non-identical additive valuation functions, the state-of-the-art algorithm by Chakrabarty, Chuzhoy and Khanna [7] achieves an approximation factor of $\mathcal{O}(n^{\varepsilon})$. Interestingly, the best-known lower bound, shown by Bezáková and Dani [6], only rules out a factor better than 2.

A well-studied special case is the so-called *restricted assignment* case, where players have identical valuations, but a bipartite graph between items and players restricts the allowed allocations. To tackle this problem, Bansal and Sviridenko [4] introduced the configuration LP, and subsequent work [4, 2, 22, 1] led to the current best approximation factor of $\frac{1}{4} - \varepsilon$ for the problem [10]. Recently, the integrality gap of the configuration LP for restricted assignment was shown to be at most 3.53 [15]. For identical valuations, Woeginger designed a polynomial-time approximation scheme (PTAS) [30].

Submodular valuations. The first approximation algorithm for general submodular valuations was given in [14], and even though improvements have been made [13], the best-known approximation factor remains polynomial in the input. For a constant number of players, Chekuri, Vondrák and Zenklusen [8] gave a $1 - 1/e$-approximation.

For max-sum, Vondrak gave a $1 - 1/e$-approximation [29], which was shown to be optimal [17, 21]. In the restricted assignment case, Bamas, Morell and Rohwedder [3] generalized the techniques from Bansal and Sviridenko to obtain a $\mathcal{O}(\log \log n)$-approximation. Under identical valuations, Krause, Rajagopal, Gupta and Guestrin [18] gave the first constant-factor approximation algorithm, which uses an algorithm for max-sum as a subroutine. Plugging in Vondrak's optimal algorithm, their method achieves a $\frac{e-1}{3e} \approx 0.21$-approximation.

More recently, another line of research has studied the problem of finding an allocation which approximates the *maximin share* of all players, which is the largest value a player can ensure by proposing an allocation and receiving the bundle which is worst for them. Under identical valuations, this problem is equivalent to submodular max-min allocation. Barman and Krishnamurthy [5] gave a different algorithm with the same guarantee as Krause et al. [18]. Ghodsi, Hajiaghayi, Seddighin, Seddighin and Yami [12] improved this factor to $\frac{1}{3}$. Finally, Uziahu and Feige [28] gave a $\frac{10}{27}$-approximation by analyzing a bidding procedure which uses a truncated version of the valuation function.

Machine scheduling. If we reverse the roles of *max* and *min*, we obtain the classical *makespan minimization* setting, where we try to assign jobs to machines so as to minimize the latest completion time. As its max-min counterpart, it admits a PTAS for identical additive valuations [16]. The submodular variant of makespan minimization, sometimes called *submodular load balancing*, is not $o\left(\sqrt{\frac{n}{\log n}}\right)$-approximable [27]. This stands in stark contrast to its "dual" max-min variant.

2 Preliminaries

Consider a set J (with $n := |J|$) of items and a *valuation function* $f : 2^J \to \mathbb{R}$ that assigns a value to each subset of J. We simplify notation by writing $f(j)$ instead of $f(\{j\})$, for $j \in J$.

We focus on the case where f is *submodular*: A function $f : 2^J \to \mathbb{R}$ is called submodular, if for every two subsets S and T,

$$f(S \cup T) \leq f(S) + f(T) - f(S \cap T).$$

Closely related to submodular functions is the notion of *marginal contribution*. We define the marginal contribution on f of an item j with respect to a subset S as $\Delta_f(j \mid S) := f(S \cup \{j\}) - f(S)$. We extend this definition to sets (i.e. $\Delta_f(S \mid T) = f(S \cup T) - f(T)$). In fact, a useful characterization of submodularity is that for all subsets $S \subseteq T$ of J and $j \in J$,

$$\Delta_f(j \mid S) \geq \Delta_f(j \mid T).$$

A function is *monotone* if for all $S \subseteq T$, we have $f(S) \leq f(T)$. We assume $f(\emptyset) = 0$, and so a monotone function is non-negative. In the full version of the paper, we give some more basic facts about submodular functions that will be used implicitly throughout our proofs.

We define P (with $m := |P|$) to be the set of *players*. Since we are focusing on identical valuations, each player p has the same valuation function f describing their satisfaction with each set of items.

More formally, a *partial allocation* of J is a partition $A = (A_1, ..., A_m)$ of a subset of J. Slightly abusing notation, we write $j \in A$ for $j \in \bigcup_{p \in P} A_p$. If every item is allocated, we speak of an *allocation*. We refer to the sets of items in a (partial) allocation as *configurations*. We define $f_{\mathsf{sum}}(A) = \sum_{p \in P} f(A_p)$ and $f_{\mathsf{min}}(A) := \min_{p \in P} f(A_p)$. We will refer to players p with $f(A_p) = f_{\mathsf{min}}(A)$ as *min players*.

We briefly define matroids since we discuss them in Section 4.3. A *matroid* $\mathcal{M} = (J, \mathcal{I})$ is a non-empty subset-closed set system in which for any $J' \subseteq J$, all sets of $\mathcal{I}$ restricted to J' which are maximal have the same cardinality. The sets in $\mathcal{I}$ are called *independent*.

Submodular Max-Min Allocation

With these definitions, we are ready to define the main computational problem of interest.

Submodular Max-Min Allocation
Input: A set of items J, a set of players $P = \{1, ..., m\}$, and a monotone submodular function $f : 2^J \to \mathbb{R}$.
Output: An allocation A that maximizes $f_{\mathsf{min}}(A)$

Notice that each player has the same valuation function f. We assume that we are given access to a so-called *value oracle*, which, given any $S \subseteq J$, returns $f(S)$.

Approximation Hardness. Khot, Lipton, Markakis and Mehta [17] showed that when players have identical valuations, it is **NP**-hard to distinguish between the case where all players can achieve value 1 and the case where a player achieves at most value $1 - \frac{1}{e} + \varepsilon$ on average. Since the average value is always at least as big as the min value, their hardness result, while originally designed for max-sum, immediately translates to our setting:

▶ **Corollary 2.1** ([17]). *Unless* $\mathbf{P} = \mathbf{NP}$, Submodular Max-Min Allocation *cannot be approximated within a factor* $1 - \frac{1}{e} + \varepsilon$ *for any* $\varepsilon > 0$.

3 Truncated Max-Sum Greedy

In this section, we present our greedy algorithm for Submodular Max-Min Allocation.

Fix a threshold $\alpha \leq 1$. We assume that the algorithm knows the optimal value OPT. [1]

[1] We assume that the size of a rational representation of $f(S)$ is at most polynomial in $n + m$. We can therefore scale f such that all values are integral and perform a binary search with start interval $[0, \overline{f}(J)]$, where $\overline{f}$ is the scaled version of f. Under our assumption, the binary search procedure will only recurse a polynomial number of times.

First, the algorithm takes the m items j with largest $f(j)$-value and allocates one of them to each player. Then, it greedily chooses the pair of available item j and player p that maximizes the marginal contribution $\Delta_f(j \mid A_p)$, under the constraint that $f(A_p) < \alpha \cdot \mathsf{OPT}$, i.e. we only consider players that have not reached an α-fraction of their desired value.

The goal is to show that each player will eventually surpass this threshold, which proves an α-approximation. The pseudocode is shown in Algorithm 1.

Algorithm 1 The truncated max-sum greedy.

Data: items J, players P, submodular function $f : 2^J \to \mathbb{R}$
Result: an allocation A
Remove the m items j with largest $f(j)$ from J, and give one of them to each player;
while $J \neq \emptyset$ *and there exists* $p \in P$ *with* $f(A_p) < \alpha \cdot \mathsf{OPT}$ **do**
  $(j^*, p^*) \leftarrow \arg\max_{j \in J, p \in P : f(A_p) < \alpha \cdot \mathsf{OPT}} \Delta_f(j \mid A_p)$;
  $J \leftarrow J - \{j^*\}$;
  $A_{p^*} \leftarrow A_{p^*} \cup \{j^*\}$;
end
return $(A_1, ..., A_m)$

▶ **Observation 3.1.** *We can assume* $f(j) < \alpha \cdot \mathsf{OPT}$ *for all* $j \in J$.

Proof. We show that if the truncated max-sum greedy gives an α-approximation on instances where for all items $j \in J$, $f(j) < \alpha \cdot \mathsf{OPT}$, then it also gives an α-approximation for the general case.

Let OPT denote the optimal value in the general instance. Assume that there are k *big* items j with $f(j) \geq \alpha \cdot \mathsf{OPT}$. In an optimal allocation, these items can be given to at most k players. Thus, there exists an instance with $\max(0, m-k)$ players and a subset $J' \subseteq \{j \in J \mid f(j) < \alpha \cdot \mathsf{OPT}\}$ of items that achieve at least OPT.

If greedy is performed on the instance with $\max(0, m-k)$ players and without the big items, then it will obtain a solution of value at least $\alpha \cdot \mathsf{OPT}$ (since the optimal solution on this reduced instance has value OPT, as argued above). Adding k players, each containing one big item, gives a solution of value at least $\alpha \cdot \mathsf{OPT}$ to the general instance. ◀

As a warm-up, we first show that we obtain a $\frac{1}{3}$-approximation when setting $\alpha = \frac{1}{3}$. Afterwards, we will show how to improve the factor to $\frac{2}{5}$ using a more refined analysis.

Consider a greedy allocation A and an optimal allocation A^*. Let q be a min player in A, i.e. $f(A_q) = f_{\mathsf{min}}(A)$. We may assume $n \geq m$, since otherwise the optimal value is 0. We first show the following simple but crucial lemma.

▶ **Lemma 3.2.** *Assume* $f(A_q) < \alpha \cdot \mathsf{OPT}$. *Then,*

$$\sum_{j \in A^*} \Delta_f(j \mid A_q) \leq \sum_{p \in P - \{q\}} f(A_p)$$

Proof. For any item j, let $A^{(j)}$ denote the partial allocation of the algorithm immediately before j is allocated. We have $\sum_{j \in A_q} \Delta_f(j \mid A_q) = 0$. For all other $p \in P - \{q\}$, we have

$$\sum_{j \in A_p} \Delta_f(j \mid A_q) \leq \sum_{j \in A_p} \Delta_f(j \mid A_q^{(j)}) \leq \sum_{j \in A_p} \Delta_f(j \mid A_p^{(j)}) = f(A_p) \tag{1}$$

The first inequality follows from the submodularity of f, since $A_q^{(j)} \subseteq A_q$. For the second inequality, notice that the algorithm always chooses to allocate an item to the player p with maximal marginal contribution, as long as $f(A_p) < \alpha \cdot \mathsf{OPT}$. Therefore,

$$\sum_{j \in A^*} \Delta_f(j \mid A_q) = \sum_{p \in P-\{q\}} \sum_{j \in A_p} \Delta_f(j \mid A_q) + \sum_{j \in A_q} \Delta_f(j \mid A_q) \leq \sum_{p \in P-\{q\}} f(A_p). \tag{2}$$

◀

We also need an upper bound on $f(A_p)$. The following lemma is sufficient for the $\frac{1}{3}$-approximation, but later we will need a stronger upper bound for the $\frac{2}{5}$-approximation.

▶ **Lemma 3.3.** *For all players $p \in P$, $f(A_p) \leq 2 \cdot \alpha \cdot \mathsf{OPT}$.*

Proof. Player p can only be allocated one item j that surpasses $\alpha \cdot \mathsf{OPT}$. This item has size $\leq \alpha \cdot \mathsf{OPT}$, by Observation 3.1. Let A' be the partial allocation before j is allocated to p. Then,

$$f(A_p) = f(A'_p) + \Delta_f(j \mid A'_p) \leq \alpha \cdot \mathsf{OPT} + f(j) \leq 2 \cdot \alpha \cdot \mathsf{OPT}$$ ◀

We can now prove the approximation ratio of $\frac{1}{3}$ by the following lemma.

▶ **Lemma 3.4.** $f_{\mathsf{min}}(A^*) < f(A_q) + \frac{1}{m} \sum_{p \in P} f(A_p)$.

Proof.

$$\begin{aligned}
f_{\mathsf{min}}(A^*) &\leq \min_{p \in P} f(A_p^* \cup A_q) && (3)\\
&\leq \min_{p \in P} f(A_q) + \sum_{j \in A_p^*} \Delta_f(j \mid A_q)\\
&\leq f(A_q) + \frac{1}{m} \sum_{p \in P} \sum_{j \in A_p^*} \Delta_f(j \mid A_q)\\
&= f(A_q) + \frac{1}{m} \sum_{j \in A^*} \Delta_f(j \mid A_q)\\
&\leq f(A_q) + \frac{1}{m} \sum_{p \in P-\{q\}} f(A_p) && (4)\\
&< f(A_q) + \frac{1}{m} \sum_{p \in P} f(A_p) && (5)
\end{aligned}$$

(3) uses monotonicity of f, (4) uses Lemma 3.2 and (5) holds since $f(A_q) > 0$ by our assumption $n \geq m$ and the fact that greedy starts by allocating one item to each player. ◀

To see that when setting $\alpha = \frac{1}{3}$, this indeed implies the approximation ratio of $\frac{1}{3}$, substitute $\frac{1}{m} \sum_{p \in P} f(A_p) \leq 2 \cdot \alpha \cdot \mathsf{OPT}$ by Lemma 3.3. Taking $2 \cdot \alpha \cdot \mathsf{OPT} = \frac{2}{3} \cdot \mathsf{OPT}$ on the other side, we obtain $\frac{1}{3}\mathsf{OPT} < f(A_q)$.

▶ Remark 3.5. This algorithm extends to the case where we are additionally given a *global cardinality constraint*. That is, we must choose a subset of at most k items to allocate.

To see this, consider again an optimal allocation A^* and the greedy allocation A. Notice that in our new setting, we might have $\bigcup_{p \in P} A_p \neq \bigcup_{p \in P} A_p^*$ and therefore the first equality in (2) no longer holds. However, we can circumvent this issue as follows. We can assume

that both A and A^* contain exactly k items by monotonicity. Let $b : \bigcup_{p\in P} A^*_p \to \bigcup_{p\in P} A_p$ be any bijection such that $b(j) = j$ for all $j \in (\bigcup_{p\in P} A^*_p) \cap (\bigcup_{p\in P} A_p)$. For all $j \in A$, let $A^{(j)}$ be the partial greedy allocation exactly before allocating j. Consider any $j \in A^*$ and let $b(j) \in A_p$. We have

$$\Delta_f(j \mid A_q) \leq \Delta_f(j \mid A_q^{(b(j))}) \leq \Delta_f(b(j) \mid A_p^{(b(j))}) \tag{6}$$

using that when $b(j)$ is allocated, item j is still available and thus greedy must prefer allocating $b(j)$ to p than allocating j to q. By (6) and the fact that b is a bijection,

$$\begin{aligned}\sum_{j\in A^*} \Delta_f(j \mid A_q) &\leq \sum_{p\in P-\{q\}} \sum_{j\in A^*:b(j)\in A_p} \Delta_f(b(j) \mid A_p^{(b(j))}) \\ &= \sum_{p\in P-\{q\}} \sum_{j\in A_p} \Delta_f(j \mid A_p^{(j)}) \\ &= \sum_{p\in P-\{q\}} f(A_p)\end{aligned}$$

reproving Lemma 3.2 in this more general setting. The rest of the analysis can be done as for the unconstrained case. Therefore, Proposition 1.2 holds.

Ratio of 2/5

We now prove that if $\alpha = \frac{2}{5}$, then $f_{\mathsf{min}}(A) \geq \frac{2}{5}\mathsf{OPT}$. To this end, suppose for contradiction that there exists a counterexample to this claim. We assume that this instance (P, J, f) is minimal with respect to $|P|$ and that f is scaled in such a way that $f_{\mathsf{min}}(A^*) = \mathsf{OPT} = 5$. We also fix a min player q with $f(A_q) \leq 2$. Note that we could have assumed $f(A_q) < 2$ but this is not necessary for the proof.

Let us define $\mathsf{avg} := \frac{1}{m}\sum_{p\in P} f(A_p)$. From Lemma 3.4 we obtain $\mathsf{avg} > 3$. Our goal is to show that $\mathsf{avg} \leq 3$, thereby arriving at a contradiction.

Letting a_p^i denote the i-th item allocated to p, define $\beta := \max_{p\in P:|A_p|\geq 2} \Delta_f(a_p^2 \mid \{a_p^1\})$ as the largest marginal contribution of an item during the greedy algorithm which is not the first item of a player. Notice that it suffices to consider a_p^2, since the marginal contributions of later items of a player can only be smaller. If $\beta \leq 1$, then no player can obtain more than value 3, so we can assume $\beta > 1$. Also, $\beta \leq 2$ holds by Observation 3.1. Notice finally that for $p \in P$, we have $f(a_p^1) \geq \beta$, as otherwise the maximizer a_p^2 in the definition of β would have been among the largest m items, and could not be the second item of a player.

▶ **Lemma 3.6.** *For each set of $k \geq 1$ players $p_1, ..., p_k \in P - \{q\}$, there is no set of k players $r_1, ..., r_k \in P$ such that $\bigcup_{i=1}^k A_{p_i} \subseteq \bigcup_{i=1}^k A^*_{r_i}$.*

Proof. Removing k players and all items in $\bigcup_{i=1}^k A_{p_i}$ from the instance leads to a smaller counter-example: The greedy algorithm still behaves identically on all other players, including q, so there is a valid greedy allocation A' on the instance with $f_{\mathsf{min}}(A') \leq 2$. On the other hand, since all removed items only belonged to k players in the optimal solution, removing these k players gives a solution A^{**} with $f(A^{**}_p) \geq 5$ for all remaining players p. ◀

The above lemma allows us to prove the existence of a "fixed" item for each player:

▶ **Lemma 3.7.** *There is a permutation $\phi : P \to P$ such that $A_p \cap A^*_{\phi(p)} \neq \emptyset$ for all $p \in P$.*

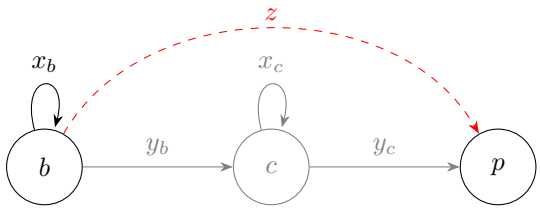


Figure 1 Visualization of Lemma 3.9.

Proof. Consider the bipartite graph $G = (P \cup P', E)$ with the set of players P on the left side and a copy P' of the set of players on the right side, and edges (p, p') if $A_p \cap A^*_{p'} \neq \emptyset$. Let $N(Q) = \bigcup_{q \in Q}\{v \in V \mid (q, v) \in E\}$ be the neighbourhood of Q. Consider an arbitrary non-empty subset $Q \subseteq P - \{q\}$. By Lemma 3.6, $|N(Q)| \geq |Q| + 1$. For the set $\{q\}$, we have $|N(\{q\})| \geq 1$ since $A_q \neq \emptyset$ and each item in q must be allocated somewhere in the optimal solution. For the set $\emptyset$, we have $|N(\emptyset)| \geq 0 = |\emptyset|$. For any other set $\{q\} \subset Q \subseteq P$, we know that $|N(Q)| \geq |N(Q - \{q\})| \geq |Q - \{q\}| + 1 = |Q|$. Altogether, we obtain that for any subset $Q \subseteq P$, $|N(Q)| \geq |Q|$. We can hence apply Hall's Marriage Theorem to conclude that G contains a perfect matching. This matching defines exactly the desired permutation ϕ. ◀

To simplify notation, we consider an optimal solution such that ϕ is the identity function, by just permuting the allocations of the players in the optimal solution. We now define the *reallocation digraph* $G = (P, E)$: Each pair of players $(p, r) \in P^2$ is associated to a set $A_{p\to r} = A_p \cap A^*_r$. An arc (p, r) exists if $A_{p\to r} \neq \emptyset$. By Lemma 3.7, each vertex has a self-loop.

We subdivide the players of our instance into two categories $P = B \cup S$. The set B of *big* players consists of all players b such that $f(A_b) > 4 - \beta$. The set S of *small* players consists of all players p with $f(A_p) \leq 4 - \beta$.

Notice that $q \in S$. This partition is mainly motivated by the following observation:

▶ **Observation 3.8.** *For any $p \in B$, it holds that $|A_p| = 2$.*

Proof. Any player $p \in B$ cannot be allocated just one item j since $f(j) < 2 \leq 4 - \beta \leq f(A_p)$. If p was allocated k items for $k > 2$, then $f(\{a^1_p, \ldots, a^{k-1}_p\}) < 2$ must hold in order for the greedy algorithm to allocate a^k_p to p. But since $f(a^1_p) \geq \beta$, we must have $\Delta_f(a^k_p \mid \{a^1_p, \ldots, a^{k-1}_p\}) \leq \Delta_f(a^2_p \mid \{a^1_p\}) \leq 2 - \beta$. So we have

$$f(A_p) = f(A_p - a^k_p) + \Delta_f(a^k_p \mid \{a^1_p, \ldots, a^{k-1}_p\}) < 2 + 2 - \beta \leq f(A_p),$$

contradicting our assumption $k > 2$. ◀

For each $p \in B$, denote by x_p the item in $A_{p\to p}$ and denote by y_p the other item of A_p. Lemma 3.7 guarantees the existence of the first item and Observation 3.8 implies that there is exactly one other item. Lemma 3.6 guarantees that the other item is not in $A_{p\to p}$.

▶ **Lemma 3.9.** *For any $b, c \in B$ with $b \neq c$, $(b, c) \notin E$.*

Proof. Suppose for contradiction that $A_{b\to c} \neq \emptyset$, where $y_b \in A_{b\to c}$. Let $p \in P$ be the player such that $y_c \in A^*_p$. Consider a modified instance where we remove player c as well as items $\{y_b, x_c, y_c\}$ and add an item z which contributes an additive factor of $t := \max(f(y_b), \Delta_f(y_c \mid A^*_p - \{y_c\}))$ to every set. This function is submodular and monotone. Notice that this addition of z does not change any marginal contribution, i.e. for $j \neq z$ we have $\Delta_f(j \mid S) = \Delta_f(j \mid S \cup \{z\})$.

The greedy algorithm on the new instance may allocate all items that give marginal contribution larger than t in the original run as before. As soon as all remaining items give marginal contribution $\leq t$, it may choose to allocate item z to player b (since z contributes the same to each player, and greedy considers b as $f(A_b) = f(\emptyset) < 2$ or $f(A_b) = f(x_b) < 2$).

Since $b \in B$, we have $f(A_b) > 4 - \beta \geq 2$. Furthermore, $f(\{x_b, z\}) \geq f(x_b) + f(y_b) \geq f(A_b) \geq 2$. Consider the moment in the greedy algorithm after z was allocated to b. If x_b had already been allocated, then b has reached the threshold of 2 and will be thus ignored by the greedy algorithm. Therefore, all following items can be allocated to the same players as in the original run of the greedy algorithm. Now suppose x_b had not yet been allocated. Since the addition of z to player b does not influence the marginal contributions and $f(z) < 2$, the greedy algorithm can allocate all items up to x_b in the same way (including x_b). After allocating x_b, player b has reached the threshold of 2 and thus again the greedy algorithm can allocate in the same way as in the original run. Thus, we again obtain $f_{\mathsf{min}}(A'_q) \leq 2$, for the new allocation A'.

On the other hand, we can modify our old optimal allocation, by just replacing item y_c by item z. Call the new optimal allocation A^{**}. We have

$$\begin{aligned} f(A_p^{**}) &= f(A_p^{**} - \{z\}) + t \\ &= f(A_p^{*} - \{y_c\}) + t \\ &\geq f(A_p^{*} - \{y_c\}) + \Delta_f(y_c \mid A_p^{*} - \{y_c\}) \\ &= f(A_p^{*}) \geq 5 \end{aligned}$$

We obtained a counterexample with less players, a contradiction. ◀

Thus, in fact, $G[B]$ is an independent set. We now argue that B cannot be too large. To this end, we first prove the following two helpful lemmata. For a player p, let us define $V_p := \bigcup_{(r,p)\in E} A_{r\to p}$ the set of items reallocated *to* p and $W_p := \bigcup_{(p,r)\in E} A_{p\to r}$ the set of items reallocated *from* p.

▶ **Lemma 3.10.** *For any player p, $\sum_{j\in W_p} \Delta_f(j \mid A_q) < 2$.*

Proof. Recall that $A^{(j)}$ denotes the partial greedy allocation exactly before item j is allocated. Let ℓ be the last item allocated to A_p. By Lemma 3.7, there exists an item $k \in A_{p\to p}$. Therefore, $W_p \subseteq A_p - \{k\}$ and we can write

$$\begin{aligned} \sum_{j\in W_p} \Delta_f(j \mid A_q) &\leq \sum_{j\in A_p - \{k\}} \Delta_f(j \mid A_q) \\ &\leq \sum_{j\in A_p - \{k\}} \Delta_f(j \mid A_q^{(j)}) \\ &\leq \Delta_f(\ell \mid A_p^{(\ell)}) + \sum_{j\in A_p - \{k,\ell\}} \Delta_f(j \mid A_p^{(j)}) \\ &\leq \Delta_f(\ell \mid A_p^{(k)}) + \sum_{j\in A_p - \{k,\ell\}} \Delta_f(j \mid A_p^{(j)}) \\ &\leq \Delta_f(k \mid A_p^{(k)}) + \sum_{j\in A_p - \{k,\ell\}} \Delta_f(j \mid A_p^{(j)}) \\ &= f(A_p - \{\ell\}) < 2 \end{aligned}$$

The third inequality is because the items were allocated to p and not q. The fourth inequality comes from $A_p^{(\ell)} \supseteq A_p^{(k)}$, then we use that at the partial allocation $A^{(k)}$, greedy prefered k over ℓ. Finally, the strict inequality holds because otherwise p would have achieved the $\alpha \cdot \mathsf{OPT}$ threshold and not received any more items. ◀

▶ **Lemma 3.11.** *For any $p \in P$, $\sum_{j \in V_p} \Delta_f(j \mid A_q) \geq 3 + \sum_{j \in W_p} \Delta_f(j \mid A_q) - f(A_p)$.*

Proof. First, notice that

$$\sum_{j \in A_p^*} \Delta_f(j \mid A_q) \geq \Delta_f(A_p^* \mid A_q) \geq f(A_p^*) - f(A_q) \geq 5 - 2 = 3. \tag{7}$$

Since $V_p = A_p^* - (A_p - W_p)$, $A_p - W_p \subseteq A_p^*$, and $W_p \subseteq A_p$, we can write

$$\sum_{j \in V_p} \Delta_f(j \mid A_q) = \sum_{j \in A_p^*} \Delta_f(j \mid A_q) + \sum_{j \in W_p} \Delta_f(j \mid A_q) - \sum_{j \in A_p} \Delta_f(j \mid A_q)$$

Now, the result follows by using (7) and (1). ◀

▶ **Lemma 3.12.** $|B| \leq |S|$.

Proof. Let S_0 be the set of small players that do not receive any item from big players, let S_1 be the set of small players that receive one item from one big player and let $S_{\geq 2}$ be the set of small players that receive items from multiple big players. If we can show $S_{\geq 2} = \emptyset$, this means that no two big players have an edge towards the same small player. But since every big player has an edge towards a small player by Lemma 3.9, by pigeonhole principle this would imply the desired $|B| \leq |S|$.

Associate with each player p the quantity

$$g(p) := \sum_{j \in W_p} \Delta_f(j \mid A_q) - \sum_{j \in V_p} \Delta_f(j \mid A_q)$$

Note that $\sum_{p \in P} g(p) = 0$, as $\bigcup_{p \in P} W_p = \bigcup_{p \in P} V_p$. We can decompose the set of players as

$$P = S_0 \cup \bigcup_{p \in S_1} \{p, b_1(p)\} \cup \bigcup_{p \in S_{\geq 2}} \{p, b_1(p), b_2(p), ..., b_k(p)\}$$

where $b_i(p)$ denotes the i-th big player that gives an item to p, in an arbitrary order. Therefore,

$$\sum_{p \in P} g(p) = \sum_{p \in S_0} g(p) + \sum_{s \in S_1} g(s) + g(b_1(s)) + \sum_{s \in S_{\geq 2}} g(s) + g(b_1(s)) + ... + g(b_k(s)) \tag{8}$$

Since $\sum_{p \in P} g(p) = 0$, to prove $S_{\geq 2} = \emptyset$, it is enough to show that all terms of (8) are ≤ 0, and that terms of the last sum are < 0. This is the purpose of the rest of the proof.

1. $g(p) \leq 0$ for $p \in S_0$. This follows directly from Lemma 3.11 and $f(A_p) \leq 4 - \beta \leq 3$.
2. $g(s) + g(b) \leq 0$ for $s \in S_1$ and $b = b_1(s)$. Since s is a small player, we have $f(A_s) \leq 4 - \beta$, and by Lemma 3.11, we get $g(s) \leq 1 - \beta$. If we can show $g(b) \leq \beta - 1$, then this implies $g(s) + g(b) \leq 0$. Recall that $W_b = \{y_b\}$.
   As the second item allocated to b contributes at most β, we have $\Delta_f(x_b \mid A_q) + \Delta_f(y_b \mid A_q) \leq 2 + \beta$. By the facts $f(A_b^*) - f(A_q) \geq 3$ and $A_b^* = V_b \cup (A_b - W_b)$, we get

$$\begin{aligned}
\sum_{j \in V_b} \Delta_f(j \mid A_q) &\geq \Delta_f(V_b \cup (A_b - W_b) \mid (A_b - W_b) \cup A_q) \\
&= \Delta_f(A_b^* \mid (A_b - W_b) \cup A_q) \\
&\geq f(A_b^*) - \Delta_f(A_b - W_b \mid A_q) - f(A_q) \\
&\geq 3 - \Delta_f(x_b \mid A_q)
\end{aligned}$$

   In total, $g(b) \leq \Delta_f(y_b \mid A_q) - \sum_{j \in V_b} \Delta_f(j \mid A_q) \leq \Delta_f(y_b \mid A_q) - 3 + \Delta_f(x_b \mid A_q) \leq \beta - 1$.

3. $g(s)+g(b)+g(c)<0$ for $s\in S_{\geq 2}$, $b=b_1(s)$ and $c=b_2(s)$. We have $f((A_b-W_b)\cup A_q)\leq f(A_b-W_b)+f(A_q)\leq 4$, but since $f(A_b^*)\geq 5$, we must have

$$\sum_{j\in V_b}\Delta_f(j\mid A_q)\geq \Delta_f(V_b\cup(A_b-W_b)\mid (A_b-W_b)\cup A_q)\geq 1.$$

The same holds for player c. By Lemma 3.10, we also know $g(s)<2-\Delta_f(y_b\mid A_q)-\Delta_f(y_c\mid A_q)$. Therefore

$$g(b)+g(c)+g(s)\leq \Delta_f(y_b\mid A_q)-1+\Delta_f(y_c\mid A_q)-1+g(s)<0. \qquad \blacktriangleleft$$

We can conclude as follows:

$$\begin{aligned}\mathsf{avg}&=\frac{1}{m}\sum_{p\in P}f(A_p)\\&\leq\frac{1}{m}\Big[|B|\cdot(2+\beta)+|S|\cdot(4-\beta)\Big]\\&\leq\frac{1}{m}\Big[\frac{|B|+|S|}{2}\cdot(2+\beta)+\frac{|B|+|S|}{2}\cdot(4-\beta)\Big]\\&=\frac{1}{m}\Big[\frac{6\cdot(|B|+|S|)}{2}\Big]\\&=3\end{aligned}$$

where the last inequality is because $2+\beta\geq 4-\beta$, and $|B|\leq|S|$ by Lemma 3.12. This contradicts $\mathsf{avg}>3$, as stated in the beginning. This finishes the proof of Theorem 1.1.

We discuss the (close-to-) tightness of this analysis in the full paper version. In particular, we show that no value of α can lead to an approximation factor better than $\frac{2+S}{4+3S}\approx 0.4068$, where $S=\sum_{i=1}^{\infty}\frac{1}{s_i-1}$ with $s_k=1+\prod_{i=1}^{n}s_i$ is a constant related to the *Sylvester sequence* or *Salzer sequence* and was already used in [20, 24] for other packing problems.

4 Integrality gaps for the configuration LP

Assignment LP (A)

$$\sum_{i=1}^{m}x_{ij}=1 \qquad \forall j=1,\dots,n$$

$$\sum_{j=1}^{n}f_i'(j)x_{ij}\geq T \qquad \forall i=1,\dots,m$$

$$x_{ij}\geq 0 \qquad \forall i,j$$

Configuration LP (P)

$$\sum_{C\in\mathcal{C}(T,p)}x_{p,C}\geq 1 \qquad \forall p\in P$$

$$\sum_{C\ni j}\sum_{p\in P}x_{p,C}\leq 1 \qquad \forall j\in J$$

$$x_{p,C}\geq 0 \quad \forall p\in P, C\in\mathcal{C}(T,p)$$

Figure 2 The assignment LP and the configuration LP.

In this section, we study the integrality gap of the *configuration LP* in the case of identical valuation functions. Before defining this LP, let us consider the simpler *assignment LP*: It searches for the largest T such that the program depicted in figure 2 on the left, where $f_i'(j)=\min(f_i(j),T)$, has a feasible solution. Let us call this LP (A). It is a folklore result that (A) has an integrality gap of 2 when all players have identical additive valuations.

Dual LP (D0)

$$\max \sum_{p\in P} z_p - \sum_{j\in J} y_j \quad \text{s.t.}$$
$$\sum_{j\in C} y_j \geq z_p \qquad \forall p, C$$
$$y_j, z_p \geq 0 \qquad \forall j, p$$

Dual LP (D1)

$$\sum_{p\in P} z_p > \sum_{j\in J} y_j$$
$$\sum_{j\in C} y_j \geq z_p \qquad \forall p, C$$
$$y_j, z_p \geq 0 \qquad \forall j, p$$

Dual LP (D2)

$$\sum_{j\in J} y_j < m$$
$$\sum_{j\in C} y_j \geq 1 \qquad \forall C$$
$$y_j \geq 0 \qquad \forall j$$

Figure 3 The duals to the configuration LP.

In the restricted assignment case, (A) has a bad integrality gap [4]. This was the initial motivation to study a stronger LP, namely the aforementioned configuration LP. Let $\mathcal{C}(T, p)$ contain the configurations C such that $f_p(C) \geq T$. We have a variable for each pair of player and configuration. The LP is shown in figure 2 on the right. The largest T such that this LP has a feasible solution is an upper bound on the optimal solution. We call this system (P).

It is well-known that the integrality gap of (P) in the general case is $\Omega\left(\sqrt{m}\right)$ [4], while in the restricted assignment case it was shown to be smaller than 3.534 [15]. When valuation functions are submodular, the gap in the restricted assignment case is $\mathcal{O}(\log \log n)$ [3].

(P) has exponentially many variables in general. Assuming that the primal minimizes the 0-vector, we can formulate its dual (D0), depicted in figure 3 on the left. The separation problem for this LP is the classical Knapsack problem. Since Knapsack admits a (F)PTAS [23], we can solve the LP up to any desired accuracy using the Ellipsoid method.

4.1 Constructing Dual Solutions

Notice that in (D0), a trivial solution always exists by setting all variables to 0. But if we had a solution with objective value larger than 0, then we could obtain an arbitrarily high objective value by scaling all variables. By LP duality, we get that if (D0) is unbounded, then (P) with the same threshold T is infeasible. Additionally, we can directly express the requirement that the objective function must be greater than 0 as a constraint. We call the resulting LP (D1), and it is shown in the middle of figure 3.

In the case of identical valuations, we can, without loss of generality, set all player variables to 1, and obtain (D2), as seen on the right of figure 3. This LP searches for a fractional hitting set of size smaller than m that hits all configurations larger than our threshold value. In (D2), we call the first constraint *value constraint* and the second set of constraints *configuration constraints*. If (D2) has a feasible solution, then (D0) is unbounded, and by weak duality, (P) is infeasible. We obtain the following:

▶ **Proposition 4.1.** *Consider a class Γ of max-min allocation instances, and let $\mathsf{OPT}(\mathcal{I})$ be the optimal integral value of instance $\mathcal{I}$. If for all $\mathcal{I} \in \Gamma$, (D2) is feasible for $\mathcal{I}$ and threshold value $\alpha \cdot \mathsf{OPT}(\mathcal{I})$, then the integrality gap of the configuration LP on Γ is at most α.*

We give one more useful observation:

▶ **Observation 4.2.** *If there are variables y_j such that the configuration constraints of (D2) are satisfied, $\sum_{j\in J} y_j \leq m$ and there exists $j^* \in J$ such that for all $C \ni j^*$, we have $\sum_{j\in C} y_j > 1$, then (D2) is feasible.*

Proof. Since y_{j^*} is not involved in any tight constraint, we can lower its value by a sufficiently small amount, satisfying the value constraint while keeping all configuration constraints satisfied. ◀

4.2 Submodular valuations

The system (P) can also be formulated for submodular valuations. While in the additive case, solving the separation problem only incurred an ε error, this is different in the submodular case. This is because we need to solve a submodular version of knapsack for the separation oracle. Due to the hardness of maximum coverage [11], which includes submodular knapsack, we lose a factor of $1 - \frac{1}{e}$ while solving the LP. On the algorithmic side, this hardness is matched by a $1 - \frac{1}{e}$-approximation by Sviridenko [26]. Therefore, any rounding algorithm that loses a factor of α translates into an approximation algorithm with factor $(1 - \frac{1}{e})\alpha$.

In the following, we show that apart from our combinatorial approach presented in the last section, an LP rounding approach could also lead to a constant-factor approximation..

▶ **Theorem 1.3.** *The integrality gap of the configuration LP with identical submodular valuations is at most* 3*.*

We first show that we can restrict our attention to items j with $f(j) \leq \mathsf{OPT}$. Let $\mathsf{OPT}(\mathcal{I})$ denote the optimal integral value for an instance $\mathcal{I}$, and let $\mathsf{OPT}_{\mathsf{LP}}(\mathcal{I})$ denote the optimal value of (P) on $\mathcal{I}$.

▶ **Lemma 4.3.** *Let $\mathcal{I}$ be an instance with an item j such that $f(j) > \mathsf{OPT}(\mathcal{I})$. Let $\mathcal{I}'$ be the instance obtained from $\mathcal{I}$ by removing j and one player. Then, $\mathsf{OPT}(\mathcal{I}) \geq \mathsf{OPT}(\mathcal{I}')$ and $\mathsf{OPT}_{\mathsf{LP}}(\mathcal{I}) \leq \mathsf{OPT}_{\mathsf{LP}}(\mathcal{I}')$.*

Proof. Fix an arbitrary player $p \in P$. For an optimal solution x to (P), we have

$$\sum_{C \ni j} \sum_{p \in P} x_{p,C} \leq 1 \leq \sum_{C \in \mathcal{C}(\mathsf{OPT}_{\mathsf{LP}}(\mathcal{I}),p)} x_{p,C}$$

Therefore, we can replace all configurations in which j appears by the conifigurations of player p, without violating any constraint except the constraint of player p. Since item j is not used anymore, this is a valid solution to (P) on instance $\mathcal{I}'$ of value $\mathsf{OPT}_{\mathsf{LP}}(\mathcal{I})$.

Now let A be an optimal integral allocation of $\mathcal{I}'$ and assume for contradiction $f_{\mathsf{min}}(A) > \mathsf{OPT}(\mathcal{I})$. Adding a new player r with $A_r = \{j\}$ gives an integral allocation for $\mathcal{I}$ of value $\min(f_{\min}(A), f(j)) > \mathsf{OPT}(\mathcal{I})$, a contradiction. ◀

Therefore, if our instance contains items that contribute more than OPT, then we can instead consider a smaller instance with larger integrality gap. We now prove Theorem 1.3.

Proof. Consider an arbitrary instance $\mathcal{I}$ of our problem and scale the valuation function so that $\mathsf{OPT}(\mathcal{I}) = 1$. We can assume that for all $j \in J$, $f(j) \leq 1$ by Lemma 4.3. Let $\overline{f}(S) := \min(2, f(S))$, and consider an allocation A that maximizes $\sum_{p \in P} \overline{f}(A_p)$.

For $j \in A_p$ and f, define $\Delta_f(-j \mid A_p) := \Delta_f(A_p \mid A_p - \{j\})$. For any $p \in P$, let $T_f(p) := \sum_{j \in A_p} \Delta_f(-j \mid A_p)$. For each $p \in P$ and $j \in A_p$, we set the variables in (D2) to

$$y_j = \frac{\Delta_{\overline{f}}(-j \mid A_p)}{T_{\overline{f}}(p)}$$

if $T_{\overline{f}}(p) \neq 0$, and $y_j = 0$ otherwise. For notational convenience, define $y(S) = \sum_{j \in S} y_j$. We prove that y is a (strictly) feasible solution to (D2) when $T = 3$. Suppose for contradiction that there is a set C of items with $f(C) > 3$ but $y(C) \leq 1$.

Let q be the min player in A. Clearly, $f(A_q) \leq 1$, and thus $\Delta_f(C \mid A_q) > 2$. Let $j^* = \arg\max_{j \in C} \frac{\Delta_f(j \mid A_q)}{y_j}$. If $\frac{\Delta_f(j^* \mid A_q)}{y_{j^*}} \leq 2$, then we would have

$$\frac{\Delta_f(C \mid A_q)}{y(C)} \leq \frac{\sum_{j \in C} \Delta_f(j \mid A_q)}{y(C)} \leq \frac{\sum_{j \in C} 2 \cdot y_j}{y(C)} = 2$$

which contradicts our assumption $y(C) \leq 1$. Therefore, we can assume $\frac{\Delta_f(j^* \mid A_q)}{y_{j^*}} > 2$.

We want to argue that reallocating item j^* to player q increases our potential according to $\overline{f}$. Notice that the gain of adding j^* to player q is $\Delta_{\overline{f}}(j^* \mid A_q)$, while the loss of removing j^* from player p is $\Delta_{\overline{f}}(-j^* \mid A_p) = y_{j^*} \cdot T_{\overline{f}}(p)$. We now bound $T_{\overline{f}}(p)$. First, we have

$$\begin{aligned} T_{\overline{f}}(p) &= \sum_{j \in A_p} \Delta_{\overline{f}}(-j \mid A_p) \\ &= \sum_{j \in A_p} \min\Big(0, \Delta_f(-j \mid A_p) - \max\Big(0, f(A_p) - 2\Big)\Big) \\ &= \sum_{j \in A_p : \Delta_f(-j|A_p) \geq f(A_p)-2} \Delta_f(-j \mid A_p) - \max\Big(0, f(A_p) - 2\Big) \end{aligned}$$

If this sum is empty, then $T_{\overline{f}}(p) = 0$. Otherwise, we can assume that $\max\big(0, f(A_p) - 2\big)$ is subtracted at least once. Thus,

$$\begin{aligned} T_{\overline{f}}(p) &\leq \Big(\sum_{j \in A_p} \Delta_f(-j \mid A_p)\Big) - \max\Big(0, f(A_p) - 2\Big) \\ &\leq f(A_p) - \max\Big(0, f(A_p) - 2\Big) \\ &\leq 2 \end{aligned}$$

where the second inequality is by submodularity of f. Therefore,

$$\Delta_{\overline{f}}(j^* \mid A_q) - \Delta_{\overline{f}}(-j^* \mid A_p) > \Delta_{\overline{f}}(j^* \mid A_q) - 2 \cdot y_{j^*} = \Delta_f(j^* \mid A_q) - 2 \cdot y_{j^*} > 0,$$

where the equality is because of $f(A_q \cup \{j^*\}) = \overline{f}(A_q \cup \{j\})$, which follows from our assumption $f(j^*) \leq 1$ and $f(A_q) \leq 1$. So, reallocating j^* from p to q increases the potential $\overline{f}$, contradicting the optimality of A.

We conclude that the configuration constraints in (D2) are satisfied and not tight for $T > 3$. We can finish the proof with Observation 4.2, since

$$\sum_{j \in J} y_j = \sum_{p \in P} \sum_{j \in A_p} y_j \leq m. \qquad \blacktriangleleft$$

A simple lower bound on the integrality gap is proven in the full version of the paper.

▶ **Proposition 1.4.** *The integrality gap of the configuration LP with identical submodular valuations is at least* $4/3$*, even with* 6 *items and* 3 *players.*

4.3 Submodular valuations with matroid constraints

Now consider the following problem: Apart from items J, players P and a submodular monotone valuation function f, we are given k matroids $\mathcal{M}_1 = (J, \mathcal{I}_1), \ldots, \mathcal{M}_k = (J, \mathcal{I}_k)$ and our task is to find a (partial) allocation A such that $A_p \in \mathcal{I}_i$ for all $p \in P$ and all $i \in [k]$. The configuration LP can be formulated in the same way as before, and the separation oracle incurs a loss of $2.7k$ by using the algorithm of [9]. In the following, we prove Theorem 1.5, which states that the integrality gap is still bounded by 5 in this case and implies a $13.5k$-estimation algorithm for max-min allocation subject to k matroid constraints.

▶ **Theorem 1.5.** *The integrality gap of the configuration LP with identical submodular valuations and any number of matroid constraints is at most* 5*.*

Proof. As before, let $\mathsf{OPT} = 1$, $\overline{f}(S) := \min(2, f(S))$ and define A and the y_j-variables in the same way. The proof of Lemma 4.3 directly translates to the matroid constraint setting, so we may again assume $f(j) \leq 1$ for all $j \in J$. Unlike before, we set $y_j = 0$ for all $j \in A_q$, where q is a fixed min player in A. We also set $y_j = 0$ for all unallocated $j \notin A$.

Now suppose for contradiction that there is a set $D \in \bigcap_{i=1}^{k} \mathcal{I}_k$ of items with $f(D) > 5$ but $y(D) < 1$. We can partition D into $D_1 \uplus D_2$ such that $f(D_i) \geq 2$ for both $i \in [2]$, by greedily adding items to D_1 until $f(D_1) > 2$. By the assumption that $f(j) \leq 1$ for all $j \in J$ and submodularity, this implies $f(D_2) = f(D - D_1) > 2$. Now, choose the partition D_i with the smaller y-value, and call it C. As a subset of D, clearly C is independent in all matroids, and $y(C) < \frac{1}{2}$.

Consider reallocating all items from C to player q and discarding all original items in A_q. We want to argue that this modified allocation has higher potential. The gain for player q is $\overline{f}(C) - \overline{f}(A_q) \geq 2 - 1 = 1$. The loss for all other players is

$$\begin{aligned}\sum_{p \in P - \{q\}} \Delta_{\overline{f}}(-C \mid A_p) &\leq \sum_{p \in P - \{q\}} \sum_{j \in A_p} \Delta_{\overline{f}}(-j \mid A_p) \\ &= \sum_{p \in P - \{q\}} \sum_{j \in A_p} y_j \cdot T_{\overline{f}}(p) \\ &\leq y(C) \cdot 2 \\ &< 1\end{aligned}$$

since we can bound $T_{\overline{f}}(p)$ as in the proof of Theorem 1.3. The total potential change is negative. This contradicts our choice of A. Hence, the configuration constraints are all satisfied. The value constraint is satisfied since

$$\sum_{j \in J} y_j = \sum_{j \notin A} y_j + \sum_{j \in A_q} y_j + \sum_{p \in P - \{q\}} \sum_{j \in A_p} y_j = \sum_{p \in P - \{q\}} \sum_{j \in A_p} y_j \leq m - 1 < m. \qquad \blacktriangleleft$$

Notice that we could replace matroids by any set system closed under subsets, but then of course we might not be able to solve the configuration LP efficiently.

Conclusion

We presented an improved approximation algorithm for submodular max-min allocation under identical valuations, and showed some constant upper bounds on the integrality gap of the configuration LP under identical valuations.

However, several natural questions remain open. First, the proposed 0.4-approximation is still far from the best-known complexity-theoretical hardness bound of $1 - \frac{1}{e} \approx 0.63$. Another promising direction is to further explore the configuration LP. It would be valuable to obtain constructive, polynomial-time rounding algorithms that achieve the proven integrality gaps. Furthermore, it would be interesting to see by how much we can relax the assumption of identical valuations. A natural first step in this direction would be to determine the integrality gap of the configuration LP in the case of submodular "related machines", where each player can be satisfied with a different value.

References

1 Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Transactions on Algorithms (TALG)*, 13(3):1–28, 2017. doi:10.1145/3070694.

2 Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms (TALG)*, 8(3):1–9, 2012. doi:10.1145/2229163.2229168.

3 Étienne Bamas, Sarah Morell, and Lars Rohwedder. The submodular santa claus problem. *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 616–640, 2025.

4 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 31–40, 2006. doi:10.1145/1132516.1132522.

5 Siddharth Barman and Sanath Kumar Krishnamurthy. Approximation algorithms for maximin fair division. *ACM Transactions on Economics and Computation (TEAC)*, 8(1):1–28, 2020. doi:10.1145/3381525.

6 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3):11–18, 2005. doi:10.1145/1120680.1120683.

7 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 107–116, 2009.

8 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584, 2010. doi:10.1109/FOCS.2010.60.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792, 2011. doi:10.1145/1993636.1993740.

10 Siu-Wing Cheng and Yuchen Mao. Restricted max-min allocation: Approximation and integrality gap. *arXiv preprint arXiv:1905.06084*, 2019. arXiv:1905.06084.

11 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

12 Mohammad Ghodsi, MohammadTaghi HajiAghayi, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods: Improvements and generalizations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 539–556, 2018.

13 Michel X Goemans, Nicholas JA Harvey, Satoru Iwata, and Vahab Mirrokni. Approximating submodular functions everywhere. *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 535–544, 2009.

14 Daniel Golovin. *Max-min fair allocation of indivisible goods*. School of Computer Science, Carnegie Mellon University, (2005).

15 Penny Haxell and Tibor Szabó. Improved integrality gap in max–min allocation, or, topology at the north pole. *Combinatorica*, 45(2):1–38, 2025.

16 Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

17 Subhash Khot, Richard J Lipton, Evangelos Markakis, and Aranyak Mehta. Inapproximability results for combinatorial auctions with submodular utility functions. *Algorithmica*, 52:3–18, 2008. doi:10.1007/S00453-007-9105-7.

18 Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. Simultaneous placement and scheduling of sensors. *2009 International Conference on Information Processing in Sensor Networks*, pages 181–192, 2009. URL: https://ieeexplore.ieee.org/document/5211932/.

19 David Kurokawa, Ariel D Procaccia, and Nisarg Shah. Leximin allocations in the real world. *ACM Transactions on Economics and Computation (TEAC)*, 6(3-4):1–24, 2018. doi:10.1145/3274641.

20 Frank M Liang. A lower bound for on-line bin packing. *Information processing letters*, 10(2):76–79, 1980. doi:10.1016/S0020-0190(80)90077-0.

21 Vahab Mirrokni, Michael Schapira, and Jan Vondrák. Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions. *Proceedings of the 9th ACM conference on Electronic commerce*, pages 70–77, 2008. doi:10.1145/1386790.1386805.

22 Lukáš Poláček and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. *ACM Transactions on Algorithms (TALG)*, 12(2):1–13, 2015. doi:10.1145/2818695.

23 Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, 22(1):115–124, 1975. doi:10.1145/321864.321873.

24 Steven S Seiden and Gerhard J Woeginger. The two-dimensional cutting stock problem revisited. *Mathematical Programming*, 102(3):519–530, 2005. doi:10.1007/S10107-004-0548-1.

25 Ola Svensson. Santa claus schedules jobs on unrelated machines. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 617–626, 2011. doi:10.1145/1993636.1993718.

26 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004. doi:10.1016/S0167-6377(03)00062-2.

27 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal on Computing*, 40(6):1715–1737, 2011. doi:10.1137/100783352.

28 Gilad Ben Uziahu and Uriel Feige. On fair allocation of indivisible goods to submodular agents. *arXiv preprint arXiv:2303.12444*, 2023. doi:10.48550/arXiv.2303.12444.

29 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 67–74, 2008. doi:10.1145/1374376.1374389.

30 Gerhard J Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20(4):149–154, 1997. doi:10.1016/S0167-6377(96)00055-7.

QPTAS for MWIS and Finding Large Sparse Induced Subgraphs in Graphs with Few Independent Long Holes

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Jadwiga Czyżewska
University of Warsaw, Poland

Tomáš Masařík
University of Warsaw, Poland

Marcin Pilipczuk
University of Warsaw, Poland

Paweł Rzążewski
Warsaw University of Technology, Poland
University of Warsaw, Poland

Abstract

We present a quasipolynomial-time approximation scheme (QPTAS) for the Maximum Independent Set (MWIS) in graphs with a bounded number of pairwise vertex-disjoint and non-adjacent long induced cycles. More formally, for every fixed s and t, we show a QPTAS for MWIS in graphs that exclude sC_t as an induced minor. Combining this with known results, we obtain a QPTAS for the problem of finding a largest induced subgraph of bounded treewidth with given hereditary property definable in Counting Monadic Second Order Logic, in the same classes of graphs.

This is a step towards a conjecture of Gartland and Lokshtanov which asserts that for any planar graph H, graphs that exclude H as an induced minor admit a polynomial-time algorithm for the latter problem. This conjecture is notoriously open and even its weaker variants are confirmed only for very restricted graphs H.

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

Keywords and phrases independent set, long holes, QPTAS, induced subgraphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.9

Funding *Jadwiga Czyżewska*: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.
Tomáš Masařík: Supported by the Polish National Science Centre SONATA-17 grant number 2021/43/D/ST6/03312.
Marcin Pilipczuk: Supported by Polish National Science Centre SONATA BIS-12 grant number 2022/46/E/ST6/00143.
Paweł Rzążewski: Supported by the National Science Centre grant 2024/54/E/ST6/00094.

Acknowledgements The project was initiated at the Structural Graph Theory workshop STWOR in September 2023. This workshop was a part of STRUG: Structural Graph Theory Bootcamp, funded by the "Excellence initiative – research university (2020–2026)" of University of Warsaw. We are grateful to Benjamin Bergougnoux, Linda Cook, and Marek Sokołowski for stimulating discussions on the topic.

© Édouard Bonnet, Jadwiga Czyżewska, Tomáš Masařík, Marcin Pilipczuk, and Paweł Rzążewski;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

In the Max Independent Set (MIS) problem, one is asked, given a graph G, for a largest *independent set*, i.e., a set of pairwise nonadjacent vertices in G. In the weighted variant of the problem, Max Weight Independent Set (MWIS), the input graph has vertex weights and we ask for an independent set of maximum weight. MIS (and thus MWIS) is a "canonical" hard problem – it is one of Karp's 21 NP-hard problems [20], W[1]-hard (with respect to the solution size) [11] and notoriously hard to approximate [19], even in the parameterized setting [7, 21]. Thus, a natural question to ask when dealing with such a hard problem is as follows: *For what classes of input graphs is the MWIS problem tractable?*

Typically, this question is studied for *hereditary* graph classes, i.e., classes closed under vertex deletion. Each such class can be equivalently characterized by specifying minimal induced subgraphs that do not belong to the class. For graphs G and H, we say that G is *H-free* if it does not contain H as an induced subgraph.

The complexity study of MIS and MWIS in restricted graph classes is among the most active research directions in algorithmic graph theory. Let us list some relevant results, focusing on the case of H-free graphs, for a single graph H. Already in the early 1980s, Alekseev [2] observed that classic NP-hardness reductions imply that MIS remains NP-hard in H-free graphs for many graphs H. First, MIS is NP-hard in subcubic graphs, and thus in H-free graphs whenever H has a vertex of degree at least 4 [13]. The second reduction involves the so-called *Poljak's subdivision trick* [29]: subdividing an edge of a graph twice yields a new graph where the size of a maximum independent set increases by exactly one. Consequently, for a fixed graph H, we can start with an arbitrary instance of MIS and subdivide each edge $2|V(H)|$ times. This way we obtain an equivalent instance that is H-free, whenever H has a cycle or two vertices of degree at least three in the same connected component.

Combining these two observations, we obtain that MIS (and thus, MWIS) remains NP-hard in H-free graphs, unless every component of H is a subcubic tree with at most one vertex of degree 3; let us call the family of such forests $\mathcal{S}$. Let us highlight that in particular $\mathcal{S}$ contains all paths and, more generally, linear forests (i.e., forests of paths).

We do not know any NP-hardness result for MWIS in H-free graphs when $H \in \mathcal{S}$. On the other hand, polynomial-time algorithms are known only for small graphs $H \in \mathcal{S}$ [3, 6, 18, 22–24, 30]. However, general belief in the community is that all cases not excluded by standard NP-hardness reductions mentioned above MWIS is polynomial-time-solvable. This belief is supported by the existence of a *quasipolynomial-time* algorithm. Indeed, for every $H \in \mathcal{S}$, the MWIS problem restricted to n-vertex H-free graphs can be solved in time $n^{\mathcal{O}(\log^{16} n)}$ [16]. Note that this is a strong indication that no $H \in \mathcal{S}$ defines an NP-hard case of MWIS, as otherwise every problem in NP can be solved in quasipolynomial time.

Let us have a closer look at the case of P_t-free graphs, i.e., graphs that exclude a t-vertex path as an induced subgraph. It turns out that in these classes we cannot only solve MWIS in quasipolynomial time [15, 28] (and in polynomial time for $t \leqslant 6$ [18, 22]), but the algorithms can actually be extended to a rich family of problems defined as follows [1, 9, 17]. For an integer r and a CMSO_2 sentence[1] ψ, we define the $(\mathsf{tw} \leqslant r, \psi)$-MWIS problem as follows (here, MWIS stands for *maximum-weight induced subgraph*).

[1] CMSO_2 is a logic where one can use vertex, edge, and (vertex or edge) set variables, check vertex-edge incidence, quantify over variables, and apply counting predicates modulo fixed integers. See [11] for a formal introduction.

$(\mathsf{tw} \leqslant r, \psi)$-MWIS

Input: A graph G equipped with a weight function $\mathsf{w}\colon V(G) \to \mathbb{R}_{\geqslant 0}$.

Task: Find a set $S \subseteq V(G)$, such that

- $G[S] \models \psi$,
- $\mathsf{tw}(G[S]) \leqslant r$, and
- S is of maximum weight subject to the conditions above,

or conclude that no such set exists.

Notable special cases of the $(\mathsf{tw} \leqslant r, \psi)$-MWIS problem are MWIS, Feedback Vertex Set (equivalently, Max Induced Forest), and Even Cycle Transversal (equivalently, Max Induced Odd Cactus).

Interestingly, the quasipolynomial-time algorithm for $(\mathsf{tw} \leqslant r, \psi)$-MWIS in P_t-free graphs can be extended to the class of $C_{\geqslant t}$-free graphs: ones that do not contain an induced cycle with at least t vertices. Note that every P_t-free graph is $C_{\geqslant t+1}$-free. We emphasize that $C_{\geqslant t}$-free graphs form the class defined by an infinite minimal family of forbidden induced subgraphs. Furthermore, we cannot hope for tractability of $(\mathsf{tw} \leqslant r, \psi)$-MWIS in H-free graphs, when H is a single graph other than a linear forest. Indeed, recall that if $H \notin \mathcal{S}$, then already MWIS ($r = 0$) is NP-hard in H-free graphs. On the other hand, if $H \in \mathcal{S}$ but is not a linear forest, then it must contain the *claw* – the three-leaf star, and Max Induced Forest ($r = 1$) is NP-hard in the class of claw-free graphs [25].

This bring us to a question: What is the crucial property of P_t-free graphs that also extends to $C_{\geqslant t}$-free graphs, but not to H-free graphs for any fixed H that is not a linear forest, and allows us to solve $(\mathsf{tw} \leqslant r, \psi)$-MWIS efficiently?

It occurs that we should be looking at these classes from a different angle. For graph G and H, we say that H is an *induced minor* of G if it can be obtained from G by deleting vertices and contracting edges. Equivalently, this means that there is an *induced minor model* of H in G: a collection of $|V(H)|$ pairwise disjoint subsets of $V(G)$, each inducing a connected graph, and a bijection that maps these sets the vertices of H so that there is an edge between two sets if and only if their corresponding vertices are adjacent in H. We say that G is *H-induced-minor-free* if it does not contain H as an induced minor. Note that every H-induced-minor-free graph is in particular H-free. However, if H is a linear forest, then H-free graphs and H-induced-minor-free graphs coincide. Furthermore, $C_{\geqslant t}$-free graphs are precisely C_t-induced-minor-free graphs.

The following conjecture asserts that the tractability of $(\mathsf{tw} \leqslant r, \psi)$-MWIS actually extends to all classes excluding a fixed planar graph as an induced minor.

▶ **Conjecture A** (Gartland, Lokshtanov [14]). *For every planar graph H, every problem expressible as $(\mathsf{tw} \leqslant r, \psi)$-MWIS is polynomial-time-solvable in H-induced-minor-free graphs.*

Let us emphasize that the assumption that we exclude a planar graph is crucial in Conjecture A. Indeed, the class of planar graphs excludes any non-planar graph as an induced minor and MWIS is NP-hard in planar graphs [13].

We are still very far from confirming Conjecture A in full generality. Indeed, even its weakening asking for a quasipolynomial-time algorithm, or even a quasipolynomial-time approximation scheme (QPTAS) seems challenging. Still, in recent years some special cases of Conjecture A were shown [4, 5, 12]. In the context of the current paper, the most relevant one seems to be the result of Bonnet et al. [5] that MWIS can be solved in quasipolynomial time in the class of $(C_4 + sC_3)$-induced-minor-free graphs, for every fixed s. Here, by "+" we mean disjoint union and multiplication by s means s-fold disjoint union. Thus, $C_4 + sC_3$ is the graph with $s + 1$ components: one being a C_4 and the remaining ones being triangles.

Thus, in quasipolynomial time we can solve MWIS (and even $(\mathsf{tw} \leqslant r, \psi)$-MWIS) in graphs that exclude *long* induced cycles (i.e., $C_{\geqslant t}$-free) and in graphs that exclude *many* induced and pairwise non-adjacent cycles (i.e., $(C_4 + sC_3)$-induced-minor-free graphs). In this paper, we are interested in the common generalization of these classes: graphs that exclude *many* induced pairwise non-adjacent *long* cycles, i.e., the class of sC_t-induced-minor-free graphs, for fixed s and t.

While we are not able to prove Conjecture A for sC_t-induced-minor-free graphs, we provide a major step towards such a result. As the main technical contribution, we show a QPTAS for MWIS in the considered classes of graphs.

▶ **Theorem 1.** *Let s, t be positive integers and $\varepsilon \in (0,1)$ be a real. There is an algorithm that, given a vertex-weighted graph G, in quasipolynomial time returns either:*

- *an induced minor model of sC_t in G, or*
- *an independent set of weight at least $(1-\varepsilon)$ times the maximum possible weight of an independent set in G.*

The aforementioned quasipolynomial-time algorithm for MWIS in H-free graphs for $H \in \mathcal{S}$ [16] was preceded by a QPTAS in the same graph classes [10]. A recent similar result is a QPTAS for MWIS in graphs excluding a fixed wheel as an induced minor [8].

Next, combining Theorem 1 with known results concerning approximation schemes [17], we obtain a QPTAS for the unweighted variant of $(\mathsf{tw} \leqslant r, \psi)$-MWIS under an additional mild assumption that ψ is a *hereditary* CMSO_2 formula: (i) if $G \models \psi$, then $G' \models \psi$ for every induced subgraph G' of G, and (ii) if $G_1 \models \psi$ and $G_2 \models \psi$, then $G_1 + G_2 \models \psi$. We note that many natural graph properties, like e.g., planarity, bounded degeneracy, or excluding a fixed graph as a minor, can be defined by hereditary CMSO_2 formulas.

▶ **Theorem 2.** *Let $r \geqslant 0$, let s, t be positive integers, $\varepsilon \in (0,1)$ be a real, and ψ be a hereditary CMSO_2 formula. There is an algorithm that, given a graph G, in quasipolynomial time returns one of the following outputs:*

- *an induced minor model of sC_t in G, or*
- *a solution to $(\mathsf{tw} \leqslant r, \psi)$-MWIS of size at least $(1-\varepsilon)$ times the optimum one, or,*
- *a correct conclusion that no solution to $(\mathsf{tw} \leqslant r, \psi)$-MWIS exists.*

2 Preliminaries

For integers k, ℓ, the set $\{k, k+1, \ldots, \ell\}$ is denoted as $[k, \ell]$ and we shorten $[1, k]$ to $[k]$.

We consider here (vertex-)weighted graphs (G, w), where w is a function $V(G) \to \mathbb{R}_{\geqslant 0}$. For a subset X of vertices, we define its weight as $\mathsf{w}(X) = \sum_{v \in X} \mathsf{w}(v)$. For a weighted graph (G, w), by $\alpha(G, \mathsf{w})$ we denote the weight of a maximum-weight independent set in G. We assume that all computations on weights are performed in constant time.

The *open neighborhood* of a vertex v in G, denoted by $N_G(v)$, is the set of vertices adjacent to v. The *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{v\}$. For a subset X of vertices of G, its *open neighborhood* (resp., *closed neighborhood*) we mean the set $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$ (resp., $N_G[X] = \bigcup_{v \in X} N_G[v]$. If G is clear from the context, we omit the subscript and simply write $N(\cdot)$ and $N[\cdot]$.

We say that two disjoint sets $X, Y \subseteq V(G)$ are non-adjacent if there is no edge with one endpoint in X and the other in Y. Since all subgraphs in the paper are induced, we often identify such a subgraph with its vertex set.

For a connected graph G and a set $A \subseteq V(G)$, the *BFS-layering of G from A* is the partition of $V(G)$ into sets $L_0, L_1, \ldots, L_h$ called *layers*, where $L_0 = A$ and for all $i \geqslant 1$, we have $L_i = N(L_{i-1}) \setminus \bigcup_{j<i-1} L_j$, and h is the largest possible so that $L_h \neq \emptyset$.

A *hole* in the graph is an induced cycle with at least 4 vertices. The following easy observation allows us to look for holes of certain length.

▶ **Lemma 3.** *Given a graph G and an integer $t \geqslant 4$, one can in time $\mathcal{O}(|V(G)|^t \cdot (|V(G)| + |E(G)|))$ find a shortest hole in G of length at least t, or conclude that no such hole exists.*

Proof. First, in time $\mathcal{O}(|V(G)|^t)$ we exhaustively check whether G has an induced cycle with exactly t vertices. If such a cycle exists, we return it as it is clearly shortest possible.

Next, for every tuple τ of t distinct vertices of G we check whether τ induces a path. Suppose this is the case and let x, y the endvertices of this path. We remove from G the closed neighborhood of all internal vertices of the path, except for x and y. Finally, we search for a shortest path connecting x and y in the obtained graph; together with the vertices from τ it forms an induced cycle with at least t vertices. We return the shortest of all cycles found in this process, or report that no cycle was found. ◀

Finally, let us recall a result that is particularly useful for constructing QPTASes for MWIS. A vertex v of a weighted graph (G, w) is *γ-heavy* with respect to a set $I \subseteq V(G)$ if $\mathsf{w}(N_G[v] \cap I) \geqslant \gamma \cdot \mathsf{w}(I)$.

▶ **Lemma 4** ([10, Lemma 4.1])**.** *Let (G, w) be a weighted graph on n vertices and $\gamma \in (0, 1)$ be a real number. In time $n^{\mathcal{O}(\log n/\gamma)}$ we can enumerate a family $\mathcal{I}'$ of $n^{\mathcal{O}(\log n/\gamma)}$ independent sets in G, each of size at most $\lceil \gamma^{-1} \log n \rceil$, such that for every independent set I there exists $I' \in \mathcal{I}'$ such that $I' \subseteq I$ and every γ-heavy vertex with respect to I belongs to $N[I']$.*

Intuitively, Lemma 4 is a quasipolynomial approximation-preserving reduction to instances without γ-heavy vertices: we can exhaustively guess $I' \in \mathcal{I}'$ and delete $N[I']$ from the graph.

3 QPTAS for MWIS: Proof of Theorem 1

In this section, we present a QPTAS for MWIS in graphs excluding sC_t as an induced minor.

▶ **Theorem 1.** *Let s, t be positive integers and $\varepsilon \in (0, 1)$ be a real. There is an algorithm that, given a vertex-weighted graph G, in quasipolynomial time returns either:*

- *an induced minor model of sC_t in G, or*
- *an independent set of weight at least $(1 - \varepsilon)$ times the maximum possible weight of an independent set in G.*

Proof. Fix s and t. Since the value of t will be fixed throughout the whole proof, by a *long hole* we mean an induced cycle with at least t vertices. We denote such a long hole shortly by $C_{\geqslant t}$.

Let (G, w) be a weighted graph on n vertices. Let $\varepsilon > 0$ be the desired precision, i.e., we aim for an $(1 - \varepsilon)$-approximation or for finding an induced subgraph isomorphic to $sC_{\geqslant t}$.

Strategy. The algorithm is a typical recursive branching algorithm. Each recursive call is invoked for an induced subgraph G' of G and an integer $s' \leqslant s$; the goal is to either exhibit an induced $s'C_{\geqslant t}$ subgraph of G' or an independent set I' of weight close to $\alpha(G', \mathsf{w})$ (the actual error analysis is made formally later; it is not just merely an $(1 - \varepsilon)$-approximation to $\alpha(G', \mathsf{w})$). The initial call is to $G' = G$ and $s' = s$.

We set

$$\beta \coloneqq \frac{\varepsilon}{s+t+\log_{6/5} n} \quad \text{and} \quad \gamma \coloneqq \frac{\beta^3}{1000t}.$$

The computation of one recursive call is embedded in the following claim.

▷ Claim 5. Given an induced subgraph G' of G and an integer $1 \leqslant s' \leqslant s$, in time $n^{\mathcal{O}(\log n/\gamma)}$ one can either report an induced $s'C_{\geqslant t}$ subgraph in G' or enumerate a family $\mathcal{X}$ of pairs (X, J) such that:

1. for every $(X, J) \in \mathcal{X}$, we have $X \subseteq V(G')$ and J is an independent set in G' with $N[J] \subseteq X$;
2. for every $(X, J) \in \mathcal{X}$, every connected component D of $G' - X$ satisfies one of the following conditions:
   a. there is a $C_{\geqslant t}$ in G' which is disjoint and nonadjacent to D; or
   b. D has at most $\frac{5}{6}|V(G')|$ vertices.
3. for every independent set I' in G' of weight $\alpha(G', \mathsf{w})$, there exists $(X, J) \in \mathcal{X}$ with $J \subseteq I'$ and $\mathsf{w}((X \setminus J) \cap I') \leqslant \beta\alpha(G', \mathsf{w})$;
4. the size of $\mathcal{X}$ is bounded by $n^{\mathcal{O}(\log n/\gamma)}$.

We now argue how Claim 5 yields Theorem 1. Consider a recursive call with parameters (G', s'). If $s' = 0$, then we return $\varnothing$ as an induced $s'C_{\geqslant t}$. If $V(G') = \varnothing$, we return the empty (independent) set.

Otherwise, we apply Claim 5 to G' and s'. If an induced $s'C_{\geqslant t}$ is returned, we return it and conclude. Otherwise, we iterate over the obtained family $\mathcal{X}$. For every $(X, J) \in \mathcal{X}$, let $\mathcal{D}_X$ be the set of connected components of $G' - X$. For every $D \in \mathcal{D}_X$, we recurse on $G'_D \coloneqq G'[D]$ and either $s'_D \coloneqq s'$ if $|D| \leqslant \frac{5}{6}|V(G')|$ (Property 2b) or $s'_D \coloneqq s' - 1$ otherwise. If an induced $s'_D C_{\geqslant t}$ is returned in G'_D, we augment it with a $C_{\geqslant t}$ non-adjacent to D in G' if $s'_D = s' - 1$ (such a long hole can be found using Lemma 3) and return. Otherwise, if an independent set I'_D is returned, we compute an independent set $I_{(X,J)} = \bigcup_{D \in \mathcal{D}_X} I'_D \cup J$ (Note that this is an independent set due to condition $N[J] \subseteq X$.) Finally, if no $s'C_{\geqslant t}$ was returned for any $(X, J) \in \mathcal{X}$, we return $I_{(X,J)}$ of maximum weight among all options $(X, J) \in \mathcal{X}$. This completes the description of the algorithm.

Complexity analysis. Let us move to the analysis. For a recursive call (G', s'), let

$$\mu(G', s') \coloneqq s' + \left\lfloor \log_{6/5} |V(G')| \right\rfloor.$$

Observe that every recursive subcall (G'_D, s'_D) satisfies $\mu(G'_D, s'_D) \leqslant \mu(G', s') - 1$. Hence, the depth of the recursion is bounded by $s + \lfloor \log_{6/5} n \rfloor$. Since every recursive call results in $n^{\mathcal{O}(\log n/\gamma)}$ subcalls, the overall running time is as follows:

$$n^{\mathcal{O}\left(\log^2 n/\gamma\right)} = n^{\mathcal{O}\left(\log^2 n/\beta^3\right)} = n^{\mathcal{O}\left(\log^5 n/\varepsilon^3\right)},$$

i.e., quasipolynomial in n (we hide terms depending on s and t in the $\mathcal{O}(\cdot)$-notation).

Correctness and approximation guarantee. Clearly, if any recursive call (G', s') finds an induced $s'C_{\geqslant t}$, it is propagated up in the recursion tree and results in exhibiting an induced $sC_{\geqslant t}$ in the root call. Assume then that every recursive call (G', s') returned an independent set, which we denote $I_{G',s'}$.

Consider a recursive call (G', s') and let I' be an independent set in G' of weight $\alpha(G', \mathsf{w})$. By the promise of Claim 5, there exists $(X, J) \in \mathcal{X}$ in this recursive call with $J \subseteq I'$ and $\mathsf{w}((X \setminus J) \cap I') \leqslant \beta\alpha(G', \mathsf{w}) = \beta\mathsf{w}(I')$. In particular, $\alpha(G' - X, \mathsf{w}) \geqslant (1 - \beta)\alpha(G', \mathsf{w})$. By

a standard induction on the depth of the subtree of the recursion tree, we obtain that if the depth of the recursion tree of a call (G', s') is h, then $\mathsf{w}(I_{G',s'}) \geqslant (1-\beta)^h \alpha(G', \mathsf{w})$. In particular, since the recursion depth at the root is bounded by $s + \lfloor \log_{6/5} n \rfloor$, the root call returns an independent set of weight at least

$$(1-\beta)^{s+\lfloor \log_{6/5} n \rfloor} \alpha(G, \mathsf{w}) \geqslant \Big(1 - \beta(s + \lfloor \log_{6/5} n \rfloor)\Big)\, \alpha(G, \mathsf{w}) \geqslant (1-\varepsilon)\alpha(G, \mathsf{w}).$$

Thus, it remains to prove Claim 5.

Proof of Claim 5. Let I' be an independent set in G' of weight $\alpha(G', \mathsf{w})$.

We start with a standard application of Lemma 4. We enumerate the family $\mathcal{I}_0$ of $n^{\mathcal{O}(\log n/\gamma)}$ independent sets in G' such that there exists $I_0 \in \mathcal{I}_0$ such that $I_0 \subseteq I'$ and every γ-heavy vertex with respect to I' belongs to $N[I_0]$. Henceforth, we will refer to this choice of I_0 as *the correct choice of* I_0.

Initiate $\mathcal{X} = \varnothing$. We iterate over the elements of $\mathcal{I}_0$. For every $I_0 \in \mathcal{I}_0$ we proceed as follows. Let $X_0 \coloneqq N_{G'}[I_0]$ and $G_0 \coloneqq G' - X_0$.

We find a shortest long hole in G_0 by Lemma 3. If Lemma 3 reports that G_0 is $C_{\geqslant t}$-free, we invoke the exact quasipolynomial-time algorithm of [17] that finds in time $n^{\mathcal{O}(\log^4 n)}$ an independent set J of maximum weight in G_0. We insert $(V(G'), I_0 \cup J)$ into $\mathcal{X}$ and conclude computation for I_0. Note that for the correct choice of I_0, we have $\mathsf{w}(I_0 \cup J) = \alpha(G', \mathsf{w})$.

Assume then that Lemma 3 returns a long hole H. If $|V(H)| \leqslant 2t+8$, then, assuming the correct choice of I_0, we have

$$\mathsf{w}(N_{G_0}[V(H)] \cap I') \leqslant (2t+8)\gamma \cdot \mathsf{w}(I') \leqslant \beta \mathsf{w}(I').$$

Hence, we can insert $(X \coloneqq X_0 \cup N_{G_0}[V(H)], I_0)$ into $\mathcal{X}$ and conclude the computation for the current I_0, as then $G' - X$ is non-adjacent to H, so every connected component of $G' - X$ satisfies Property 2a.

Therefore, from now on, we assume that $|V(H)| > 2t+8$.

▷ Claim 6. Every vertex $v \in V(G_0) \setminus V(H)$ has its neighbors in $V(H)$ included in a 3-vertex subpath of H, or has a neighbor in every subpath of H at least $t-1$ vertices.

Proof. Assume v fails at realizing the latter condition. Then there is a $(t-1)$-vertex subpath P of H, disjoint from the neighborhood of v. Let $u \neq u' \in V(H)$ be such that u and u' are neighbors of v, and delimit a subpath P' of H containing P and containing only two neighbors of v (its endpoints u and u'). Observe that if v has at most one neighbor in $V(H)$, it readily satisfies the first condition of the claim. For $V(P') \cup \{v\}$ not to contradict that H is a shortest long hole of G, vertices u and u' have to be at distance at most 2 in H (along the *other* subpath of H delimited by u and u'), and so v satisfies the first condition. ◁

We first get rid of the vertices satisfying the second condition of Claim 6. Let P be a subpath of H with $t-1$ vertices. We set $X_1 \coloneqq N_{G_0}(P) \setminus V(H)$ and let G_1 be the connected component of $G_0 \setminus X_1$ that contains H.

We will insert $X_0 \cup X_1$ into any output set X in what follows, so one may think of this step as deleting the vertices of X_1. Every connected component of $G_0 \setminus X_1$ except for G_1 satisfies Property 2a, so we need only to focus on G_1.

As $|V(P)| = t-1$, assuming the correct choice of I_0, we have

$$\mathsf{w}(X_1 \cap I') \leqslant (t-1)\gamma \mathsf{w}(I') \leqslant \frac{\beta}{5} \mathsf{w}(I').$$

Note that every vertex satisfying the second condition of Claim 6 lies in X_1.

Let us now consider BFS layering from H in G_1 with layers $V(H) = L_0,\ L_1,\ L_2, \ldots, L_h$. If $h \leqslant \lceil \frac{5}{\beta} \rceil$, we set $h' \coloneqq h + 1$, and otherwise we iterate over all options of $1 \leqslant h' \leqslant \lceil \frac{5}{\beta} \rceil$. In both cases, there exists a choice of h' with $\mathsf{w}(I' \cap L_{h'}) \leqslant \frac{\beta}{5}\mathsf{w}(I')$; we call it henceforth *the correct choice of* h'. For fixed h', we set $X_2 \coloneqq L_{h'}$ and $G_2 \coloneqq G_1[\bigcup_{i=0}^{h'-1} L_i]$. Note that G_2 is connected and every connected component of $G_1 - X_2$ distinct from G_2 lies in $\bigcup_{i>h'} L_i$ and, consequently, is non-adjacent to H and therefore satisfies point 2a. Hence, in what follows we will insert X_2 into any returned set X and we focus on G_2.

Note that $\mathcal{L} \coloneqq L_0, L_1, \ldots, L_{h'-1}$ is the BFS layering in G_2 from H. The following claims and definitions refer to G_2.

For $i < j$, a *vertical path* between $u \in L_i$ and $v \in L_j$ with respect to $\mathcal{L}$ is a path P such that $V(P)$ intersects exactly once every layer L_k such that $k \in [i, j]$, and no other layer. The *cone* of a vertex $v \in V(G_2)$ (still with respect to $\mathcal{L}$) is the set of vertices

$$\{x \in V(H) \mid \text{there is a vertical path between } x \text{ and } v\}.$$

▷ Claim 7. In G_2, let $x \in L_p$, $y \in L_q$, with $p \leqslant q$ such that there exists a path $P \subseteq \bigcup_{i \geqslant p} L_i$ with ℓ vertices, connecting x and y which inner vertices are non-adjacent to H. Let x_1 and y_1 be two vertices in the cones of x and y, respectively. Then, the distance between x_1 and y_1 along H is at most $p + q + \ell$.

Proof. Let x, y, L_p, L_q, ℓ, x_1, y_1 and P be as in the claim statement.

Let Q_x and Q_y be two vertical paths from x and y to x_1 and y_1, respectively. Let x_2 and y_2 be the penultimate vertices of Q_x and Q_y, respectively. Note that $x_2, y_2 \in L_1$ and if $p = 1$, then $x_2 = x$ and $y_2 = y$. Let R be the shortest path in $G_2[V(P) \cup V(Q_x) \cup V(Q_y) \setminus \{x_1, y_1\}]$ connecting x_2 and y_2. Note that R is non-adjacent to H except for the endpoints. Furthermore, $|E(R)| \leqslant (p-1) + (q-1) + \ell$.

Let A and B be the two subpaths of H connecting x_1 and y_1. Without loss of generality, we assume that $|E(A)| \geqslant |E(B)|$. As $|V(H)| > 2t + 8$, we have $|E(A)| \geqslant t + 5$.

Let x_3 and y_3 be the neighbors of x_2 and y_2 on A, respectively, such that no vertex of A between x_3 and y_3 is adjacent to neither x_2 nor y_2. Let A' be the subpath of A from x_3 to y_3. Since the neighborhood in H of a vertex from L_1 is contained in a subpath of at most three consecutive vertices (by Claim 6), we have $|E(A')| \geqslant |E(A)| - 4 \geqslant t$.

We now consider a cycle H' that is a concatenation of x_2x_3, A', y_3y_2, and R. Note that H' is an induced cycle in G_2. As $|E(A')| \geqslant t$, H' is a long hole. As H is the shortest hole of length at least t, we have

$$|E(A)| + |E(B)| = |E(H)| \leqslant |E(H')| = 2 + |E(A')| + |E(R)| \leqslant |E(A)| + p + q + \ell.$$

Hence, $|E(B)| \leqslant p + q + \ell$. This completes the proof. ◁

▷ Claim 8. In G_2, let $x \in L_p$, $y \in L_q$, with $p \leqslant q$ such that there exists a path $P \subseteq \bigcup_{i \geqslant p} L_i$ with ℓ vertices connecting x and y which is disjoint and non-adjacent to H. Then the union of cones of x and y is contained in a subpath of H with at most $2(p + q + \ell)$ vertices.

Proof. Let Z be the union of the cones of x and y. By Claim 7, any two vertices, one belonging to the cone of x and another to the cone of y respectively, in Z lie within distance at most $p + q + \ell$ along H. Hence, Z lies in the subpath of H with at most $2(p + q + \ell)$ vertices. ◁

In the further proof, we use the following two corollaries of Claim 8:

▷ Claim 9. For any p, the cone of any vertex $v \in L_p$ is contained in a subpath of H with at most $4p$ vertices.

▷ Claim 10. Let $v_1 \in L_p$ and $v_2 \in (L_{p-1} \cup L_p) \cap N(v_1)$. Then the union of the cones of v_1 and v_2 is contained in a subpath of H with at most $4p+1$ vertices.

For every $u \in V(H)$, let $D(u) \subseteq V(G_2)$ be the vertices whose cone contains u. Let $u_0, u_1, \ldots, u_{|V(H)|-1}$ be a numbering of the vertices of H along the hole H. We set

$$D_{a,b} \coloneqq \bigcup_{k \in [a, a+b-1]} D(u_{k \bmod |V(H)|}) \quad \text{and } b \coloneqq 4\left\lceil \frac{5}{\beta} \right\rceil + 1.$$

As $p \leqslant h' \leqslant \left\lceil \frac{5}{\beta} \right\rceil$, so $4p < b$. The following claim is a direct consequence of Claim 9.

▷ Claim 11. Two sets $D_{a,b}$ and $D_{a',b}$ are disjoint if $|a - a'| \bmod |V(H)| > 2b$.

The removal of $D_{a,b} \cup D_{a',b}$ disconnects G, provided $D_{a,b}$ and $D_{a',b}$ are disjoint.

▷ Claim 12. Let $a, a' \in [0, |V(H)| - 1]$ be such that $|a - a'| \bmod |V(H)| > 2b$. Then the removal of $D_{a,b} \cup D_{a',b}$ disconnects $\tilde{G}$. In particular $u_{a+b \bmod |V(H)|}$ and $u_{a'+b \bmod |V(H)|}$ are in distinct connected components of $\tilde{G} - (D_{a,b} \cup D_{a',b})$.

Proof. Let $G_3 \coloneqq G_2 - (D_{a,b} \cup D_{a',b})$, and assume without loss of generality that $a < a'$. We want to argue that G_3 is disconnected. By Claim 10, no vertex of $V(G_3) \cap \bigcup_{j \in [a+b, a'-1]} D(u_j)$ can be adjacent to a vertex of $V(G_3) \cap \bigcup_{j \in [0, a-1] \cup [a'+b, |V(H)|-1]} D(u_j)$. Thus in particular u_{a+b} and $u_{a'+b \bmod |V(H)|}$ are in distinct connected components of G'. ◁

Let $n_2 \coloneqq |V(G_2)|$. Fix $0 \leqslant a_1 \leqslant a_2 \leqslant |V(H)| - 1$ minimizing $a_2 - a_1$ subject to

$$\left| \bigcup_{j \in [a_1, a_2]} D(u_j) \right| \geqslant \frac{n_2}{6}. \tag{1}$$

Observe that $a_1 = a_2$ or

$$\left| \bigcup_{j \in [a_1, a_2]} D(u_j) \right| < \frac{n_2}{3}. \tag{2}$$

We set $d \coloneqq 2b \cdot \left\lceil \frac{5}{\beta} \right\rceil$, and distinguish two cases: $a_2 - a_1 \leqslant d$ or $a_2 - a_1 > d$.

Case: $a_2 - a_1 \leqslant d$. Let us consider the family of sets

$$\mathcal{F}_3 \coloneqq \{D_{a_1-d,b}, D_{a_1-d+2b,b}, D_{a_1-d+4b,b}, \ldots, D_{a_1-4b,b}, D_{a_1-2b,b}\}.$$

By Claim 11, the $\lceil \frac{5}{\beta} \rceil$ sets of $\mathcal{F}_3$ are pairwise disjoint, thus there exists $X_3 \in \mathcal{F}_3$ such that $\mathsf{w}(X_3 \cap I') \leqslant \frac{\beta}{5}\mathsf{w}(I')$. For the same reason, there exists

$$X_4 \in \mathcal{F}_4 \coloneqq \{D_{a_2+1,b}, D_{a_2+1+2b,b}, D_{a_2+1+4b,b}, \ldots, D_{a_2+d+1-4b,b}, D_{a_2+d+1-2b,b}\}$$

such that $\mathsf{w}(X_4 \cap I') \leqslant \frac{\beta}{5}\mathsf{w}(I')$. We iterate over all choices of X_3 and X_4. Finally, we exploit the fact that $a_2 - a_1$ is small by adding $X_5 \coloneqq \bigcup_{j \in [a_1, a_2]} N(u_j)$ to the set of vertices to remove. As $d \cdot \gamma \leqslant \frac{\beta}{5}$, for the correct choices of I_0 and h' we have $\mathsf{w}(X_5 \cap I') \leqslant \frac{\beta}{5}\mathsf{w}(I')$. We define $X \coloneqq X_0 \cup X_1 \cup X_2 \cup X_3 \cup X_4 \cup X_5$ and insert (X, I_0) into $\mathcal{X}$. We have $\mathsf{w}(X \cap (I' \setminus I_0)) \leqslant \beta \mathsf{w}(I')$.

We already argued that the connected components of $G' - X$ disjoint from $V(G_2)$ satisfy Property 2a. By Claim 12, $G_2 - (X_3 \cup X_4 \cup X_5)$ has two sets of connected components: those intersecting $\bigcup_{j\in[a_1,a_2]} D(u_j)$, and those not. The former kind are not adjacent to H, so satisfy Property 2a. The latter kind has at most $5n_2/6 \leqslant 5|V(G')|/6$ vertices, by design of the interval $[a_1, a_2]$ using (1), so satisfy Property 2b.

Case: $a_2 - a_1 > d$. Recall that by (2) we have $|\bigcup_{j\in[a_1,a_2]} D(u_j)| < n_2/3$. Then $\bigcup_{j\in[a_1-d,a_1-1]} D(u_j)$ and $\bigcup_{j\in[a_2+1,a_2+d]} D(u_j)$ contains each less than $n_2/6$ vertices, by the minimality of $a_2 - a_1$. Hence $|\bigcup_{j\in[a_1-d,a_2+d]} D(u_j)| < 2n_2/3$. By the pigeonhole principle, as $d = 2b\cdot\lceil\frac{5}{\beta}\rceil$, there is $a \in \{a_1-d, a_1-d+2b, \ldots, a_1-2b\}$ (resp., $a' \in \{a_2+2b, a_2+4b, \ldots, a_2+d\}$) such that $\mathsf{w}(D_{a,b} \cap I') \leqslant \frac{\beta}{5}\mathsf{w}(I')$ (resp., $\mathsf{w}(D_{a',b} \cap I') \leqslant \frac{\beta}{5}\mathsf{w}(I')$). We iterate over all choices of a and a' and set $X_3 := D_{a,b}$, $X_4 := D_{a',b}$. We take $X := X_0 \cup X_1 \cup X_2 \cup X_3 \cup X_4$ and insert (X, I_0) into $\mathcal{X}$. For the correct choice of I_0, h', a, a' we have $\mathsf{w}(X \cap (I' \setminus I_0)) \leqslant \beta\mathsf{w}(I')$.

Observe finally that $G_2 - (X_3 \cup X_4)$ has no connected component of size larger than $2n_2/3$, hence larger than $5|V(G')|/6$. This completes the proof of Claim 5 and thus, of Theorem 1. ◀

4 QPTAS for $(\mathsf{tw} \leqslant r, \psi)$-MWIS: Proof of Theorem 2

The proof of Theorem 2 is based on the approach introduced by Gartland et al. [17, Section 4]. For a graph G, we define its *blob graph* G° as follows:

$$V(G^\circ) = \{X \subseteq V(G) \mid G[X] \text{ is connected}\}$$
$$E(G^\circ) = \{XY \mid G[X \cup Y] \text{ is connected}\}.$$

Equivalently, edges join sets that are either non-disjoint, or there is an edge from one set to another. Gartland et al. [17] proved that for every $t \geqslant 4$, a graph G is $C_{\geqslant t}$-free if and only if G° is $C_{\geqslant t}$-free. First, let us extend this result to the setting of H-induced-minor-free graphs, where every component of H is a hole. The proof closely follows the proof of Paesani et al. for the case if H is a linear forest [27].

▶ **Theorem 13.** *Let H be a graph whose every component is a hole. The graph G contains H as an induced minor if and only if G° contains H as an induced minor.*

Proof. As G is an induced subgraph of G°, the forward implication is immediate. We prove the backward implication by induction on the number s of connected components of H. The case $s = 1$ was shown by Gartland et al. [17]. Thus, assume that $s > 1$ and let C be a connected component of H. Let $H' = H - C$.

Suppose that G° contains H as an induced minor. Fix one induced minor model of H in G and let $\mathcal{X}$ be the set of vertices of the model. This means that $G^\circ[\mathcal{X}]$ has s components, each of which is a cycle, and there is a one-to-one mapping from the components of $G^\circ[\mathcal{X}]$ and components of H, so that each cycle is mapped to a cycle of at most its own length.

Let $\mathcal{Y} \subseteq \mathcal{X}$ be the vertices of G° that form the cycle mapped to C; each element of $\mathcal{Y}$ is a subset of $V(G)$. Let $Y \subseteq V(G)$ be the union of all sets in $\mathcal{Y}$. Note that $G^\circ[\mathcal{Y}]$ is an induced subgraph of $(G[Y])^\circ$. Thus, by the inductive assumption, $G[Y]$ contains an induced cycle with at least $|V(C)|$ vertices.

Now let $X \subseteq V(G)$ be the union of sets in $\mathcal{X} \setminus \mathcal{Y}$. Since $\mathcal{Y}$ is the vertex set of one component of $G^\circ[\mathcal{X}]$, there are no edges between $\mathcal{X} \setminus \mathcal{Y}$ and $\mathcal{Y}$. Consequently, there are no edges in G between X and Y, and thus $G^\circ[X \setminus Y]$ is an induced subgraph of $(G - N[Y])^\circ$.

Since $G^\circ[\mathcal{X} \setminus \mathcal{Y}]$, and thus $(G - N[Y])^\circ$, contains H' as an induced minor, by the inductive assumption we know that $G - N[Y]$ contains H' as an induced minor. Combining its model with the model of C in $G[Y]$, we obtain an induced minor model of H in G. ◀

Let $\mathcal{H}$ be a class of graphs. An *induced $\mathcal{H}$-packing* in a graph G is a set $S \subseteq V(G)$ such that every component of $G[S]$ belongs to $\mathcal{H}$. For example, if $\mathcal{H} = \{K_1\}$, then induced $\mathcal{H}$-packing in G if and only if it is an independent set.

The construction of blob graphs allows us to reduce the problem of finding an induced $\mathcal{H}$-packing of maximum size (or weight) in G to solving the MWIS problem in an appropriate induced subgraph of G°. This is expressed in the following lemma whose proof is immediate, see e.g., [17, Section 4].

▶ **Lemma 14.** *Let G be a graph, $\mathsf{w} : V(G) \to \mathbb{R}_{\geqslant 0}$ be a weight function, and $\mathcal{H}$ be any class of graphs. Let $\mathcal{X} = \{X \subseteq V(G) \mid G[X] \in \mathcal{H} \text{ and is connected}\}$. Let $\mathsf{w}^\circ : \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ be defined as $\mathsf{w}^\circ(X) = \sum_{v \in X} \mathsf{w}(v)$. Then the maximum weight of an induced $\mathcal{H}$-packing in G is equal to $\alpha(G^\circ[\mathcal{X}], \mathsf{w}^\circ)$.*

The problem with using Lemma 14 is that $\mathcal{X}$ might be very large, so the running time of the algorithm solving MWIS on $(G^\circ[\mathcal{X}], \mathsf{w}^\circ)$ is not bounded by a moderate function of $|V(G)|$.

A class $\mathcal{H}$ of graphs is *weakly hyperfinite* if for every $\delta > 0$ there exists $c_\delta \in \mathbb{N}$ such that for any $G \in \mathcal{H}$ there exists a set $S \subseteq |V(G)|$ of size at most $\delta \cdot |V(G)|$ so that every component of $G - S$ has at most c_δ vertices [26, Section 16.2]. The following theorem follows the result of Gartland et al. [17]. Again, we include the proof for completeness.

▶ **Theorem 15.** *Let $\mathcal{H}$ be a non-empty hereditary weakly hyperfinite class of graphs. Let s, t be positive integers and $\varepsilon \in (0, 1)$ be a real. There is an algorithm that, given a graph G, in quasipolynomial time returns one of the following outputs:*

- *an induced minor model of sC_t in G, or*
- *an induced $\mathcal{H}$-packing of size at least $(1 - \varepsilon)$ times the size of a largest induced $\mathcal{H}$-packing in G.*

Proof. Without loss of generality assume that $t \geqslant 4$. Let $\delta = \varepsilon/2$ and let c_δ be the constant witnessing that $\mathcal{H}$ is weakly hyperfinite. Let G be an n-vertex instance instance of the problem. Let

$$\mathcal{X} = \{X \subseteq V(G) \mid |X| \leqslant c_\delta \text{ and } G[X] \text{ is a connected graph from } \mathcal{H}\}.$$

Consider the graph $G^\circ[\mathcal{X}]$; note that it has $\mathcal{O}(n^{c_\delta})$ vertices and can be constructed in time polynomial in n as c_δ is a constant. We call the algorithm from Theorem 1 for $G^\circ[\mathcal{X}]$, weigths w° defined as in Lemma 14, and precision $\varepsilon/2$. It running time is quasipolynomial in $\mathcal{O}(n^{c_\delta})$ and thus in n.

If an induced minor of sC_t is reported, we report an induced minor of sC_t in G. Indeed, recall that $G^\circ[\mathcal{X}]$ is an induced subgraph of G°. Consequently, by Theorem 13, if $G^\circ[\mathcal{X}]$ contains sC_t as an induced minor, so does G.

Thus, suppose that the algorithm from Theorem 1 returns an induced $\mathcal{H}$-packing $\mathcal{S}$ in $G^\circ[\mathcal{X}]$. It is straightforward to verify that $S = \bigcup \mathcal{S}$ is an induced $\mathcal{H}$-packing in G. Let us argue how its size compares to the optimum one.

Let S^* be an optimum induced $\mathcal{H}$-packing in G. Let us construct another induced $\mathcal{H}$-packing S' as follows. We consider every component of $G[S^*]$ separately, let C be the vertex set of one such component. If $|C| \leqslant c_\delta$, we set $C' = C$ and include this set in S'. Otherwise, since $G[C]$ is weakly hyperfinite, there is a set $Q_C \subseteq C$ of size at most $\delta|C| = \varepsilon/2 \cdot |C|$, such that every component of $G[C \setminus Q_C]$ has at most c_δ vertices. We set $C' = C \setminus Q_C$ and include it into S'. Since $\mathcal{H}$ is hereditary, it holds that $G[C'] \in \mathcal{H}$. Note that in both cases we have $|C'| \geqslant (1 - \delta)|C| = (1 - \varepsilon/2)|C|$. Consequently, we obtain

$$|S^*| = \sum_{C\text{: component of } G[S^*]} |C| \leqslant \sum_{C\text{: component of } G[S^*]} \frac{|C'|}{(1 - \varepsilon/2)} = \frac{|S'|}{(1 - \varepsilon/2)}.$$

Now, by Lemma 14 notice that the size of S' is equal to the weight of a largest-weight independent set in $G^\circ[\mathcal{X}]$. Thus, by Theorem 1, we have $|S| \geqslant (1-\varepsilon/2)|S'|$. Summarizing, it holds that

$$|S| \geqslant (1-\varepsilon/2)|S'| \geqslant (1-\varepsilon/2)(1-\varepsilon/2)|S^*| = (1-\varepsilon+\varepsilon^2)|S^*| \geqslant (1-\varepsilon)|S^*|.$$

This completes the proof. ◀

Now let us argue how Theorem 15 implies Theorem 2.

▶ **Theorem 2.** *Let $r \geqslant 0$, let s, t be positive integers, $\varepsilon \in (0,1)$ be a real, and ψ be a hereditary CMSO$_2$ formula. There is an algorithm that, given a graph G, in quasipolynomial time returns one of the following outputs:*

- *an induced minor model of sC_t in G, or*
- *a solution to $(\mathsf{tw} \leqslant r, \psi)$-MWIS of size at least $(1-\varepsilon)$ times the optimum one, or,*
- *a correct conclusion that no solution to $(\mathsf{tw} \leqslant r, \psi)$-MWIS exists.*

Proof. Note that if ψ is *consistent*, i.e., if there is any graph H such that $H \models \psi$, then, since ψ is hereditary, we have $K_1 \subseteq \psi$. Consequently, in such a case, every non-empty instance of $(\mathsf{tw} \leqslant r, \psi)$-MWIS has a solution (for example, a single vertex). Since ψ is a fixed formula, we can verify if it is consistent in constant time. If not, we return the third outcome.

So from now on let us assume that some solution exists. It is known that graphs of treewidth at most r, for any fixed r, form a weakly hyperfinite class of graphs [26, Theorem 16.5]. Furthermore, this class is closed under vertex deletion and disjoint unions. Consequently, graphs of treewidth at most r satisfying ψ form a non-empty class closed under vertex deletion and disjoint unions. Thus, we obtain Theorem 2 as an immediate corollary of Theorem 15. ◀

5 Conclusion

Let us conclude the paper with pointing out two specific problems for further research. First, it would be interesting to strengthen our Theorem 1 by solving the problem *exactly* without significantly increasing the running time.

▶ **Problem P1.** *Show that for every fixed s, t, MWIS can be solved in quasipolynomial time in sC_t-induced-minor-free graphs.*

Second, let us suggest the following possible extension of our Theorem 1. Our starting point was a quasipolynomial-time algorithm for H-induced-minor-free graphs and we aimed to extend this result for H'-induced-minor-free graphs, where every component of H is isomorphic to H. Perhaps this can be done in a more general setting?

▶ **Problem P2.** *Suppose that H_1 and H_2 are graphs such that MWIS is (quasi)polynomial-time solvable in H_i-induced-minor-free graphs for $i \in \{1,2\}$. Show a quasipolynomial-time algorithm (or at least a QPTAS) for MWIS in $(H_1 + H_2)$-induced-minor-free graphs.*

We remark that the solution to Problem P2 is known if instead of forbidding certain graphs as induced minors, we forbid them as induced subgraphs [15]. However, the approach relies on the fact that induced subgraphs are constant-size objects, while an induced minor model of a constant-size graph might be arbitrarily large.

References

1 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, Paweł Rzążewski, and Paul D. Seymour. Induced subgraphs of bounded treewidth and the container method. *SIAM J. Comput.*, 53(3):624–647, 2024. doi:10.1137/20M1383732.

2 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.

3 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1):3–16, 2004. Russian Translations II. doi:10.1016/S0166-218X(02)00290-1.

4 Marthe Bonamy, Édouard Bonnet, Hugues Déprés, Louis Esperet, Colin Geniet, Claire Hilaire, Stéphan Thomassé, and Alexandra Wesolek. Sparse graphs with bounded induced cycle packing number have logarithmic treewidth. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3006–3028. SIAM, 2023. doi:10.1137/1.9781611977554.CH116.

5 Édouard Bonnet, Julien Duron, Colin Geniet, Stéphan Thomassé, and Alexandra Wesolek. Maximum independent set when excluding an induced minor: $K_1 + tK_2$ and $tC_3 \uplus C_4$. *Algorithmica*, 88(1):16, 2026. doi:10.1007/S00453-025-01356-2.

6 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent set for ℓclaw-free graphs in polynomial time. *Discret. Appl. Math.*, 237:57–64, 2018. doi:10.1016/J.DAM.2017.11.029.

7 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-Exponential Time Hypothesis to fixed parameter tractable inapproximability: Clique, dominating set, and more. *SIAM J. Comput.*, 49(4):772–810, 2020. doi:10.1137/18M1166869.

8 Maria Chudnovsky, Jochen Pascal Gollin, Matjaž Krnc, and Martin Milanič. Dominated balanced separators in wheel-induced-minor-free graphs. *CoRR*, abs/2512.12329, 2025. doi:10.48550/arXiv.2512.12329.

9 Maria Chudnovsky, Rose McCarty, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Sparse induced subgraphs in P_6-free graphs. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 5291–5299. SIAM, 2024. doi:10.1137/1.9781611977912.190.

10 Maria Chudnovsky, Marcin Pilipczuk, Michal Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs. *SIAM J. Comput.*, 53(1):47–86, 2024. doi:10.1137/20M1333778.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. III. Tree-independence number of graphs with a forbidden structure. *Journal of Combinatorial Theory, Series B*, 167:338–391, 2024. doi:10.1016/j.jctb.2024.03.005.

13 M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

14 Peter Gartland. *Quasi-Polynomial Time Techniques for Independent Set and Beyond in Hereditary Graph Classes*. PhD thesis, University of California Santa Barbara, 2023.

15 Peter Gartland and Daniel Lokshtanov. Independent set on P_k-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

16 Peter Gartland, Daniel Lokshtanov, Tomás Masařík, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Maximum weight independent set in graphs with no long claws in quasi-polynomial time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 683–691. ACM, 2024. doi:10.1145/3618260.3649791.

17 Peter Gartland, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Finding large induced sparse subgraphs in $C_{>t}$-free graphs in quasipolynomial time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 330–341. ACM, 2021. doi:10.1145/3406325.3451034.

18 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for maximum weight independent set on P_6-free graphs. *ACM Trans. Algorithms*, 18(1):4:1–4:57, 2022. doi:10.1145/3414473.

19 Johan Håstad. Clique is hard to approximate within $n^{(1-\epsilon)}$. In *Acta Mathematica*, pages 627–636, 1996.

20 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pages 85–103, Boston, MA, 1972. Springer US. doi:10.1007/978-1-4684-2001-2_9.

21 Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Improved hardness of approximating k-Clique under ETH. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 285–306. IEEE, 2023. doi:10.1109/FOCS57990.2023.00025.

22 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581. SIAM, 2014. doi:10.1137/1.9781611973402.43.

23 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.

24 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

25 Andrea Munaro. On line graphs of subcubic triangle-free graphs. *Discret. Math.*, 340(6):1210–1226, 2017. doi:10.1016/J.DISC.2017.01.006.

26 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

27 Giacomo Paesani, Daniël Paulusma, and Paweł Rzążewski. Feedback Vertex Set and Even Cycle Transversal for H-free graphs: Finding large block graphs. *SIAM J. Discret. Math.*, 36(4):2453–2472, 2022. doi:10.1137/22m1468864.

28 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for independent set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

29 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15:307–309, 1974.

30 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.

Robotic Arm Rotation: Standing up Is Harder Than You Think

Nicolas Bousquet
CNRS, Université de Montréal CRM - CNRS, Canada
Univ. Lyon, CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69621, Lyon, France

Frank Connor
McGill University, Montreal, Canada

Remy El Sabeh
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Louis-Roy Langevin
McGill University, Montreal, Canada

Amer E. Mouawad
American University of Beirut, Lebanon
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Naomi Nishimura
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Agnes Totschnig
McGill University, Montreal, Canada

Abstract

We study motion-planning problems for planar robotic arms that rotate around fixed centers while avoiding collisions. In the SM-RAMP model, each unit-length arm may rotate at most once; the question is whether all arms can be rotated to the vertical position. We resolve an open problem of Bousquet et al. [1] by proving that SM-RAMP is NP-complete, even in the horizontal-to-vertical setting. Our hardness proof uses a structural analysis of rotation-propagation chains and introduces a combinatorial abstraction of independent interest, the Lighthouse Propagation problem, which we show is itself NP-complete. We then consider the multi-move variant MM-RAMP, where each arm may rotate multiple times among a fixed set of allowed angles (or orientations). We prove that MM-RAMP is PSPACE-complete even when each arm has only a few allowed angles, in sharp contrast with the single-move case. Finally, we give two fixed-parameter tractable algorithms: for MAX-SM-RAMP parameterized by the number k of arms to be made vertical, and for 2A-MM-RAMP (restricted to horizontal and vertical) parameterized by the number ℓ of allowed rotations.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases search, optimization, robotics, robotic arms, parameterized complexity, computational geometry, combinatorial reconfiguration

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.10

Funding *Remy El Sabeh and Naomi Nishimura*: supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).
Nicolas Bousquet: supported by the ANR (Agence Nationale de la Recherche) project TOAD (ANR-25-CE48-3716).
Agnes Totschnig: supported by a McCall MacBain Scholarship and partially by FRQNT Grant 332481.

© Nicolas Bousquet, Frank Connor, Remy El Sabeh, Louis-Roy Langevin, Amer E. Mouawad, Naomi Nishimura, and Agnes Totschnig;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Motion planning for multiple robots operating in a shared workspace is a fundamental problem in robotics, studied under both centralized [13] and decentralized [7, 19] frameworks. The core challenge is to design collision-free motions that transform a given initial configuration of robots into a desired target configuration. While sampling-based algorithms often perform well in practice, they provide limited guarantees and do not capture the combinatorial structure underlying interference constraints. Motivated by this, we adopt a reconfiguration viewpoint, in which we study reachability questions on highly restricted robotic systems where collisions arise solely from geometric interactions.

Combinatorial reconfiguration problems were formally introduced in algorithmic form by Ito et al. [14], who studied how to transform one feasible solution (or *configuration*) of a combinatorial problem into another through a sequence of small, valid changes. Since then, the area has grown into a broad framework encompassing graph problems, geometric objects, and combinatorial optimization. The Nondeterministic Constraint Logic (NCL) framework of Hearn and Demaine [6, 11, 12] has been used as a central tool in this setting to prove PSPACE-completeness of many natural puzzles and motion-planning problems. The resulting theory features both rich algorithmic structure and numerous hardness results [2, 4, 5, 15, 17, 18]. For an overview of developments in the area, we refer the reader to the surveys [3, 16, 21].

In this paper, we continue the study of extremely restricted robotic systems [1]. Each robot is a simple robotic arm represented by a unit-length line segment in the plane which can move only by rotating around its midpoint, either clockwise or anticlockwise. A configuration (specified midpoints and angles of each arm) is *conflict-free* if no two arms intersect; moreover, no arm is allowed to intersect any other arm at any point during its rotation. Despite their simplicity, these systems encode surprisingly rich geometric dependencies that make even very restricted motion-planning questions computationally complex.

We first study the simplest such question in the Single-Move Robotic Arm Motion Planning (SM-RAMP) problem [1]: starting from an initial configuration, can all arms be made vertical by at most one rotation per arm? A solution to the problem consists of a sequence of intermediate configurations, each obtained by rotating exactly one arm to the vertical orientation. Since each arm has only a single opportunity to rotate, the order in which we choose to rotate the arms is crucial, e.g., rotating one arm too early may block the rotation of another arm that lies "behind" it. Intuitively, SM-RAMP asks whether it is possible to choose an ordering in which to "stand up" the arms one by one.

Previous work proved NP-completeness of related problems, but left open the complexity of SM-RAMP [1]. We resolve this open problem by proving that SM-RAMP is NP-complete, even when all arms are initially horizontal. A horizontal-to-vertical (H–V) instance eliminates all freedom in the choice of angles and imposes an extremely regular structure on allowed rotations, yet remains computationally intractable.

▶ **Theorem 1.** *SM-RAMP is* NP*-complete, even when restricted to H–V instances where every robotic arm interacts with at most three others.*

To prove Theorem 1, we introduce the Lighthouse Propagation (LPR) problem, where chains (sequences of arms) are abstracted into rectilinear light rays; doing so allows us to break the quite technical reduction of Theorem 1 into two parts. Given a set of lighthouses, positioned at points in a grid, where each lighthouse L_i can illuminate either of two light ray *bundles* X_i or Y_i (oriented subtrees rooted at L_i), the LPR problem consists of choosing, for each i, either X_i or Y_i so that the union of the chosen bundles is acyclic. We

define the behavior of these light rays to match the behavior of chains in SM-RAMP. This abstraction allows us to design gadgets to encode instances of Planar 3-Bounded 3-SAT, an NP-complete problem, into instances of LPR with reflection count at most three, that is, where each light ray intersects at most three others.

▶ **Theorem 2.** *Lighthouse Propagation is* NP*-complete, even when restricted to instances with a reflection count of at most three.*

The LPR reduction (Theorem 2, deferred to Section 3) contains all the combinatorial ingredients and explains how we will use chains to derive hardness, without having to account for two important aspects of an SM-RAMP instance. First, we need not be concerned with the exact positions of the midpoints of the robotic arms, which allows us to bypass much of the geometry of the problem. Second, we assume in the definition of the problem that, without having to design gadgets, we have the desired propagation behaviour. We will handle these two points in the reduction from LPR to SM-RAMP (in Section 4).

While SM-RAMP restricts each arm to a single rotation, the original framework [1] and most applications in robotics also consider more general reconfiguration problems where arms (and more generally robots) may rotate multiple times among a finite set of allowed angles. This leads to the Multiple-Move Robotic Arm Motion Planning problem (MM-RAMP), which generalizes SM-RAMP by allowing each arm to rotate more than once and by specifying multiple allowed angles. We are given initial and target conflict-free configurations of a set of robotic arms. Each arm has a specified finite set of allowed angles, including its starting and ending angles. A valid reconfiguration sequence is a sequence of conflict-free intermediate configurations that transforms the initial configuration into the target one, moving the robotic arms one by one. As in SM-RAMP, each step may rotate only a single arm, which must be intersection-free throughout its rotation. However, unlike in SM-RAMP, an arm may rotate multiple times over the course of the sequence.

Intuitively, MM-RAMP asks whether it is possible to "navigate" each arm through its allowed angles, one movement at a time, without ever intersecting another arm. The challenge lies not only in finding a feasible path for each individual arm, but also in orchestrating movements so that rotations of one arm do not obstruct the movements of others. This makes MM-RAMP much more expressive and significantly more complex than its single-move counterpart. We establish the intrinsic difficulty of MM-RAMP by showing that it is PSPACE-complete, even when each arm is allowed only a constant number of angles.

▶ **Theorem 3.** *MM-RAMP is* PSPACE*-complete, even when the total number of distinct angles is at most* 5*, and when the number of possible angles per robotic arm is at most* 3*.*

Complementing these hardness results, we identify two settings in which RAMP problems become tractable from a parameterized perspective. Positive results in such constrained geometric reconfiguration settings appear to be scarce, and our findings suggest that parameterization may be a promising avenue toward identifying nontrivial tractable cases.

- Let MAX-SM-RAMP denote a variant of SM-RAMP where arms can initially be either horizontal or vertical and the goal is to determine whether there exist at least k arms that can be rotated from horizontal to vertical (allowing at most one rotation per arm). We show that MAX-SM-RAMP is fixed-parameter tractable (and admits a polynomial kernel) when parameterized by k.
- Let 2A-MM-RAMP denote a variant of MM-RAMP allowing only two admissible orientations per robotic arm. Given an integer ℓ, the goal is to determine whether there exists a sequence of at most ℓ rotations that can transform the initial configuration to

the target one (allowing multiple rotations per arm). We show that 2A-MM-RAMP is fixed-parameter tractable when parameterized by ℓ, provided only the horizontal and vertical orientations are allowed.

These results are modest in scope, but they indicate that a careful choice of parameters can help isolate tractable structure within RAMP problems.

Collectively, our results demonstrate that even extremely constrained geometric motion-planning problems encode deep combinatorial structure. The RAMP framework thus serves as a minimal yet expressive setting for understanding the complexity landscape of intersection-free reconfiguration. More broadly, we believe that the problems and techniques introduced here, most notably the structural analysis of rotation dependencies and the lighthouse propagation abstraction, could provide a foundation for studying hardness in other highly constrained geometric reconfiguration systems. These models capture fundamental propagation mechanisms that arise whenever simple geometric objects interact through rigid interference constraints, and we hope that they will serve as starting points for establishing complexity results in a wider class of geometric reconfiguration problems.

Due to space constraints, all missing (formal) proofs and all details about the MM-RAMP results can be found in the full version of the paper.

2 The SM-RAMP problem

The Robotic Arm Motion Planning (RAMP) problems are defined in the plane $\mathbb{R}^2$. A *robotic arm* is a unit-length segment with midpoint (x, y) and angle $a \in [0, \pi)$, measured from the horizontal. Each arm r has an associated circle and square centered at its midpoint, used only to define geometric constraints. A set of arms is *conflict-free* if their segments are pairwise non-intersecting. We focus on *horizontal* arms ($a = 0$) and *vertical* arms ($a = \pi/2$).

A reconfiguration step rotates a single arm r_1 about its midpoint, changing a to $a' = (a + \alpha) \bmod \pi$ with $-\pi < \alpha < \pi$. If the swept sector intersects another arm r_2 (even in a single point), the rotation *collides* with r_2. In the Single-Move RAMP (SM-RAMP) problem, each arm may rotate at most once. Given a conflict-free set of arms, we ask whether all arms can be made vertical via a sequence of collision-free reconfiguration steps.

2.1 Definitions and initial observations

The solution to a RAMP problem, or *rotation ordering*, is a pair $(\mathcal{R}_{\text{sol}}, \mathcal{D}_{\text{sol}})$, where $\mathcal{R}_{\text{sol}} = (r'_1, \ldots, r'_n)$ lists the arms in $\mathcal{R} = \{r_1, \ldots, r_n\}$ in the order in which they rotate and $\mathcal{D}_{\text{sol}} = (d_1, \ldots, d_n)$, $d_i \in \{\mathsf{C}, \mathsf{A}\}$, records clockwise/anticlockwise rotation. Properties of a rotation ordering are characterized by two types of constraints (Figure 1) between arms r_i and r_j:

- A clockwise/anticlockwise *precedence* constraint on r_i (involving r_j) means r_j must rotate before r_i can rotate in that direction (Figures 1(a) and 1(b)).
- A clockwise/anticlockwise *succession* constraint on r_i (involving r_j) means that if r_j has already rotated, then rotating r_i in that direction would cause a collision, so r_i must rotate before r_j (Figures 1(c) and 1(d)).

We encode constraints in four directed graphs $P_{\mathsf{C}}, P_{\mathsf{A}}, S_{\mathsf{C}}, S_{\mathsf{A}}$ on vertex set $\mathcal{R}$: P (resp. S) encodes precedence (resp. succession), and the subscript indicates rotation direction. In P_{X}, an arc (r_i, r_j) means r_j must rotate before r_i if r_i rotates in direction X; in S_{X}, an arc (r_i, r_j) means r_i must rotate before r_j if r_j rotates in direction X. If any such constraint exists, arms r_i and r_j *interact*.

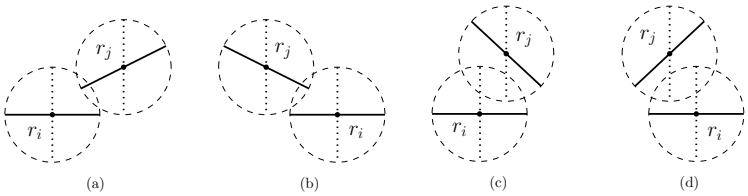


Figure 1 Precedence and succession constraints between two arms. Target orientations are depicted by dashed line segments, while initial orientations are depicted by solid lines.

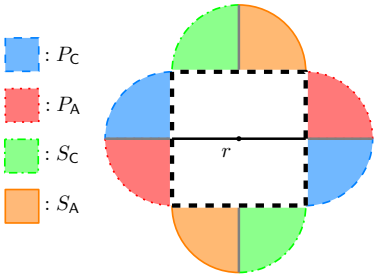


Figure 2 A flower of an arm: the petal containing another arm's midpoint determines the constraint.

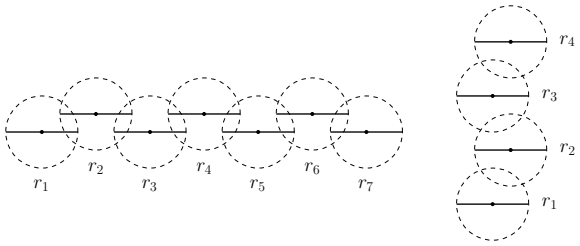


Figure 3 A horizontal chain (left) and a vertical chain (right).

Given a direction assignment $\delta : \mathcal{R}' \to \{\mathsf{C}, \mathsf{A}\}$ with $\mathcal{R}' \subseteq \mathcal{R}$, we call δ *compatible* if there is a rotation ordering in which each $r \in \mathcal{R}'$ rotates according to $\delta(r)$. The associated *rotation dependency graph* G_δ has vertex set $\mathcal{R}'$ and an arc (r_i, r_j) whenever $r_j \in P_{\delta(r_i)}$ or $r_i \in S_{\delta(r_j)}$. Then δ is compatible if and only if G_δ is acyclic. If $\mathcal{R}' = \mathcal{R}$, any reverse topological ordering of G_δ yields a rotation ordering.

In a horizontal-to-vertical (H–V) instance, all arms start horizontal and must become vertical. We capture the geometry of constraints via *flowers* (Figure 2): we place eight quarter-circular regions (*petals*) around an arm's square, each petal being of type $P_\mathsf{C}, P_\mathsf{A}, S_\mathsf{C}$, or S_A. We observe that, for any type in $\{P_\mathsf{C}, P_\mathsf{A}, S_\mathsf{C}, S_\mathsf{A}\}$, if the midpoint of r_j lies in a petal of that type of r_i, then the midpoint of r_i lies in a petal of the same type of r_j. Thus every interacting pair appears in exactly one of the four graphs and every constraint is bidirectional. This structural property distinguishes H–V instances from the general case and underlies both hardness and positive results (Section 5).

2.2 Chains in SM-RAMP instances

We now introduce *chains*, the key structured gadgets used in our hardness proof. A *horizontal chain* is a sequence $r_1, \ldots, r_h$ of horizontal arms whose pairwise interactions use only P-constraints, with symmetric constraints only between consecutive arms and no constraints between non-consecutive arms. Analogously, a *vertical chain* uses only S-constraints. Neighboring arms in a chain always rotate in opposite directions (Figure 3).

In a horizontal chain, only the two ends can rotate first; rotations then propagate from the ends toward an arm at the center of the chain, which rotates last. In a vertical chain, any arm may rotate first, but once the first arm and its direction are fixed, the directions and order of all other arms are forced; the pattern is essentially the reverse of that of a horizontal chain. Choosing a rotation direction for some arm r_i in a chain induces a *propagation*: a forced ordering and directions on a set of *constrained* arms, including *root* r_i.

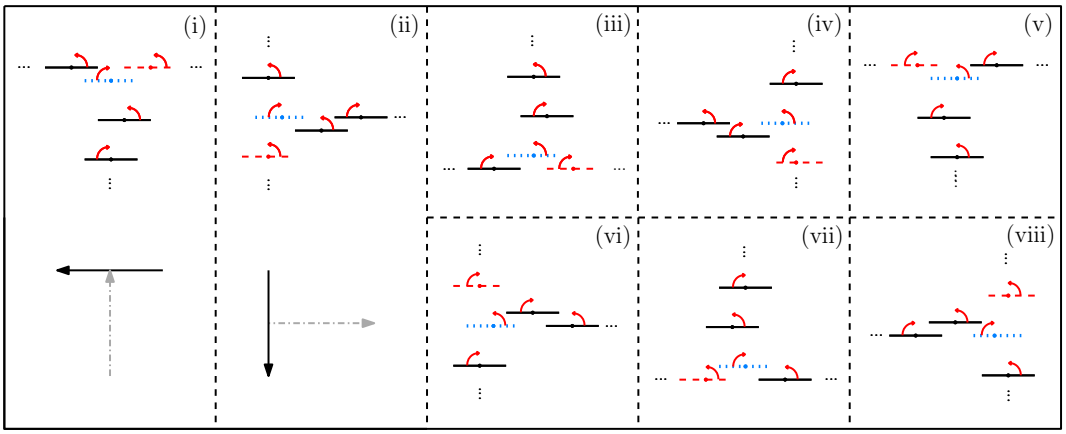


Figure 4 The eight local propagation patterns between intersecting chains, together with arrow representations for cases (i) and (ii). In each pattern, propagation is rooted at the red dashed arm, whose rotation direction is indicated, and the dotted blue arm is the arm shared by the two chains.

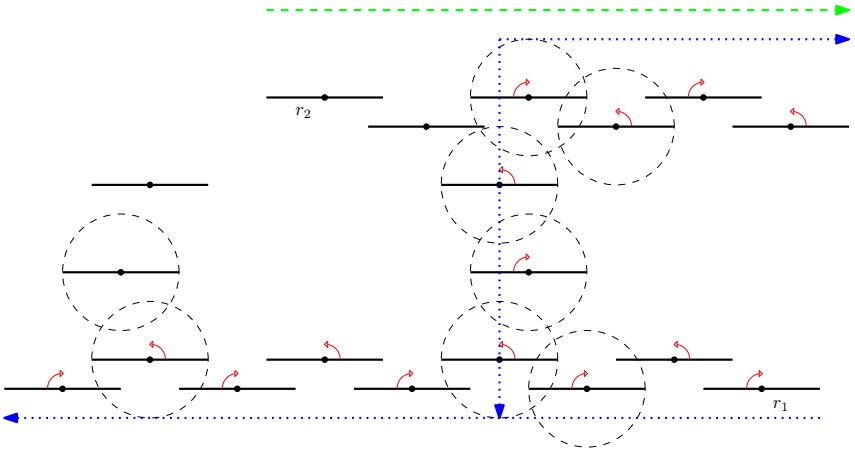


Figure 5 Propagation arrows summarizing the forced rotation orders and directions for different roots (dotted blue for the clockwise rotation of r_1, dashed green for the clockwise rotation of r_2). The red rotation arrows indicate the directions forced by the clockwise rotation of r_1.

A horizontal chain and a vertical chain that share an arm intersect; propagation along one chain may spread to the other through the shared arm. When propagation spreads, we distinguish a main chain (where propagation starts) and a branching chain (where propagation spreads to). All possible local propagation patterns between two intersecting chains fall into eight cases, depending on the geometry and on the chosen root and direction (Figure 4).

Because propagation along a chain has a well-defined direction and order, we represent it by a *propagation arrow* (Figures 4 and 5): arms at the head rotate first and at the tail rotate last. In our constructions chains are horizontal or vertical, so arrows are rectilinear. Propagation can also travel through a sequence of intersecting chains, possibly stopping at intersections depending on the chosen root and direction; this behavior is captured combinatorially in Section 2.3 and realized geometrically in the reduction to SM-RAMP.

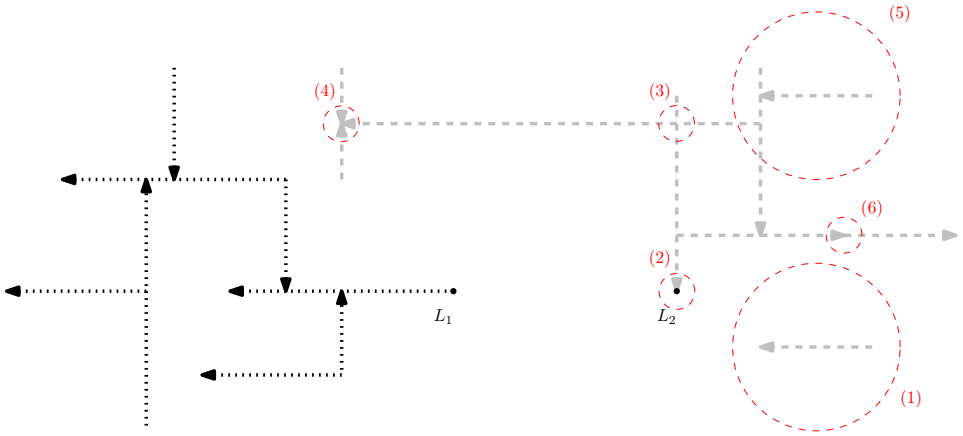


Figure 6 Valid (left) and invalid (right) light ray bundles for two lighthouses. The dashed grey bundle is invalid because it is (1) disconnected; (2) vertically attached to the lighthouse; (3) cyclic or contains a degree-4 vertex; (4) contains a vertex incident to three distinct light rays; (5) contains intersecting light rays whose arrows do not correspond to any local propagation pattern in Figure 4; or (6) contains collinear intersecting light rays.

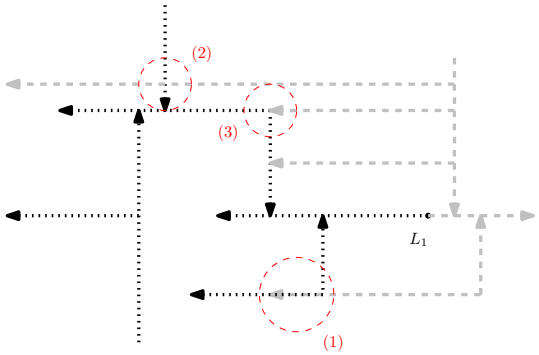


Figure 7 Forbidden interactions between light ray bundles. An interaction is invalid if (1) a light ray overlaps a light ray of the other bundle; (2) a light ray crosses a light ray of the other bundle; or (3) a light ray intersects a collinear light ray of the other bundle.

2.3 The lighthouse propagation problem

We abstract propagation behavior in the Lighthouse Propagation (LPR) problem. We work on an $(m+1) \times (m+1)$ integer grid, where $m \in \mathbb{N}^+$. Each lighthouse L_i sits at a grid point l_i and has two *bundles* X_i, Y_i of directed grid edges. Each bundle is a tree-like rectilinear structure of horizontal and vertical unit-length edges of maximum degree 3, with exactly one horizontal edge incident to the lighthouse; orientations are chosen so that each bundle behaves like a propagation pattern along rectilinear chains. Maximal collinear sequences of edges are grouped into *(light) rays*, each drawn as an arrow (Figure 6). Bundles must also interact in a restricted way: light rays of different bundles never overlap and only intersect orthogonally at endpoints, mirroring how propagation spreads between chains (Figure 7).

Each lighthouse chooses exactly one of its two bundles to illuminate; all edges in the chosen bundle become *light-directed*. A *directed light cycle* is a directed cycle whose edges are all light-directed. A set of illuminated rays *induces* such a cycle if it contains all its edges. The decision problem Lighthouse Propagation asks, given lighthouses $L_1, \ldots, L_n$, where each lighthouse L_i has bundles X_i, Y_i, whether there exists a choice function c with $c(L_i) \in \{X_i, Y_i\}$ such that the illuminated rays do not induce a directed light cycle. The *reflection count* is the maximum number of intersections (reflections) along any light ray.

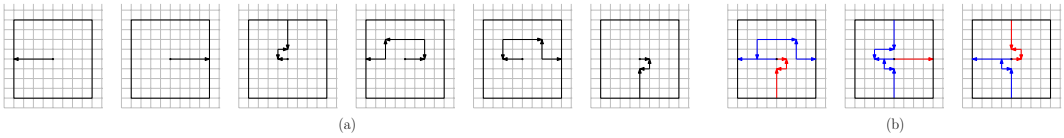


Figure 8 Variable blocks: local ray patterns for different relative positions of incident clauses (a) and variable blocks constructed from them (b).

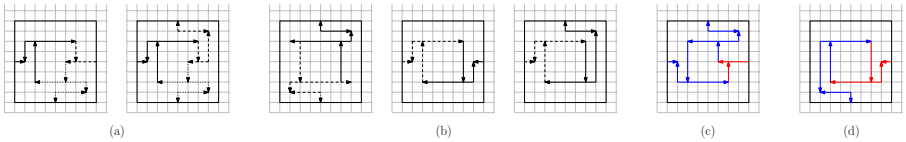


Figure 9 Representative block configurations for 3-clauses (a) and 2-clauses (b). Light rays with the same style belong to the same lighthouse. Configurations (c) and (d) are obtained by mirroring.

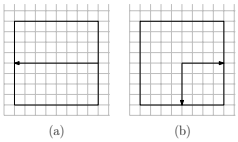


Figure 10 Connector blocks: a straight 1-connector (a) and a bending 2-connector (b).

We show that LPR is NP-complete even when the reflection count is at most 3. The reduction is from Planar 3-Bounded 3-SAT; each lighthouse encodes a variable (choosing X_i vs. Y_i corresponds to setting the variable to `true` vs. `false`), and carefully designed clause gadgets force a directed light cycle whenever a clause is not satisfied.

3 Hardness of the lighthouse propagation problem

3.1 Description of the reduction

We sketch the reduction proving Theorem 2. We begin with the incidence graph $G(\varphi)$ of a planar formula φ, for which a polynomial-size rectilinear embedding is known to exist.

▶ **Lemma 4** (Tamassia [20]). *A connected planar graph on n vertices admits a rectilinear embedding in an $\mathcal{O}(n) \times \mathcal{O}(n)$ grid, computable in polynomial time.*

We compute a planar rectilinear embedding $\mathcal{E}$ of $G(\varphi)$. Each variable–clause incidence is drawn as a rectilinear path whose interior grid points are *connectors* (straight or elbow). A clause is top/bottom/left/right-positioned with respect to a variable if the path initially goes up/down/left/right; we use $\mathsf{T}, \mathsf{B}, \mathsf{L}, \mathsf{R}$ as abbreviations for these four positions.

We scale $\mathcal{E}$ by a factor of 8 to obtain $\mathcal{E}'$ and replace each vertex by a square *block* of size 8×8: a *variable block*, a *clause block*, or a *connector block*. Adjacent vertices in $\mathcal{E}$ become blocks that share a side. Each block has four *outlets* (midpoints of its sides), namely its T-outlet, B-outlet, L-outlet, and R-outlet, through which light rays may enter or exit.

We color-code the light rays to make their bundle assignments explicit. For a lighthouse L_i, a light ray is colored red (resp. blue) if it belongs to X_i (resp. Y_i).

Variable blocks. Each variable x_i is replaced by a variable block with lighthouse L_i at its center and two bundles X_i (for $x_i =$ `true`) and Y_i (for $x_i =$ `false`). Since each variable appears in at most three clauses, we need only a constant number of local configurations

$$\varphi = C_1 \wedge C_2 = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$$

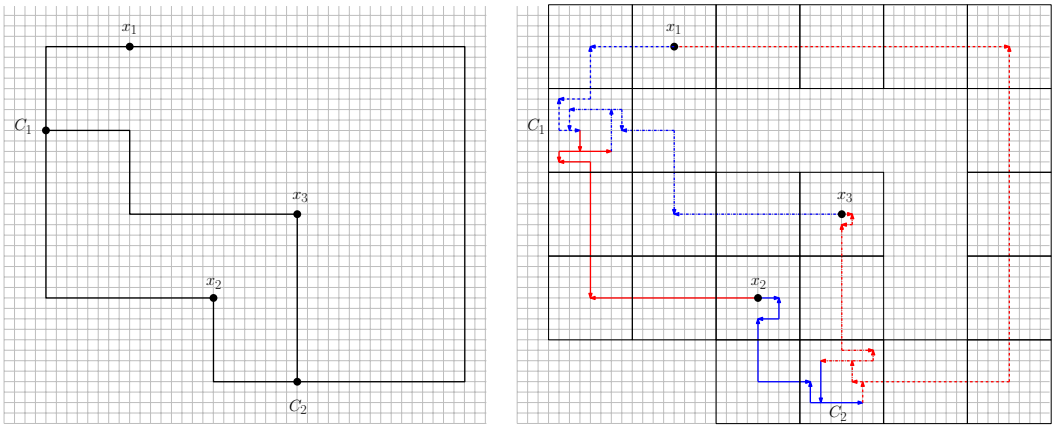


Figure 11 Example of the reduction from Planar 3-Bounded 3-SAT to LPR.

for all possible positions of clauses around x_i (Figure 8). Each occurrence contributes one ray exiting through the appropriate outlet and belonging to X_i or Y_i depending on whether the literal is positive or negative. Variables with fewer occurrences are handled by deleting unused rays or simplifying single occurrences.

▶ **Observation 5.** *For any variable x_i with up to three occurrences and positions $\Pi \subset \{\mathsf{T}, \mathsf{B}, \mathsf{L}, \mathsf{R}\}$, there exist bundles X_i, Y_i such that for each clause C containing x_i at position $t \in \Pi$, the light ray at the t-outlet belongs to X_i if the literal is positive and to Y_i otherwise.*

Clause blocks. Each clause is replaced by a clause block. A 3-literal (resp. 2-literal) clause uses a *3-clause* (resp. *2-clause*) block. Each constituent variable contributes one set of rays entering at the outlet facing its variable block, belonging to X_i or Y_i according to the sign of the literal. Inside the block, these sets are arranged so that:

- if all literals are `false`, all rays are illuminated, and their union has a directed light cycle;
- if at least one literal is `true`, the illuminated rays contain no directed cycle.

All patterns of positions are obtained from a small number of representatives (Figure 9).

▶ **Observation 6.** *Let C be a clause and Π the set of positions of its variables in $\mathcal{E}$. There exist $|\Pi|$ light ray sets $(\mathcal{L}_j)_{1 \le j \le |\Pi|}$, each rooted at a distinct t-outlet ($t \in \Pi$), that satisfy all interaction constraints and whose union contains a directed cycle that uses at least one edge from each $\mathcal{L}_j$. Moreover, if the variable x_i that is t-positioned appears positively (resp. negatively) in C, then the light ray set rooted at the t-outlet is a subset of X_i (resp. Y_i).*

Connector blocks. Each connector is replaced by a connector block. Straight connectors yield *1-connector blocks* with a single light ray; elbows yield *2-connector blocks* with two perpendicular rays forming a bend (Figure 10). Along each variable-to-clause path, connector rays are directed to represent propagation behavior within robotic arm chains.

Finally, for each variable x_i we build X_i and Y_i by combining (i) the rays inside the variable block (Observation 5), (ii) appropriate connector rays along each path, and (iii) the corresponding clause-block rays (Observation 6). This yields an instance of LPR (Figure 11).

3.2 Validity of the construction

We briefly justify correctness; full details can be found in the full version of the paper.

Although each block individually satisfies the bundle requirements of a valid LPR instance (Figures 6 and 7), we must first verify that these properties are preserved when the blocks are embedded in the plane.

▶ **Lemma 7.** *The construction in Section 3.1 yields a valid instance of LPR having reflection count at most* 3*.*

Next, to prove correctness, we examine the implications of choosing which bundle to illuminate for the light rays within clause blocks (Observation 8); should the illuminated light rays induce a directed light cycle, the blocks are specifically designed to force the directed light cycle to be contained within a clause block (Lemma 9).

▶ **Observation 8.** *For any variable x_i and clause C, rays associated with a positive (resp. negative) literal of x_i in C are illuminated iff the lighthouse for x_i illuminates X_i (resp. Y_i). Moreover, no directed path joins two distinct outlets in any variable or clause block.*

▶ **Lemma 9.** *If the illuminated rays induce a directed light cycle, then all edges of this cycle lie within a single clause block.*

Proof sketch. No variable or connector block contains a directed cycle. A cycle spanning multiple blocks would imply a directed path between two outlets of a variable or clause block, since entering and exiting a block via distinct outlets would require a directed path inside that block, contradicting Observation 8. ◀

Armed with the previous two results, if the illuminated bundles do not induce a directed light cycle, we assign each variable x_i the value `true` (resp. `false`) if X_i (resp. Y_i) is illuminated, and this assignment must satisfy every clause, since any unsatisfied clause would induce a forbidden directed light cycle by Observation 8. Conversely, given a satisfying assignment for φ, illuminating X_i if x_i is `true` and Y_i if x_i is `false` cannot create a directed light cycle; otherwise, if a clause block's illuminated light rays contain such a cycle (Lemma 9), the associated clause is unsatisfied by Observation 8.

Thus, a directed light cycle exists if and only if some clause is unsatisfied, as needed to prove the following:

▶ **Theorem 2.** *Lighthouse Propagation is* NP*-complete, even when restricted to instances with a reflection count of at most three.*

4 Hardness of the SM-RAMP problem

The construction mirrors that of Section 3. The key idea is to replace the light rays of an LPR instance by chains of robotic arms, treating light rays as propagation arrows, while carefully encoding the binary choice at each lighthouse.

4.1 Description of the reduction

We start from an LPR instance of Section 3.1. After a slight local modification to the block structure, we replace each unit segment of each light ray by a small *cell*, containing a constant-size arrangement of robotic arms, thereby forming an H–V SM-RAMP instance. We define the following cells (Figures 12, 13, 14 and 15):

- *Lighthouse cells*, which encode the choice between two bundles of a lighthouse by two interacting horizontal chains whose special ends, r_i^t and r_i^f, represent truth values.
- *Rectilinear cells*, replacing straight unit segments of rays by horizontal or vertical chains.
- *L cells*, replacing a bend between two rays by interacting horizontal and vertical chains.
- *T cells*, generalizing L cells to replace a degree-3 junction of rays.

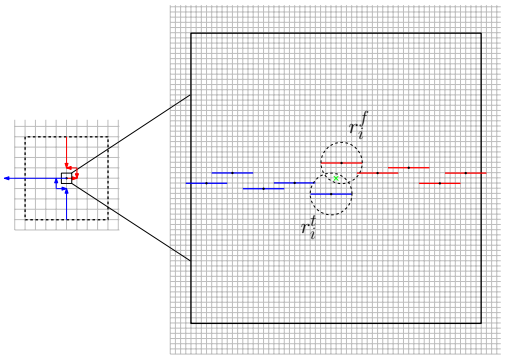


Figure 12 Robotic arm lighthouse cell for replacing a lighthouse L_i, with green cross at center.

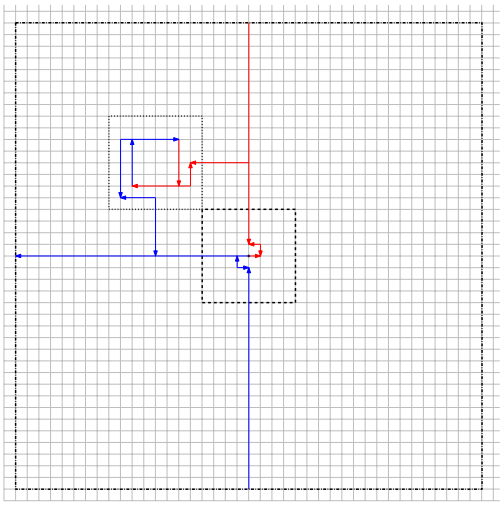


Figure 13 Augmented variable block (dash-dotted) with original variable block (dashed) and dummy 2-clause block (dotted).

To ensure that at each lighthouse exactly one of the two bundles behaves as "chosen", we augment the variable gadgets with a dummy clause enforcing that one of r_i^t, r_i^f must rotate clockwise and the other anticlockwise. We enlarge each light-ray block to an *augmented block* of size 40×40, nest the original 8×8 gadget in the center, and attach a dummy 2-clause block representing $(x \vee \overline{x})$ inside the same augmented variable block (Figure 13). The augmented construction still yields an NP-hardness LPR reduction, now from Planar 4-Bounded 3-SAT (original instance plus tautological clauses $(x_i \vee \overline{x_i})$).

Finally, each unit-length square of the light-ray embedding is replaced by one of the four robotic arm cells (lighthouse, rectilinear, L, T), according to the local ray pattern (Figures 14 and 15). The midpoints of robotic arms are aligned on the grid so that chains connect across block boundaries, producing an H–V SM-RAMP instance with polynomially many arms.

4.2 Implications of the construction

Two observations about local structure can be made about the constructed SM-RAMP instances: each robotic arm interacts with only a few others, and the arm cells are designed so that propagation of forced directions along chains mirrors the direction of light rays (Observation 10).

▶ **Observation 10.** *In the constructed SM-RAMP instance, each robotic arm interacts with at most three others, and the propagation induced by choosing a rotation direction for r_i^t or r_i^f matches the propagation arrows of the corresponding bundle in the LPR instance.*

The second part of Observation 10 ensures that exactly one truth-value arm rotates clockwise in each lighthouse cell, which is crucial for the correctness of the reduction:

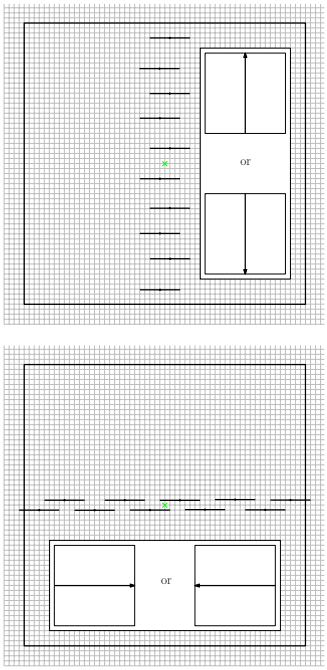


Figure 14 Rectilinear cells: a vertical chain (top) and a horizontal chain (bottom).

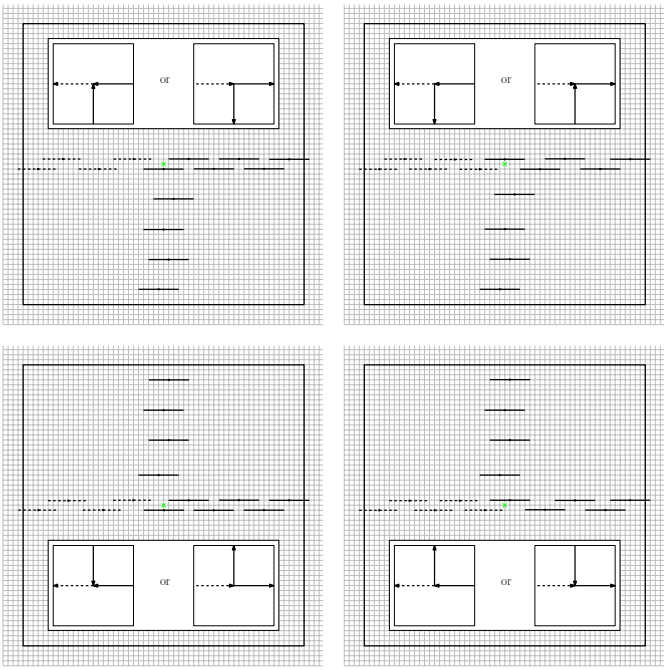


Figure 15 Base L cells (solid arms) and corresponding T cells (solid + dashed arms), replacing bends and degree-3 junctions of rays, respectively.

▶ **Observation 11.** *In any solution of the constructed SM-RAMP instance, for every lighthouse L_i exactly one of r_i^t and r_i^f rotates clockwise.*

Proof sketch. If both truth-value arms of a variable x_i rotate clockwise, there would exist a cyclic rotation dependency among the robotic arms replacing x_i's 2-clause dummy block. ◀

We now relate satisfying assignments and rotation orderings. A cyclic rotation dependency analogous to that in the proof of Observation 11 arises within any clause block, should propagation happen according to the rays, which we now interpret as propagation arrows (Observation 10). Breaking the cycle requires rotating a subset of the chains in an order opposite to that of the propagation arrow, which is possible only when the truth-value arm at the end of the chain is rotated anticlockwise. Which truth-value arm to rotate anticlockwise in each lighthouse cell can be derived from which bundle is being illuminated (Figure 16).

▶ **Lemma 12.** *If the LPR instance $\mathcal{I}$ is a yes-instance, then the corresponding H–V SM-RAMP instance $\mathcal{I}'$ is also a yes-instance.*

Conversely, suppose that the constructed H–V SM-RAMP instance $\mathcal{I}'$ admits a rotation ordering, and fix any clause C. If the rotations within the robotic arms replacing C's clause block were forced to follow the propagation arrows, then, by construction, they would induce a cyclic rotation dependency. Hence some subset of these arms must rotate in an order that does not adhere to the propagation pattern. These arms are in a chain replacing a light ray in a lighthouse's bundle, and that lighthouse would illuminate its other bundle.

▶ **Lemma 13.** *If the H–V SM-RAMP instance $\mathcal{I}'$ is a yes-instance, then the LPR instance $\mathcal{I}$ is also a yes-instance.*

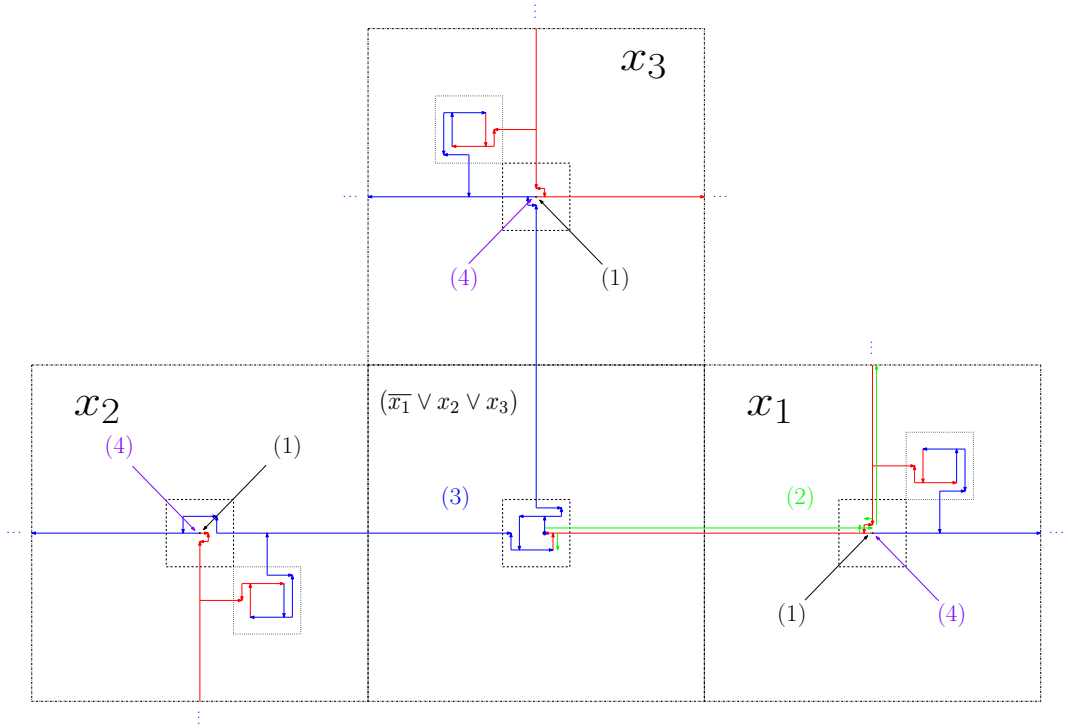


Figure 16 Example of the four rotation phases (numbered) within a clause gadget and its incident variables. The two chains replacing the red light rays in the clause block rotate in the order inverse to the light rays' arrows, starting with the anticlockwise rotation of r_1^f.

Since collision checking is polynomial, SM-RAMP is in NP; SM-RAMP being NP-complete follows from Lemmas 12 and 13.

▶ **Theorem 1.** *SM-RAMP is* NP*-complete, even when restricted to H–V instances where every robotic arm interacts with at most three others.*

5 Positive results

Before presenting our algorithmic results, we briefly recall the framework of parameterized complexity, which provides a natural formalism for capturing tractability when certain aspects of an instance are bounded. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where an instance consists of the main input x together with an integer κ, the *parameter*. The problem is *fixed–parameter tractable* (FPT) if membership in L can be decided in time $f(\kappa) \cdot |x|^{O(1)}$ for some computable function f. We refer the reader to [8–10] for more on the theory.

Our positive results rely on a strong geometric phenomenon that emerges when robotic arms are restricted to the two allowed orientations $\{0, \pi/2\}$ (horizontal and vertical); the possible interactions among arms are extremely local, in a sense that can be described precisely using the flower representation (Figure 17).

Consider any instance (single-move or multi-move) in which each arm may occupy only the horizontal or vertical orientation. For such an arm r, the regions swept by a clockwise or anticlockwise rotation form two fixed 90° sectors around the unit square centered at r. Together, these sectors partition the region adjacent to the square into eight quarter-circular petals, collectively called the flower of r.

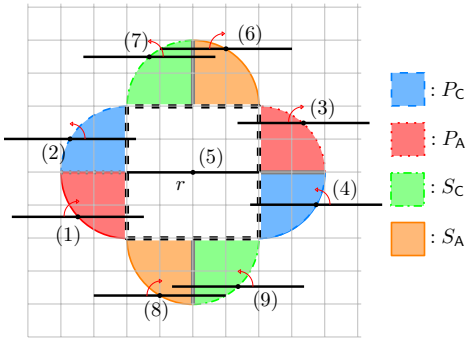


Figure 17 The flower of a robotic arm r, consisting of eight quarter-circular petals surrounding the unit square centered at r. The petal in which a second arm r' lies determines exactly one constraint between r and r' among P_C, S_C, P_A, S_A.

A fundamental property of this decomposition is that the location of the midpoint of another arm r' relative to these petals determines *exactly one* of the possible rotation constraints between r and r'. Specifically, each petal of r corresponds to exactly one of the four constraint graphs P_C, S_C, P_A, S_A, depending on whether the interaction concerns a clockwise or anticlockwise rotation of one arm (P-constraints) or the effect of having already rotated the other arm (S-constraints). Thus the flower captures the entire interaction pattern between r and the rest of the configuration. If the midpoint of r' lies inside the middle square of r's flower, then r and r' cannot rotate past one another in *any* sequence of allowed moves. This motivates the following definition.

▶ **Definition 14.** *A robotic arm r is **frozen** if no collision-free sequence of rotations, of any ordering and any choice of clockwise or anticlockwise directions, allows r to rotate.*

Frozen arms occur either because another arm's midpoint lies in their middle square or because they participate in a cycle of rotation dependencies. Such arms may be ignored in constructive algorithms, as they can never participate in any feasible reconfiguration. Flowers allow us to express a key locality property that underlies our parameterized algorithms.

▶ **Observation 15.** *For any instance where all arms are restricted to horizontal and vertical orientations, in the union of the four constraint graphs P_C, S_C, P_A, S_A induced by R_{nf}, the set of non-frozen arms, each arm $r \in R_{nf}$ has at most one neighbour in each petal of its flower. Consequently, r has degree at most eight in this induced union graph.*

In particular, each non-frozen arm interacts with at most eight other non-frozen arms, so the constraint graphs, when restricted to R_{nf}, have constant maximum degree. This strong locality is the structural cornerstone of our fixed-parameter tractability results.

5.1 The number of robotic arms to be made vertical as a parameter

We now study the parameterized complexity of MAX-SM-RAMP in the two-orientation setting (horizontal or vertical). In MAX-SM-RAMP, the goal is not necessarily to rotate all horizontal arms to the vertical orientation, but rather to decide whether at least k arms can be made vertical by a collision-free sequence of clockwise or anticlockwise rotations.

▶ **Definition 16.** *For a conflict-free configuration of robotic arms $\mathcal{R}$, each either horizontal or vertical, an arm r currently in the horizontal orientation is*

- ***clockwise-free** (C-free) in $\mathcal{R}$ if it is possible to rotate r clockwise to the vertical orientation without collisions, yielding a conflict-free configuration; and*
- ***anticlockwise-free** (A-free) in $\mathcal{R}$ if the property holds for an anticlockwise rotation.*

An arm may be C-free, A-free, both, or neither; directional freedom is determined purely by whether the relevant sweep sector is unobstructed. To reason about interactions among directionally free arms, we use directional interaction graphs.

▶ **Definition 17.** *For a configuration $\mathcal{R}$ and a direction* $\mathsf{dir} \in \{\mathsf{C}, \mathsf{A}\}$*, the graph $G_{\mathsf{dir}}(\mathcal{R})$ has one vertex for each* dir*-free arm in $\mathcal{R}$, and an edge $\{r, r'\}$ whenever rotating r in direction* dir *causes the swept sector of r to intersect the current position of r' (or vice versa).*

A key geometric fact proved earlier is that the flower of any arm contains at most one non-frozen neighbor in each petal. Because frozen arms are never directionally free, this gives an immediate degree bound on the directional interaction graphs.

▶ **Observation 18.** *Let $\mathcal{R}$ be any configuration and* $\mathsf{dir} \in \{\mathsf{C}, \mathsf{A}\}$*. Then every arm r that is* dir*-free in $\mathcal{R}$ has at most eight neighbors in $G_{\mathsf{dir}}(\mathcal{R})$.*

Let $V_{\mathsf{dir}}(\mathcal{R})$ denote the set of dir-free arms. The bounded degree of $G_{\mathsf{dir}}(\mathcal{R})$ implies:

▶ **Lemma 19.** *If $|V_{\mathsf{dir}}(\mathcal{R})| \geq 9k$, then we can compute in polynomial time a set S of k arms that are all* dir*-free in $\mathcal{R}$ and pairwise non-adjacent in $G_{\mathsf{dir}}(\mathcal{R})$.*

Such an independent set sweeps disjoint sectors, so its arms can be safely rotated (in any order) in direction dir.

▶ **Theorem 20.** *MAX-SM-RAMP parameterized by k is fixed-parameter tractable. Specifically, it can be solved in time $(36k)^k \cdot \mathrm{poly}(n)$.*

We can adapt this FPT algorithm into a polynomial kernel using geometric arguments.

▶ **Theorem 21.** *MAX-SM-RAMP parameterized by k admit a kernel on $\mathcal{O}(k^3)$ arms.*

5.2 The number of allowed rotations as a parameter

We now study 2A-MM-RAMP under a bound ℓ on the total number of allowed rotations, when each arm is restricted to the horizontal or vertical orientation. Unlike in SM-RAMP, where each arm may be rotated at most once and be made vertical, the 2A-MM-RAMP model allows each arm to visit any of its allowed angles, possibly moving away from its target and later returning to it. Thus, short valid sequences may temporarily rotate arms that already match their target angles.

Recall that an arm is frozen if no collision-free sequence of rotations ever allows it to change its angle. If there exists an arm r_i such that $\alpha_i \neq \beta_i$ and r_i is frozen, then the instance is immediately a no-instance. We henceforth restrict attention to instances in which this does not happen. We now define a static interaction graph on non-frozen arms that captures all potential geometric conflicts.

Let R be the set of all arms and let $R_{\mathrm{nf}} \subseteq R$ denote the set of non-frozen arms in the instance. For each unordered pair of distinct arms $\{r, r'\} \subseteq R_{\mathrm{nf}}$, the flower construction and two-orientation model determine whether r and r' are in conflict in one of the four constraint graphs $P_{\mathsf{C}}, S_{\mathsf{C}}, P_{\mathsf{A}}, S_{\mathsf{A}}$ (that is, whether the midpoint of r' lies in a petal of r, or vice versa, that induces a constraint when rotating in some direction). We define the *union interaction graph* as the undirected graph $H = (V(H), E(H))$ with $V(H) = R_{\mathrm{nf}}$ and where an edge $\{r, r'\}$ belongs to $E(H)$ if and only if r and r' are adjacent in at least one of $P_{\mathsf{C}}, S_{\mathsf{C}}, P_{\mathsf{A}}, S_{\mathsf{A}}$ (restricted to R_{nf}). In other words, $E(H) = E\big(P_{\mathsf{C}}[R_{\mathrm{nf}}]\big) \cup E\big(S_{\mathsf{C}}[R_{\mathrm{nf}}]\big) \cup E\big(P_{\mathsf{A}}[R_{\mathrm{nf}}]\big) \cup E\big(S_{\mathsf{A}}[R_{\mathrm{nf}}]\big)$. Thus $\{r, r'\} \in E(H)$ precisely when some clockwise or anticlockwise rotation of r or r'

(between the two allowed angles) can cause a collision between them. The graph H is entirely determined by the positions of the arm midpoints and the two-angle model; it does not depend on the particular configuration (source, intermediate, or target).

By Observation 15, every non-frozen arm r participates in at most eight constraints in total, one per petal of its flower, and each such constraint appears in exactly one of $P_{\mathsf{C}}, S_{\mathsf{C}}, P_{\mathsf{A}}, S_{\mathsf{A}}$. Restricting to R_{nf} can only remove neighbours, and therefore:

▶ **Observation 22.** *The union interaction graph H has maximum degree at most* 8.

Let $D = \{ i \in R_{\mathrm{nf}} : \alpha_i \neq \beta_i \}$ be the set of (necessarily non-frozen) arms whose initial and target orientations differ. Every arm in D must be rotated at least once in any valid solution. Hence, if $|D| > \ell$, the instance is immediately a no-instance. We therefore assume from now on that $|D| \leq \ell$. For a set $X \subseteq V(H)$ and an integer $r \geq 0$, define the ball $B_r(X) = \{ v \in V(H) : \mathrm{dist}_H(v, X) \leq r \}$, where $\mathrm{dist}_H(v, X)$ denotes the shortest graph distance between v and a vertex in X.

▶ **Lemma 23.** *If $|D| \leq \ell$, then $|B_\ell(D)| \leq |D|\,(1 + 8 + 8^2 + \cdots + 8^\ell) \leq \ell\,\frac{8^{\ell+1}-1}{7}$.*

▶ **Lemma 24.** *Assume a valid sequence of length at most ℓ transforming $\mathcal{R}_{\mathrm{s}}$ to $\mathcal{R}_{\mathrm{t}}$ exists, and let R_{rot} be the set of arms that are actually rotated at least once in this sequence. Then every arm in R_{rot} belongs to $B_\ell(D)$.*

Combining Lemmas 23 and 24, we see that the set $R' = B_\ell(D)$ contains every arm that is ever rotated in any valid solution of length at most ℓ. Thus all arms outside R' remain fixed obstacles; the search for an ℓ-bounded solution can be restricted to R'.

▶ **Theorem 25.** *Let $\ell \in \mathbb{N}$. Given an instance of 2A-MM-RAMP in which each arm has at most two allowed orientations (horizontal and vertical), one can decide whether there exists a valid sequence of at most ℓ rotations transforming $\mathcal{R}_{\mathrm{s}}$ into $\mathcal{R}_{\mathrm{t}}$ in time $\left(\ell \cdot 8^{\ell+1}\right)^{\ell} \cdot \mathrm{poly}(n)$.*

References

1 Nicolas Bousquet, Remy El Sabeh, Amer E. Mouawad, and Naomi Nishimura. On the complexity of constrained reconfiguration and motion planning. In *51st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2026)*, pages 1–15. Springer, 2026. doi:10.1007/978-3-032-17801-5_1.

2 Nicolas Bousquet, Felix Hommelsheim, Yusuke Kobayashi, Moritz Mühlenthaler, and Akira Suzuki. Feedback vertex set reconfiguration in planar graphs. *Theoretical Computer Science*, 979:114188, 2023. doi:10.1016/J.TCS.2023.114188.

3 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of reconfiguration problems. *Computer Science Review*, 53:100663, 2024. doi:10.1016/J.COSREV.2024.100663.

4 Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1-2):41–68, 2004. doi:10.1142/S0218195904001354.

5 Marcin Brianski, Stefan Felsner, Jedrzej Hodor, and Piotr Micek. Reconfiguring independent sets on interval graphs. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *LIPIcs*, pages 23:1–23:14, 2021. doi:10.4230/LIPIcs.MFCS.2021.23.

6 Kevin Buchin, Mart Hagedoorn, Irina Kostitsyna, and Max van Mulken. Dots & boxes is PSPACE-complete. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *LIPIcs*, pages 25:1–25:18, 2021. doi:10.4230/LIPIcs.MFCS.2021.25.

7 Petar Ćurković and Bojan Jerbić. Dual-Arm robot motion planning based on cooperative coevolution. In *DoCEIS*, volume 314 of *IFIP Advances in Information and Communication Technology*, pages 169–178. Springer, 2010. doi:10.1007/978-3-642-11628-5_18.

8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

10 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

11 Robert A. Hearn. *Games, puzzles, and computation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006. URL: https://hdl.handle.net/1721.1/37913.

12 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/J.TCS.2005.05.008.

13 Luu Thi Hue and Nguyen Pham Thuc Anh. Planning the optimal trajectory for a dual-arm robot system using a genetic algorithm considering the controller. In *2022 International Conference on Advanced Technologies for Communications (ATC)*, pages 292–297, 2022. doi:10.1109/ATC55345.2022.9942975.

14 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. doi:10.1016/J.TCS.2010.12.005.

15 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In *9th International Symposium on Parameterized and Exact Computation (IPEC 2014)*, pages 246–257, 2014. doi:10.1007/978-3-319-13524-3_21.

16 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/A11040052.

17 Naoto Ohsaka. Gap amplification for reconfiguration problems. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1345–1366, 2024. doi:10.1137/1.9781611977912.54.

18 Naoto Ohsaka. Gap preserving reductions between reconfiguration problems. *Journal of Computer and System Sciences*, 157:103733, 2026. doi:10.1016/J.JCSS.2025.103733.

19 Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for coordinated multi-arm motion planning. *International Journal of Robotics Research*, 37(10), 2018. doi:10.1177/0278364918765952.

20 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.

21 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

Improved Approximation of Two Watchmen's Routes in Simple Polygons

Anna Brötzner
Faculty of Technology and Society, Malmö University, Sweden

Bengt J. Nilsson
Faculty of Technology and Society, Malmö University, Sweden

Christiane Schmidt
Department of Science and Technology, Linköping University, Sweden

Abstract

We study the watchman route problem for a set of two watchmen for the objective of minimizing the length of the longest route (min-max) inside a simple polygon P having n vertices, which is known to be weakly NP-hard. First, we consider seeing a discrete set of m points in the interior of P. We show that even this problem is weakly NP-hard and present an approximation algorithm with approximation ratio $2+\varepsilon$ that runs in $O(m^5 n)$ time, assuming that a starting point for each of the two routes is given. We generalize the algorithm to see all of the interior of P in $O(n^6)$ time with approximation ratio $2+\pi/2 \approx 3.571$, improving on the previously known best algorithm that has an approximation ratio of ≈ 6.922 and runtime $O(n^2)$ [8]. Finally, we describe how to extend this algorithm to the case where no starting points are given, this taking $O(n^8)$ time, yielding an approximation ratio of $3+\pi/2 \approx 4.571$, improving on the previously known best approximation algorithm with ratio ≈ 5.969 also having runtime $O(n^8)$ [8].

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases Art gallery problem, watchman route problem, multiple watchmen, path planning, polygons

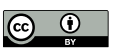
Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.11

Funding All authors were supported by grant 2021-03810 (Illuminate: provably good algorithms for guarding problems) from the Swedish Research Council (Vetenskapsrådet). Brötzner and Nilsson were also funded by grant 20250562 (Illuminate and Move: provably good algorithms for exploration problems) from the Crafoord foundation.

1 Introduction

The problem of finding a shortest watchman route goes back to Chin and Ntafos, who also provided algorithms for rectilinear polygons [2]. Algorithms for simple polygons were first given in [3] and corrected in [10]. The shortest watchman route problem has two variants, the *anchored* or *fixed* variant, where the route must pass through a given point sometimes specified to lie on the boundary, and the *non-anchored* or *floating* variant with no requirement to pass a specific point. The currently best algorithm for the anchored version requires $O(n^3)$ time in the worst case [11], if the anchor point is assumed to lie on the boundary of the polygon. The floating variant can be computed in $O(n^4)$ time by combining the result in [11] with the method in [9].

For two watchmen, the so-called *two-watchman route* problem, Mitchell and Wynters [7] showed that computing routes with the objective of minimizing the length of the longer route (min-max) is weakly NP-hard. Nilsson and Packer [8] recently provided approximation


© Anna Brötzner, Bengt J. Nilsson, and Christiane Schmidt;
licensed under Creative Commons License CC-BY 4.0

algorithms for computing min-max two-watchman routes with an approximation ratio of ≈ 5.969 for the floating case taking $O(n^8)$ time, and ≈ 6.922 for the case of two given starting points, not necessarily on the boundary, taking $O(n^2)$ time.

We first consider the following variant of the two-watchman route problem. We are given m point sized treasures (the set $\mathcal{S}$) in the simple polygon that need to be guarded [1, 4]. For the single watchman route case, an appropriate modification of the algorithm of Tan and Jiang [11] obtains a shortest route that visits the visibility polygons of the treasures in $O(m^2n)$ time for the boundary anchored version and can be extended to $O(m^3n)$ time for the floating version using Tan's technique [9]. We want to compute the shortest two routes V_s and $V_{s'}$, starting at two points s and s', respectively, not necessarily on the boundary of the polygon, such that every point $p_i \in \mathcal{S}$ is seen from at least one point on one of the routes. Two natural optimization criteria have previously been considered for these problems: the *min-max* measure and the *min-sum* measure. For the *min-max* measure we wish to obtain two routes such that the longer of the two is as short as possible. As we prove, this problem is weakly NP-hard not only for the objective of seeing all of the simple polygons [7], but also for the objective of seeing all points in $\mathcal{S}$. The *min-sum* measure on the other hand asks for two routes such that the sum of their lengths is minimized. The complexity status for this version is unknown.

We develop two algorithms for this anchored discrete problem, optimizing on the min-max criterion, one having approximation ratio $2 + \pi/2$ and running in $O(m^5n)$ time and an improved version that, for any fixed $\varepsilon > 0$, gives an approximation ratio of $2+\varepsilon$, also running in $O(m^5n)$ time.

Secondly, we also consider the problem of guarding the complete polygon P and modify the first algorithm developed for a discrete point set using results from Nilsson and Packer [8]. We obtain algorithms having approximation ratio $2 + \pi/2$ for the anchored and $3 + \pi/2$ for the floating versions of the two-watchman route problem. The algorithm for the anchored version runs in $O(n^6)$ time and the algorithm for the floating version runs in $O(n^8)$ time.

2 Preliminaries

Let P be a simple polygon with n vertices $\mathcal{V} = \{v_1, \ldots, v_n\}$ labeled in clockwise order, and let $\mathcal{S} = \{p_1, \ldots, p_m\}$ be a set of m points in P, the *treasures*. We assume that no three points among the polygon vertices and $\mathcal{S}$ are collinear. Furthermore, let s and s' be the two starting points for the watchman routes in P, not necessarily lying on the boundary.

For a point $p \in P$, let $VP(p)$ be the *visibility polygon* of p, the set of points in P that are visible to p. An edge of $VP(p)$ that is not contained in an edge of P is called a *window* of p.

Without loss of generality, we may assume that $s, s' \notin VP(p_i)$, for any $p_i \in \mathcal{S}$, as those points are already seen from the starting points. Let C_s^i be the window that separates the subpolygon of $P \setminus VP(p_i)$ containing s from $VP(p_i)$. Analogously, let $C_{s'}^i$ be the window that separates the subpolygon of $P \setminus VP(p_i)$ containing s' from $VP(p_i)$. Now, observe that a watchman who starts walking at s (s') sees point p_i as soon as it visits C_s^i ($C_{s'}^i$). We assign a direction to each window C_s^i ($C_{s'}^i$) so that $VP(p_i)$ lies locally to the left of C_s^i ($C_{s'}^i$) if the polygon is rotated so that the window is directed upwards, see also Figure 1. We denote the lower endpoint of the window the *bottom point* of the window and the upper endpoint the *top point*.

We study the problem of computing two routes such that each of the treasures can be seen from at least one of the routes. We denote the length of a route W by $\|W\|$.

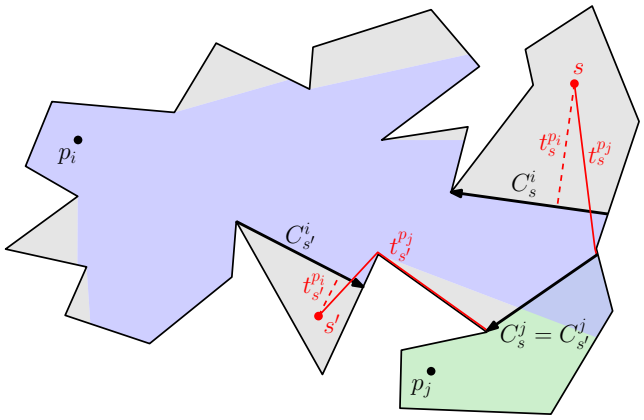


Figure 1 Illustration of the visibility polygon of a point and the windows (black arrows). The windows point towards their corresponding top points. For point p_j, $C_s^j = C_{s'}^j$, hence $p_j \in \mathcal{S}_1$, while for p_i, $C_s^i \neq C_{s'}^i$, thus $p_i \in \mathcal{S}_2$.

▶ **Problem 1** (Min-Max Two-Watchman Route Problem for Treasures). *Given a simple polygon P, a set of points $\mathcal{S}$ in P, (and two starting points s and $s' \in P$,) compute two routes W and W' (where W passes through s and W' passes through s'), such that every point of $\mathcal{S}$ is visible from at least one point on W or W' and such that $\max\{\|W\|, \|W'\|\}$ is minimized.*

▶ **Theorem 1.** *Problem 1 is weakly NP-hard.*

Proof. The reduction is from 2-partition [6]. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of integers. We construct a polygon P with points $p_1, \ldots, p_n$ where an optimal watchman route that sees all points has length $\sum_{i=1}^{n} a_i + \delta$ for some small constant δ, and an optimal solution to Problem 1 consists of two tours with length $1/2 \cdot \sum_{i=1}^{n} a_i + \delta$ each, corresponding to an optimal partition of A.

Let $\delta \ll \min_{a_i \in A} a_i$. We start with a regular convex polygon with n vertices and perimeter δ, which we call the *center* of P. The starting points s and s' are placed on the boundary of the center. On the i^{th} side of the center, we attach a rectilinear tube of width δ/n and length $a_i/2$. At the end of each corridor, there is a small notch that contains point p_i such that the window of p_i has distance $a_i/2$ to the center. See Figure 2 for an illustration of the polygon.

The shortest watchman route that sees all points $p_1, \ldots, p_n$ runs along the boundary of the center, and thus passes through both s and s', and walks through each tunnel up to the window C_s^i of point p_i, for $1 \leq i \leq n$. For two watchman routes, only one of them needs to visit the window of a point p_i. As each route is connected and starts at its corresponding starting point, the watchmen also walk a portion within the center of the polygon. However, note that even walking along all of the boundary of the center extends the total route length only by an additive constant of δ, which does not affect the choice of points assigned to each route. Thus, for an optimal pair of routes T_s, $T_{s'}$ for Problem 1, the points $p_1, \ldots, p_n$ are partitioned into two sets $\mathcal{X}_s$ and $\mathcal{X}_{s'}$, such that T_s visits the windows of the points in $\mathcal{X}_s$ and $T_{s'}$ visits the windows of the points in $\mathcal{X}_{s'}$. Hence, $(\mathcal{X}_s, \mathcal{X}_{s'})$ corresponds to a 2-partition of A. ◀

We can extend this problem and require the whole polygon to be seen.

▶ **Problem 2** (Min-Max Two-Watchman Route Problem for a Polygon). *Given a simple polygon P (and two starting points s and $s' \in P$), compute two routes W and W' (where W passes through s and W' passes through s'), such that every point in P is visible from at least one point on W or W' and such that $\max\{\|W\|, \|W'\|\}$ is minimized.*

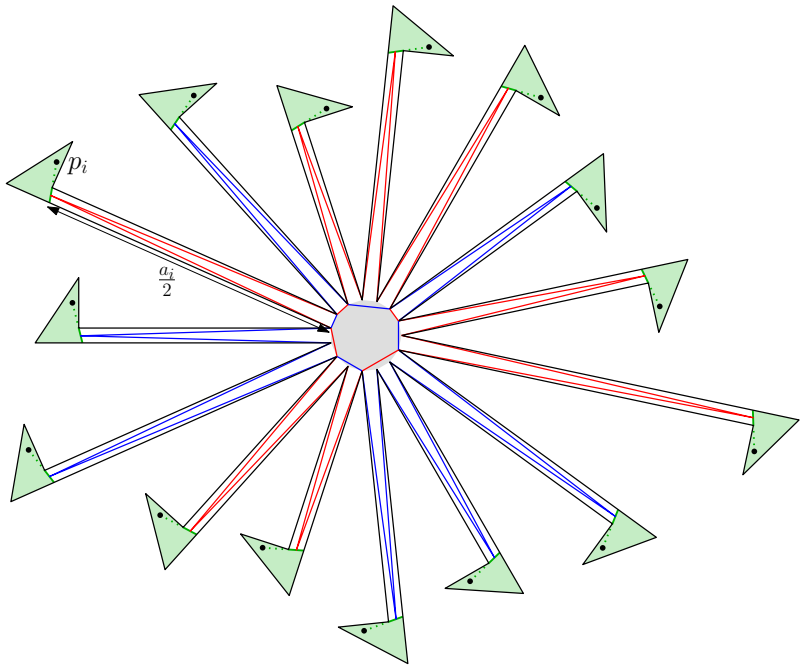


Figure 2 Schematic representation of the polygon constructed from an instance of 2-partition. Each of the two watchmen (red and blue) guards a portion of the points p_i, such that the route lengths of both watchmen are (almost) equal.

Let W_s ($W_{s'}$) be the shortest route that sees all the points in $\mathcal{S}$ and passes through s (s'). If s (s') lies on the boundary of P, we can use the modified algorithm by Tan and Jiang to compute it in $O(m^2n)$ time. If s (s') does not lie on the boundary, we need to modify the algorithm to obtain which window is visited first along a clockwise traversal of the tour starting at s (s'). By trying all possible windows, we can compute the optimal route in $O(m^3n)$ time.

We denote the min-max shortest pair of routes that together see all the points in $\mathcal{S}$, one passing through s and the other passing through s', by T_s and $T_{s'}$. We can without loss of generality assume that each of T_s and $T_{s'}$ is as short as possible. Furthermore, we define $OPT \stackrel{\text{def}}{=} \max\{\|T_s\|, \|T_{s'}\|\}$.

For each $p_i \in \mathcal{S}$, let $t_s^{p_i}$ be the shortest path from starting point s to $VP(p_i)$, i.e., to the window C_s^i, and analogously let $t_{s'}^{p_i}$ be the shortest path from starting point s' to $VP(p_i)$, i.e., to the window $C_{s'}^i$; these paths are marked in red in Figure 1. Let $t_{\min}^{p_i} = \arg\min\{\|t_s^{p_i}\|, \|t_{s'}^{p_i}\|\}$ be the shorter of the two paths to see p_i and let $t_{\max}^{p_i} = \arg\max\{\|t_s^{p_i}\|, \|t_{s'}^{p_i}\|\}$ be the longer of these two paths.

We remark that finding an (approximate) min-max optimal pair of routes is a question of correctly partitioning the points to be seen between the route passing through s and the route passing through s'. Once the partitioning is obtained, the routes can be computed in $O(m^3n)$ time as previously described.

3 The Algorithm

We describe an algorithm to compute two watchman routes to see a set of treasures $\mathcal{S}$ in a polygon P (Problem 1). This is also the basis for the algorithm for seeing the whole polygon (Problem 2) that we present in Section 4, and then generalize to the floating version in Section 5.

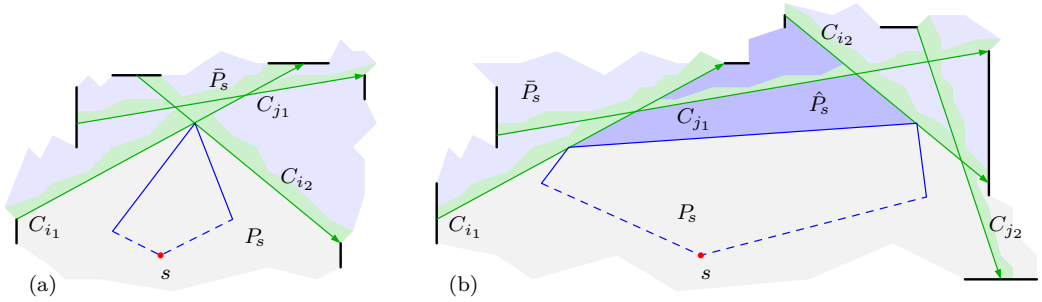


Figure 3 Illustrating the proof of Lemma 2.

First, compute a shortest watchman route W_s that sees all points in $\mathcal{S}$ and starts at s, and a shortest watchman route $W_{s'}$ that sees all points in $\mathcal{S}$ and starts at s'. If the two routes T_s and $T_{s'}$ of an optimal two-watchman route intersect, then either of W_s and $W_{s'}$, together with a route of length 0 at the respective other starting point, gives a 2-approximation. This follows from the fact that we can trace the subpaths of the routes between their intersection points to obtain a route that sees all the points in $\mathcal{S}$. Thus, $\max\{\|W_s\|, \|W_{s'}\|\} \leq \|T_s\| + \|T_{s'}\| \leq 2 \cdot \max\{\|T_s\|, \|T_{s'}\|\}$. All the remaining steps use the assumption that the routes T_s and $T_{s'}$ do not intersect.

We partition the points in $\mathcal{S}$ into two sets, depending on where s and s' lie with respect to $VP(p_i)$. Let $\mathcal{S}_1$ be the set of points in $\mathcal{S}$ for which $C_s^i = C_{s'}^i$, and let $\mathcal{S}_2$ be the remaining set of points in $\mathcal{S}$; see Figure 1 for an illustration. To compute the two watchman routes, we consider $\mathcal{S}_1$ and $\mathcal{S}_2$ separately in Sections 3.1 and 3.2, respectively.

3.1 Handling the Points in $\mathcal{S}_1$

Let $t_s^{p_i}$ and $t_{s'}^{p_i}$ be the shortest paths from s and s', respectively, to $VP(p_i)$. The endpoints of $t_s^{p_i}$ and $t_{s'}^{p_i}$ different from s and s' lie on windows C_s^i and $C_{s'}^i$, respectively. If $C_s^i = C_{s'}^i$, then p_i belongs to the set $\mathcal{S}_1$ and we denote the window C_i. Our objective in this section is to show how to compute two routes V_s^1 and $V_{s'}^1$ of optimal min-max length such that they together see all the points in $\mathcal{S}_1$.

Consider the set $P \setminus VP(p_i)$ of disjoint subpolygons of P. Let P_i be the subpolygon that contains s and s', for each point $p_i \in \mathcal{S}_1$. If $P_i \subset P_j$, we say that p_j *dominates* p_i, $i \neq j$, since any route through s or s' that visits $VP(p_j)$ must also visit $VP(p_i)$. We call any point in $\mathcal{S}_1$ *dominating* if it is not dominated by any other point in $\mathcal{S}_1$ and assume henceforth that $\mathcal{S}_1$ contains only dominating points.

For the sake of convenience in this section, we relabel each window C_i associated to the point $p_i \in \mathcal{S}_1$ based on the first appearance of its bottom point on the boundary of P along a clockwise traversal of the boundary. To do this, start at vertex v_1 and traverse the boundary in clockwise direction. Then, relabel the windows to $C_0, C_1, \ldots, C_{|\mathcal{S}_1|-1}$ in the order as the bottom point of each window is encountered. We order the associated points in $\mathcal{S}_1$ correspondingly.

Let T_s^1 and $T_{s'}^1$ be two min-max length routes that together see all points in $\mathcal{S}_1$.

▶ **Lemma 2.** *Let $C_0, C_1, \ldots, C_{|\mathcal{S}_1|-1}$ be the windows of the dominating points in $\mathcal{S}_1$, ordered as their bottom points appear along the boundary in clockwise order. If T_s^1 and $T_{s'}^1$ do not intersect, then there exist at most two indices i and j such that T_s^1 visits $C_i, C_{i+1}, \ldots, C_{j-1}$ and $T_{s'}^1$ visits $C_j, C_{j+1}, \ldots, C_{i-1}$. (Implicitly assuming that indices are taken modulo $|\mathcal{S}_1|$.)*

Proof. For three distinct indices i, i', and i'', we have that $i \prec i' \prec i''$ if one of the following sequences of inequalities holds: $i < i' < i''$, or $i'' < i < i'$, or $i' < i'' < i$, i.e., there exists a boundary point on P such that a clockwise traversal of the boundary starting at this point reaches the bottom points of the windows in the order C_i, $C_{i'}$, $C_{i''}$.

Assume that there are indices $i_1 \prec j_1 \prec i_2 \prec j_2 \prec i_1 \prec j_1$ such that T_s^1 intersects C_{i_1} and C_{i_2}, but not C_{j_1} and C_{j_2}, and $T_{s'}^1$ intersects C_{j_1} and C_{j_2}, but not C_{i_1} and C_{i_2}. Consider the polygonal chain defined by the part of C_{i_1} between its bottom point and its intersection with T_s^1, the part of T_s^1 until its intersection with C_{i_2}, and the part of C_{i_2} until its top point; see Figure 3. This chain separates P into two regions. By definition, s lies in one of these regions P_s. As T_s^1 does not visit C_{j_1}, the window C_{j_1} lies in the other region, $\bar{P}_s = P \setminus P_s$. Let $\hat{P}_s \subseteq \bar{P}_s$ be the region enclosed by C_{i_1}, C_{i_2}, T_s^1, and, if C_{i_1} and C_{i_2} do not intersect, possibly part of the polygon boundary; see Figure 3(b). The region $\hat{P}_s$ may be empty; see Figure 3(a) for such an example. Since s' lies outside the visibility polygons associated to the four windows we consider, s' either lies in P_s or in $\hat{P}_s$.

If s' lies in P_s, then $T_{s'}^1$ has to cross C_{i_1} or C_{i_2} in order to visit C_{j_1}, contradicting the stated assumptions; see Figures 3(a) and (b). If s' lies in $\hat{P}_s$, then in order for $T_{s'}^1$ to visit C_{j_2}, it must cross the boundary of $\hat{P}_s$, again contradicting our stated assumptions. Hence, T_s^1 (and $T_{s'}^1$) visits an ordered subsequence of the windows as stated. ◀

The previous lemma gives us an immediate algorithm to compute T_s^1 and $T_{s'}^1$. By trying every index pair $0 \le i, j \le |\mathcal{S}_1| - 1$, computing the shortest route starting at s that visits the windows $C_i, C_{i+1}, \ldots, C_{j-1}$ and the shortest route starting at s' that visits the windows $C_j, C_{j+1}, \ldots, C_{i-1}$, indices always taken modulo $|\mathcal{S}_1|$, and selecting the min-max shortest pair of these, we obtain the two desired routes $V_s^1 = T_s^1$ and $V_{s'}^1 = T_{s'}^1$. There are $O(m^2)$ index pairs, and computing the shortest route takes $O(m^3 n)$ for each of these pair, as described in Section 2. Thus, the time complexity is bounded by $O(m^5 n)$. We state this result as a theorem.

▶ **Theorem 3.** *The shortest min-max pair of routes in P that see the points in $\mathcal{S}_1$ can be computed in $O(m^5 n)$ time under the assumption that they do not intersect.*

3.2 Handling the Points in $\mathcal{S}_2$

Let C_s^i and $C_{s'}^i$ be the windows of $VP(p_i)$ that $t_s^{p_i}$ and $t_{s'}^{p_i}$ reach, respectively. If $C_s^i \neq C_{s'}^i$, then p_i belongs to the set $\mathcal{S}_2$. The visibility polygon $VP(p_i)$ must intersect the shortest path $SP(s, s')$ since the starting points lie in different components of $P \setminus VP(p_i)$. Our objective in this section is to show how to compute two routes V_s^ε and $V_{s'}^\varepsilon$ of almost optimal min-max length such that they together see all the points in $\mathcal{S}_2$.

First, we introduce a concept that we will use extensively in the sequel. Let L be some length $L \le \|SP(s, s')\|$. If p_L is the point at distance L from s on $SP(s, s')$, we can trace the geodesic quarter circle Q_U of radius L upwards from p_L, the connected locus of points having shortest paths in P of length L to s, ending the trace if it encounters the polygon boundary. Similarly, we can trace the geodesic quarter circle Q_D of radius L downwards from p_L; see Figure 4(a). Define $H_s^L = Q_U \cup Q_D$ as the *geodesic half circle* of radius L from s. We call the region $F_s^L = \{SP(s, p) \mid p \in H_s^L\}$ that is bounded by $Q_U \cup Q_D$ and the two shortest paths from s to the endpoints of H_s^L the *funnel* from s of radius L. We symmetrically define the geodesic half circle $H_{s'}^L$ and the funnel $F_{s'}^L$ from s' of radius L.

Recall that $t_{\min}^{p_i} = \arg\min\{\|t_s^{p_i}\|, \|t_{s'}^{p_i}\|\}$ is the shorter of the two paths from s and s' to the visibility polygon $VP(p_i)$, for every point $p_i \in \mathcal{S}$. Let $B = \max_{p_i \in \mathcal{S}_2}\{\|t_{\min}^{p_i}\|\}$ be the longest among all such paths for the points in $\mathcal{S}_2$, and let F_s^B and $F_{s'}^B$ be the funnels of radius B from s and s', respectively. We can show that for every point $p_i \in \mathcal{S}_2$, one of its windows C_s^i, $C_{s'}^i$ intersects the funnel (of radius B) from the corresponding starting point.

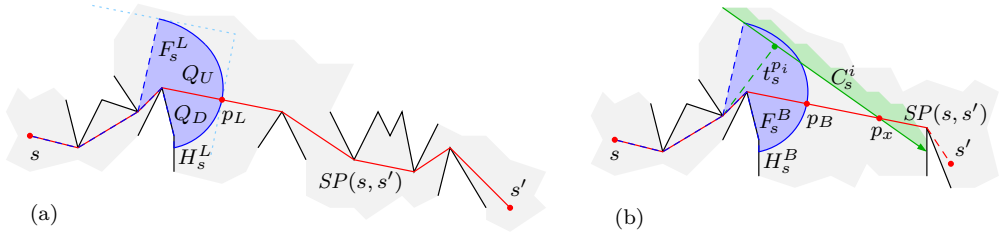


Figure 4 (a) Illustrating the definition of a funnel. (b) Illustrating the proof of Lemma 4.

▶ **Lemma 4.** *For any point $p_i \in \mathcal{S}_2$, either $VP(p_i) \cap F_s^B \neq \emptyset$ or $VP(p_i) \cap F_{s'}^B \neq \emptyset$.*

Proof. Consider the point $p_i \in \mathcal{S}_2$. Both windows C_s^i and $C_{s'}^i$ of $VP(p_i)$ intersect $SP(s,s')$. Without loss of generality assume that $t_s^{p_i} = t_{\min}^{p_i}$, the other case is proved in the same way.

If C_s^i intersects the subpath $SP(s,p_B)$, where p_B is the point at distance B from s along $SP(s,s')$, then C_s^i intersects the funnel F_s^B and the lemma holds. If C_s^i does not intersect the subpath $SP(s,p_B)$, then it intersects the subpath $SP(s',p_B)$ at p_x; see Figure 4(b). The pseudotriangle bounded by $t_s^{p_i}$, C_s^i, and $SP(s,p_x)$ must have an acute angle at p_x, since the angle between $t_s^{p_i}$ and C_s^i is right or obtuse. Therefore, C_s^i must intersect the geodesic half circle H_s^B, since $\|t_s^{p_i}\| \leq B$ and $t_s^{p_i}$ is a subpath of one of the paths $F_s^B = \{SP(s,p) \mid p \in H_s^B\}$, hence $VP(p_i)$ intersects F_s^B also in this case. ◀

Next, we compute an initial solution for which we can bound the min-max length. Let $\mathcal{S}_s^{\text{std}}$ and $\mathcal{S}_{s'}^{\text{std}}$ be a partitioning of the points in $\mathcal{S}_2$ that we obtain as follows. If for a point $p_i \in \mathcal{S}_2$ $\|t_s^{p_i}\| \leq \|t_{s'}^{p_i}\|$, then p_i belongs to $\mathcal{S}_s^{\text{std}}$, otherwise p_i belongs to $\mathcal{S}_{s'}^{\text{std}}$. We call this the *standard partitioning* of the points in $\mathcal{S}_2$. It follows from Lemma 4 that each visibility polygon $VP(p_i)$, for every $p_i \in \mathcal{S}_2$, is at distance at most B from one of s or s'.

Let V_s^{std} and $V_{s'}^{\text{std}}$ be the two shortest routes that see the points in $\mathcal{S}_s^{\text{std}}$ and $\mathcal{S}_{s'}^{\text{std}}$, respectively, that we can compute with the modified method of Tan and Jiang [11].

▶ **Lemma 5.** *The length of the two routes V_s^{std} and $V_{s'}^{\text{std}}$ is bounded by*

$$\max\left\{\|V_s^{\text{std}}\|, \|V_{s'}^{\text{std}}\|\right\} \leq \left(1 + \frac{\pi}{2}\right) OPT.$$

Proof. Consider the boundary of the funnel F_s^B. It consists of two paths of length B and a sequence of connecting arcs of circles having radius at most B; the sum of their lengths is bounded by πB. The perimeter is a route that intersects all the visibility polygons of the points in $\mathcal{S}_s^{\text{std}}$. Since V_s^{std} is the shortest route that intersects these visibility polygons, its length is bounded by $(2+\pi)B$. The length of the perimeter of $F_{s'}^B$ is bounded in the same way and since $OPT \geq 2B$, the lemma follows. ◀

Next, we improve on the previous result by updating the standard partitioning and obtain a $(1+\varepsilon)$-approximation, for any $\varepsilon > 0$. We consider ε to be a constant, sufficiently small, and define the integer $k = \lceil 69/\varepsilon \rceil$.

We partition the points into two sets in such a way that for each point the shortest path from the corresponding starting point is within a $(1+\varepsilon)$ factor from the optimum. In order to do this, we make a decomposition of the points in $\mathcal{S}_2$ into few subsets such that those points for which their corresponding windows are "similar" are placed in the same subset. By then choosing one representative point from each subset, we can compute the optimal partition of them using brute force, since there are only a few points to partition. The partition of the representative points indicates in which partition each similar point should be in. Once

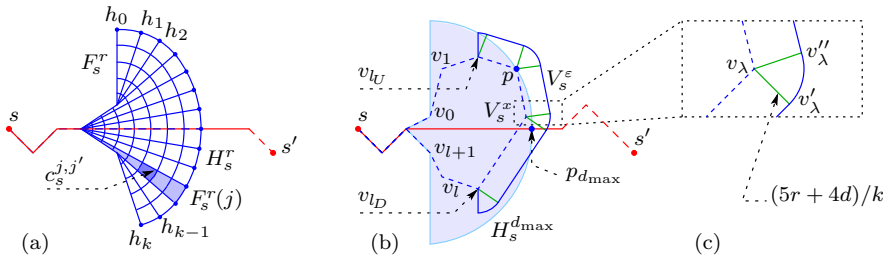


Figure 5 (a) Description of the cells in the similarity decomposition. (b)-(c) Construction of V_s^ε (blue) from V_s^x (dashed blue).

the partition for the full set of points has been computed, we can compute the two shortest watchman routes using the algorithm described in Section 2. We provide the details of the construction below.

For any point $p_i \in \mathcal{S}_2$ where $\|t_{\max}^{p_i}\| > 3B$, we can immediately assume that p_i must be seen by the route from the starting point for which the geodesic is shorter. Otherwise, $OPT > 6B$, as the optimal route must at least be as long as the geodesic $t_{\max}^{p_i}$, followed to $VP(p_i)$ and back. In this case, the standard partitioning together with the optimal routes for $\mathcal{S}_1$, i.e., the solution $(V_s^1 \cup V_s^{\text{std}}, V_{s'}^1 \cup V_{s'}^{\text{std}})$ gives at worst a 2-approximation to our problem, since $\max\left\{\|V_s^{\text{std}}\|, \|V_{s'}^{\text{std}}\|\right\} \leq (2+\pi)B < 6B \leq OPT$. Let $\mathcal{S}_2^{\text{L}}$ be the subset of the points in $\mathcal{S}_2$ for which $t_{\max}^{p_i}$ is "too long", in particular, $\|t_{\max}^{p_i}\| > 3B$, and assume they are partitioned according to the starting point of their shorter geodesic $t_{\min}^{p_i}$. Furthermore, let $\mathcal{S}_2^{\text{S}} = \mathcal{S}_2 \setminus \mathcal{S}_2^{\text{L}}$ be the remaining points in $\mathcal{S}_2$ for which $t_{\max}^{p_i}$ is "short". It remains to find an appropriate partition of the points in $\mathcal{S}_2^{\text{S}}$. To do this, we define a refinement of the funnel, and partition it into k^2 cells.

Define $r \stackrel{\text{def}}{=} \min\{3B, \|SP(s,s')\|/2\}$. For the points $p_i \in \mathcal{S}_2^{\text{S}}$, we have $\|t_{\max}^{p_i}\| \leq r$ by definition, giving us that $\|t_s^{p_i}\|, \|t_{s'}^{p_i}\| \leq r$. Let F_s^r and $F_{s'}^r$ be the two funnels of radius r from s and s', respectively.

We make a similarity decomposition of the set of points in $\mathcal{S}_2^{\text{S}}$ into at most k^4 distinct subsets $\mathcal{S}_2^{(1)}, \mathcal{S}_2^{(2)}, \ldots, \mathcal{S}_2^{(k^4)}$ as follows. H_s^r is the geodesic half circle of radius r centered at s; see Figure 5(a). Consider the $k+1$ consecutive equidistant points h_j, for $0 \leq j \leq k$, along H_s^r, h_0 and h_k being the two endpoints of H_s^r. The points of F_s^r between the two geodesics $SP(s, h_{j-1})$ and $SP(s, h_j)$, for each $1 \leq j \leq k$, define a *subfunnel* of F_s^r, denoted $F_s^r(j)$, lightly shaded in Figure 5(a). We can further decompose each subfunnel $F_s^r(j)$ into k cells $c_s^{j,j'} = \left\{p \in F_s^r(j) \,\middle|\, r \cdot (j'-1)/k < \|SP(s,p)\| \leq r \cdot j'/k\right\}$, for $1 \leq j' \leq k$, depending on the distance to s. A cell $c_s^{j,j'}$ is more heavily shaded in Figure 5(a). The funnel F_s^r thus decomposes into k^2 cells. Similarly, we decompose the funnel $F_{s'}^r$ into k^2 cells $c_{s'}^{j,j'}$, for $1 \leq j, j' \leq k$.

We say that a window C_s^i *belongs to* cell $c_s^{j,j'}$ if the intersection point of $t_s^{p_i}$ and C_s^i is a point in the cell $c_s^{j,j'}$. The point $p_i \in \mathcal{S}_2^{\text{S}}$ lies in the subset $\mathcal{S}_2^{(\kappa)}$ if C_s^i belongs to $c_s^{j_s,j'_s}$, $C_{s'}^i$ belongs to $c_{s'}^{j_{s'},j'_{s'}}$, and $\kappa = j_s + (j'_s - 1)k + (j_{s'} - 1)k^2 + (j'_{s'} - 1)k^3$. In this way we have partitioned all the points in $\mathcal{S}_2^{\text{S}}$ into at most k^4 distinct subsets.

For each non-empty subset, we select one point as the representative for the subset and let $\mathcal{R} \subseteq \mathcal{S}_2^{\text{S}}$ be the set of all the chosen representative points. By trying all $2^{|\mathcal{R}|} \leq 2^{k^4}$ partitions of the representative points of $\mathcal{R}$ into two subsets, we compute the two shortest tours V_s^x and $V_{s'}^x$ that visit the visibility polygons of the points in $\mathcal{R} \cup \mathcal{S}_2^{\text{L}}$, over all possible partitions of $\mathcal{R}$.

Let V_s^{ref} and $V_{s'}^{\text{ref}}$ be the two shortest routes obtained from the partition given by placing each point in a subset $\mathcal{S}_2^{(\kappa)}$ in the same partition as the representative point of this subset. The two routes can be computed in $O(m^3 n)$ time. To bound the lengths of V_s^{ref} and$V_{s'}^{\text{ref}}$, we introduce two auxiliary routes V_s^{ε} and $V_{s'}^{\varepsilon}$ that are easier to bound and prove that V_s^{ref} and $V_{s'}^{\text{ref}}$ are not longer than V_s^{ε} and $V_{s'}^{\varepsilon}$, respectively.

Let $d = \max\left\{\|V_s^x\|, \|V_{s'}^x\|\right\}$. We construct the route V_s^{ε} from V_s^x as follows ($V_{s'}^{\varepsilon}$ is constructed from $V_{s'}^x$ in the same way). Let $d_{\max}$ be the supremum value for which the geodesic half circle $H_s^{d_{\max}}$ intersects V_s^x, i.e., $d_{\max} = \sup_{L>0}\left\{H_s^L \cap V_s^x \neq \emptyset\right\}$. Let p be a point of contact between $H_s^{d_{\max}}$ and a convex vertex of V_s^x; see Figure 5(b). From p follow V_s^x in counterclockwise order until a reflex vertex v_0 of V_s^x is encountered. From p now follow V_s^x in clockwise order until a reflex vertex v_{l+1} of V_s^x is encountered, where l is the number of convex vertices on the clockwise path of V_s^x from v_0 to v_{l+1}. Let $p_{d_{\max}}$ be the point of intersection of $H_s^{d_{\max}}$ and $SP(s, s')$ and let θ be the angle between the tangent of $H_s^{d_{\max}}$ at $p_{d_{\max}}$ and the x-axis. For simplicity of construction, assume P, $F_s^{d_{\max}}$, and V_s^x to be rotated so that $\theta = \pi/2$, i.e., the tangent of $H_s^{d_{\max}}$ at $p_{d_{\max}}$ is vertical. Let v_{l^*}, for some $1 \leq l^* \leq l$, be the vertex of V_s^x such that the edges $[v_{l^*-1}, v_{l^*}]$ and $[v_{l^*}, v_{l^*+1}]$ lie on the same side of a vertical line through v_{l^*}, i.e., the rightmost vertex of V_s^x, if s lies locally to the left and s' lies locally to the right of $p_{d_{\max}}$. For some $1 \leq l_U \leq l^*$, assume the edges $[v_{\lambda-1}, v_\lambda]$ have non-negative slope, for all $1 \leq \lambda \leq l_U$. Similarly, for some $l^* \leq l_D \leq l$, assume the edges $[v_\lambda, v_{\lambda+1}]$ have non-positive slope, for all $l_D \leq \lambda \leq l$. Let the clockwise subpath from s to v_{l_U} of V_s^x also be the first part of V_s^{ε}. Add a vertical edge of length $(5r + 4d)/k$ at v_{l_U} upward to a new vertex v'_{l_U} and a vertical edge of length $(5r + 4d)/k$ at v_{l_D} downward to a new vertex v''_{l_D}. While $l_U \leq \lambda < l_D$, the segment $[v_\lambda, v_{\lambda+1}]$ has slope angle θ_λ, and we offset the segment by a distance $(5r + 4d)/k$ in the direction $\theta_\lambda + \pi/2$ (increasing angles going clockwise around a circle), which gives the segment $[v''_\lambda, v'_{\lambda+1}]$; see Figures 5(b) and (c). We connect v'_λ with v''_λ by a circle arc with origin v_λ and radius $(5r + 4d)/k$, for each $l_U \leq \lambda \leq l_D$ and finally connect v_{l_D} back to s with the corresponding subpath of V_s^x; again see Figures 5(b) and (c). If at some point the construction intersects the polygon boundary we shortcut the path so that it is the shortest to connect to in the free space of the polygon. We show that $(V_s^{\varepsilon}, V_{s'}^{\varepsilon})$ sees all points in $\mathcal{S}_2$ and bound the min-max length.

▶ **Lemma 6.** *The two-watchman route $(V_s^{\varepsilon}, V_{s'}^{\varepsilon})$ sees all points in $\mathcal{S}_2$ and*

$$\max\left\{\|V_s^{\varepsilon}\|, \|V_{s'}^{\varepsilon}\|\right\} \leq d + \frac{30r + 24d}{k}.$$

Proof. Recall that $(T_s, T_{s'})$ is an optimal min-max solution to the shortest two-watchman route problem for the points in $\mathcal{S}$, and let $(T_s^2, T_{s'}^2)$ be an optimal min-max solution for the points in $\mathcal{S}_2$. Since $(V_s^x, V_{s'}^x)$ is an optimal min-max solution for the problem on the points in $\mathcal{R} \cup \mathcal{S}_2^{\text{L}} \subseteq \mathcal{S}_2$, it follows that $\|V_s^x\| \leq \|T_s^2\| \leq \|T_s\| \leq OPT$ and $\|V_{s'}^x\| \leq \|T_{s'}^2\| \leq \|T_{s'}\| \leq OPT$, whereby $d \leq OPT$.

Let p_i be a point in $\mathcal{R} \cap \mathcal{S}_2^{(\kappa)}$, for some κ, $1 \leq \kappa \leq k^4$, and assume that V_s^x sees p_i. This implies that V_s^x intersects the window C_s^i at some point p. Consider any other point p_j in the same subset $\mathcal{S}_2^{(\kappa)}$ as p_i. Since the endpoints of $t_s^{p_i}$ and $t_s^{p_j}$ lie in the same cell, the angle between the windows C_s^i and C_s^j is at most π/k and the distance between the two windows within the cell is at most $(1 + \pi)r/k$ as this is an upper bound on the farthest distance between any two points in a cell (the height of the cell plus the width of the cell); see Figure 6. Let $\hat{p}_i$ be the endpoint of $t_s^{p_i}$ intersecting the window C_s^i. Thus, the distance from p to C_s^j is bounded by

$$\left\|p, C_s^j\right\| \leq \frac{(1 + \pi) \cdot r}{k} + \left\|\hat{p}_i, p\right\| \cdot \sin\frac{\pi}{k} \leq \frac{(1 + \pi) \cdot r}{k} + d\frac{\pi}{k} < \frac{5r + 4d}{k}.$$

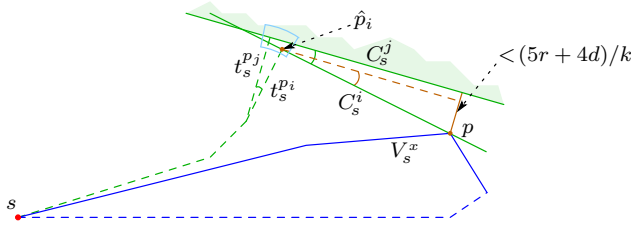


Figure 6 Bounding the distance between C_s^i and C_s^j at p.

Hence, if V_s^x sees p_i, then V_s^ε sees all points in $\mathcal{S}_2^{(\kappa)}$ as p_j was chosen arbitrarily in $\mathcal{S}_2^{(\kappa)}$. We can argue the other possibility when $V_{s'}^x$ sees p_i in the same way and therefore, since V_s^x and $V_{s'}^x$ see all representative points in $\mathcal{R}$, V_s^ε and $V_{s'}^\varepsilon$ together see all points in $\mathcal{S}_2$.

It remains to bound the lengths of V_s^ε and $V_{s'}^\varepsilon$. We do this only for V_s^ε as the argument for $V_{s'}^\varepsilon$ is the same. V_s^ε consists of translated segments from V_s^x (recall that $\|V_s^x\| \leq d$), two additional segments of length $(5r+4d)/k$, and circular arcs, also of radius $(5r+4d)/k$, where the angle sum is at most π; see Figures 5(b) and (c). The total length of V_s^ε is therefore bounded by

$$\|V_s^\varepsilon\| \leq \|V_s^x\| + \frac{2(5r+4d)}{k} + \frac{\pi(5r+4d)}{k} \leq d + \frac{30r+24d}{k},$$

as claimed. ◀

We summarize the results on handling the points in $\mathcal{S}_2$.

▶ **Theorem 7.** *The pair of watchman routes $(V_s^{\text{ref}}, V_{s'}^{\text{ref}})$, that see the points in $\mathcal{S}_2$, can be computed in $O(m^3 n)$ time, and has length bounded by*

$$\max\left\{\|V_s^{\text{ref}}\|, \|V_{s'}^{\text{ref}}\|\right\} \leq (1+\varepsilon)\,OPT.$$

Proof. By construction, the routes V_s^{ref} and $V_{s'}^{\text{ref}}$ visit the same windows as the routes V_s^ε and $V_{s'}^\varepsilon$ respectively. As V_s^{ref} and $V_{s'}^{\text{ref}}$ are the shortest routes that visit the windows in this way, $\|V_s^{\text{ref}}\| \leq \|V_s^\varepsilon\|$ and $\|V_{s'}^{\text{ref}}\| \leq \|V_{s'}^\varepsilon\|$. Since $r \leq 3B \leq 3OPT/2$ and $d \leq OPT$, we obtain

$$\begin{aligned}\max\{\|V_s^{\text{ref}}\|, \|V_{s'}^{\text{ref}}\|\} &\leq \max\{\|V_s^\varepsilon\|, \|V_{s'}^\varepsilon\|\} \leq d + \frac{30r+24d}{k} \leq OPT + \frac{45\,OPT + 24\,OPT}{k} \\ &\leq \left(1 + \frac{69}{k}\right) OPT \leq (1+\varepsilon)\,OPT\end{aligned}$$

by our choice of k. Moreover, we can compute V_s^{ref} and $V_{s'}^{\text{ref}}$ in $O(m^3 n)$ time, as mentioned in Section 2. ◀

3.3 Putting It All Together

Finally, we chose the shortest two routes among the possible pairs: a watchman route W_s starting at s and a stationary watchman at s'; a watchman route $W_{s'}$ starting at s' and a stationary watchman at s; and the combined watchman routes $V_s^1 \cup V_s^\varepsilon$ and $V_{s'}^1 \cup V_{s'}^\varepsilon$, where V_s^1 and $V_{s'}^1$ are the two routes that see the points in $\mathcal{S}_1$ as defined in Section 3.1, and V_s^ε and $V_{s'}^\varepsilon$ are the improved routes that see the points in $\mathcal{S}_2$ as computed in Section 3.2. Let

$$(V_s, V_{s'}) = \arg\min\Big\{\max\left\{\|W_s\|, \|s'\|\right\}, \max\left\{\|s\|, \|W_{s'}\|\right\}, \max\left\{\|V_s^1 \cup V_s^\varepsilon\|, \|V_{s'}^1 \cup V_{s'}^\varepsilon\|\right\}\Big\}.$$

By combining the algorithms developed in Sections 3.1 and 3.2 together with the bounds in Theorems 3 and 7, we immediately have the following theorem. Since ε is considered to be a constant, it does not affect the total running time.

▶ **Theorem 8.** *The two routes V_s and $V_{s'}$ see all points in $\mathcal{S}$, can be computed in $O(m^5 n)$ time, and*

$$\max\left\{\|V_s\|, \|V_{s'}\|\right\} \leq (2+\varepsilon)\, OPT,$$

for any fixed constant $\varepsilon > 0$.

We remark that the two routes V'_s and $V'_{s'}$ given by

$$(V'_s, V'_{s'}) = \arg\min\Big\{\max\left\{\|W_s\|, \|s'\|\right\}, \max\left\{\|s\|, \|W_{s'}\|\right\}, \max\left\{\|V_s^1 \cup V_s^{\text{std}}\|, \|V_{s'}^1 \cup V_{s'}^{\text{std}}\|\right\}\Big\}$$

using the standard partitioning instead of the improved partitioning can be computed in $O(m^5 n)$ time, have a min-max length bound of $(2+\pi/2)\,OPT$, but do not require exhaustive search to find the correct partition of the representative points in set $\mathcal{R}$.

4 Computing the Shortest Watchman Routes That See All of P

We can generalize the method obtained in Section 3 to find two watchman routes $(V_s, V_{s'})$ that together see the whole polygon P. We assume that the two optimal routes T_s and $T_{s'}$ for Problem 2 do not intersect, as otherwise a shortest watchman route W_s starting at s, or a shortest watchman route $W_{s'}$ starting at s', together with a route of length 0 at the respective other starting point, gives a 2-approximation (as explained in Step 1 in Section 3).

We carefully choose a set $\mathcal{S}$ of $O(n)$ points along the boundary of P with the property that whenever V_s and $V_{s'}$ see these points, they see all of the boundary of P, and for two watchmen, this is enough to see the polygon.

▶ **Theorem 9** (Lemma 3.1 in [8]). *If two watchmen together see all of the boundary of a polygon P, then they see all of P.*

Consider a point p on the boundary of P that is not seen from s or s', and let $VP(p)$ be the visibility polygon of p. We use the notation P_s^p to denote the subpolygon in $P \setminus VP(p)$ that contains s, and $P_{s'}^p$ to denote the subpolygon in $P \setminus VP(p)$ that contains s'. Analogous to before, we call the edge $C_s^p = VP(p) \cap P_s^p$ the *window* of p with respect to s, and the edge $C_{s'}^p = VP(p) \cap P_{s'}^p$ the *window* of p with respect to s'.

We want to see every point on the boundary. To be able to apply the strategy from Section 3, we start by defining two infinite sets $\mathcal{S}_1$ and $\mathcal{S}_2$, and allocate every boundary point into one of them based on the windows of it. Let $\mathcal{S}_1$ be the set of boundary points for which $C_s^p = C_{s'}^p$, and let $\mathcal{S}_2$ be the remaining set of boundary points. This partitions every polygon edge into intervals of points that lie in the same set, $\mathcal{S}_1$ or $\mathcal{S}_2$. We call such an interval a *subedge* and denote the set of subedges with points in $\mathcal{S}_1$ by $\mathcal{E}_1$ and the set of subedges with points in $\mathcal{S}_2$ by $\mathcal{E}_2$. The endpoints of the subedges are either vertices of P, i.e., points in $\mathcal{V}$, or interior points of the edges. We call these interior points *transition points*, and denote the set of all transition points by $\mathcal{T}$. Observe that each transition point is defined by one of the following:

- The intersection of the polygon boundary with a line through two polygon vertices
- A vertex of the visibility polygons $VP(s)$ and $VP(s')$

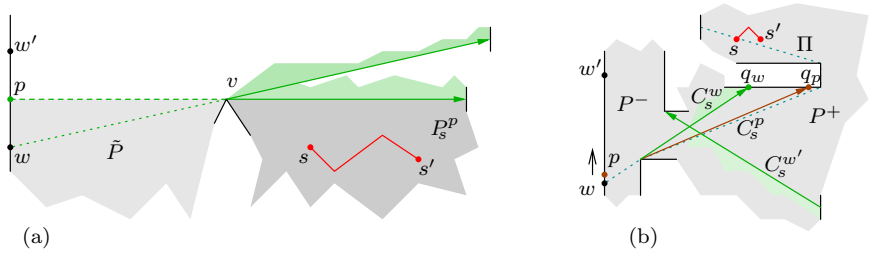


Figure 7 (a) Illustrating the proof of Lemma 10. (b) Determining the split point of a subedge in $\mathcal{E}_1$.

We can trivially construct the arrangement of lines through polygon vertices in $O(n^4)$ time, using a brute-force algorithm. This arrangement has complexity $O(n^2)$, and the polygon consists of n edges. Thus, the intersection points of the arrangement and the polygon edges can be computed in $O(n^3)$ time. The visibility polygon of a point in a polygon can be computed in $O(n)$ time, see [5]. With this, $\mathcal{T}$ can be computed in $O(n^4)$ time (note that this is not optimal, but does not affect the final complexity of our algorithm).

The visibility polygon of every point in $\mathcal{S}_2$ intersects the shortest path between the two starting points $SP(s, s')$. Conversely, the visibility polygon of any point in $\mathcal{S}_1$ cannot intersect $SP(s, s')$. As the area seen by a point that moves along a polygon boundary edge changes continuously, we observe the following.

▶ **Lemma 10.** *If w and w' are two points in $\mathcal{S}_2$ that lie on the same edge of P then, every point between w and w' is also in $\mathcal{S}_2$.*

Proof. Assume for a contradiction that there exists a point p in the interior of ww' that lies in $\mathcal{S}_1$. Then, the shortest path $SP(s, s')$ between the starting points lies fully inside $P_s^p = P_{s'}^p$. Let v be the vertex endpoint on C_s^p that is closest to p, and consider the segment pv. This segment splits P into two subpolygons, one containing w and one containing w'. Assume w.l.o.g. that the subpolygon $\tilde{P}$ that contains w does not contain P_s^p, see Figure 7(a). Then, $VP(w) \cap SP(s, s') = \emptyset$, since no ray emanating from w can intersect both pv and C_s^p except at v and w then belongs to $\mathcal{S}_1$, a contradiction. ◀

Thus, every edge contains at most one subedge in $\mathcal{E}_2$ and at most two subedges in $\mathcal{E}_1$, and there is at most a linear number of transition points.

To see the polygon boundary, we need to ensure that every subedge is seen. In a first step, we apply the strategy from Section 3 to $\mathcal{T} \cup \mathcal{V}$. This ensures that all points in $\mathcal{T} \cup \mathcal{V}$ are seen. To do so, we partition the points in $\mathcal{V}$ into two sets $\mathcal{V}_1 = \mathcal{V} \cap \mathcal{S}_1$ and $\mathcal{V}_2 = \mathcal{V} \cap \mathcal{S}_2$, and consider the two discrete, linear-sized sets, $\tilde{\mathcal{S}}_1 = \mathcal{V}_1 \cup \mathcal{T}$ and $\tilde{\mathcal{S}}_2 = \mathcal{V}_2 \cup \mathcal{T}$. We take care of the subedges in $\mathcal{E}_1$ and $\mathcal{E}_2$ separately. By adding the transition points to both sets $\tilde{\mathcal{S}}_1$ and $\tilde{\mathcal{S}}_2$, we ensure that the endpoints of each subedge in $\mathcal{E}_1$ lie in $\tilde{\mathcal{S}}_1$, and the endpoints of each subedge in $\mathcal{E}_2$ lie in $\tilde{\mathcal{S}}_2$.

Again, our goal is to take care of each subedge separately. We assign a direction to C_s^p so that $VP(p)$ lies locally to the left of C_s^p, if the polygon is rotated so that the window is directed upwards. The shortest path t_s^p that visits C_s^p is fully contained in the subpolygon P_s^p, and visits C_s^p locally from the right. We analogously assign a direction to $C_{s'}^p$.

We refine our set of subedges by splitting every subedge in $\mathcal{E}_1$ into segments that we call *fragments* such that for every point on a fragment, the windows of the points have the same orientation, either towards the subedge or away from the subedge.

▶ **Lemma 11.** *Every subedge ww' in $\mathcal{E}_1$ is split into at most two fragments, that is, a transition between points with windows of different orientations happens at most once when sliding along ww'.*

Proof. Consider the path $SP(s, w)$. Extending the first segment of $SP(s, w)$ (adjacent to s) backwards until it hits the boundary of P gives us a shortest path Π connecting two boundary points of P. The path Π separates P into two simply connected polygons P^+ and P^-, P^+ lying locally to the left of (or on) Π as we move from s towards w along Π and P^- lying locally to the right of (or on) Π as we move from s towards w. Assume, without loss of generality, that the farthest endpoint q_w of C_s^w lies in P^-, otherwise we reverse the roles of P^+ and P^-. An illustration is given in Figure 7(b). The point q_w is a bottom or top point of C_s^w. Let p be a point that moves from w to w' along the subedge and consider the window C_s^p. As p moves from w to w', the corresponding top (bottom) point q_p of the window C_s^p traverses the polygon boundary in P^-. If C_s^p aligns with a segment of Π at $p = p^*$, the point q_p moves from P^- to P^+ and C_s^p flips its orientation, q_p goes from being a bottom (top) point in the open ended interval wp^* to being a top (bottom) point in the open ended interval p^*w'. We call such a point p^* a *split point*. Since Π is a shortest path, the alignment of C_s^p and a segment of Π can occur only once. Hence, every subedge is split into at most two fragments at a split point. ◀

For an edge ww', the split point is given by the intersection of the polygon boundary and a line through two polygon vertices, and can therefore be computed in the same way as the transition points. We replace the subedge in $\mathcal{E}_1$ by these fragments, add the split points to $\mathcal{T}$, and update $\tilde{\mathcal{S}}_1$ and $\tilde{\mathcal{S}}_2$ accordingly.

With this modification, we handle the points in $\tilde{\mathcal{S}}_1$ in the same way as described in Section 3.1, and obtain the two routes V_s^1 and $V_{s'}^1$. The following Lemma guarantees that every subedge in $\mathcal{E}_1$ is seen.

▶ **Lemma 12.** *Let ww' be a fragment in $\mathcal{E}_1$. Either V_s^1 or $V_{s'}^1$ sees all of ww'.*

Proof. Since w and w' lie on the same fragment of $\mathcal{E}_1$, one of the two subpolygons $P_s^w, P_s^{w'}$ is fully contained in the other, say w.l.o.g. $P_s^w \supset P_s^{w'}$. The situation is as depicted in Figure 7(b), with w' at p. Thus, if V_s^1 visits C_s^w to see w, it also visits $C_s^{w'}$, and since ww' is part of a polygon edge, V_s^1 sees every point on ww'. The case for $V_{s'}^1$ is analogous. ◀

Next, we handle the points in $\tilde{\mathcal{S}}_2$ as described in Section 3.2 by computing the standard partitioning of the points. We make use of the following lemma.

▶ **Lemma 13** (Lemma 4.1 in [8])**.** *For a subedge $ww' \in \mathcal{E}_2$, if a watchman sees w and w' then it sees the whole subedge ww'.*

Thus, only those subedges in $\mathcal{E}_2$ where each endpoint is seen by a different watchman, meaning that its two endpoints are separated in the standard partitioning, are not yet seen. Let $\mathcal{E}_2^{\text{diff}}$ be the set of these subedges. For our approximate solution, we aim to find a good way for both watchmen to cooperatively guard the edges in $\mathcal{E}_2^{\text{diff}}$. To do so, we refine our discretization by splitting every subedge in $\mathcal{E}_2^{\text{diff}}$ into two, and let each watchman guard one of the resulting subedges.

For each subedge ww' in $\mathcal{E}_2^{\text{diff}}$, we compute the point p on ww' for which both watchmen have an equal distance from their starting point to the window of p's visibility polygon, i.e., $\|t_s^p\| = \|t_{s'}^p\|$, see Figure 8. Then, we replace ww' with wp and pw'. This can be done in $O(n)$ time per subedge, using the strategy described in [8]. We also add p to a third set of

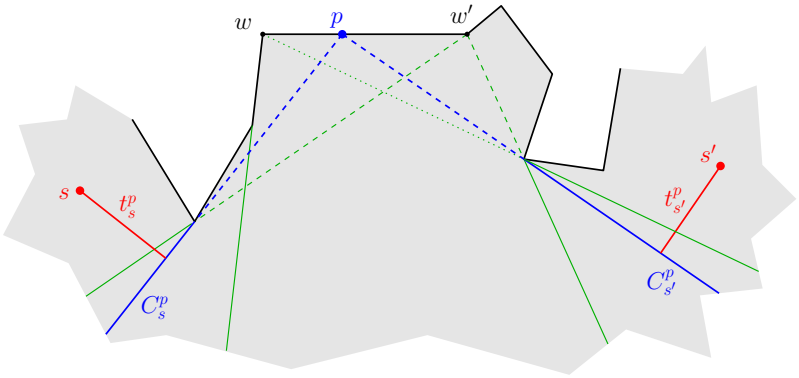


Figure 8 Splitting ww' at p where $\|t_s^p\| = \|t_{s'}^p\|$.

points $\tilde{\mathcal{S}}_3$, and modify the partitioning of $\tilde{\mathcal{S}}_2$ by adding all points in $\tilde{\mathcal{S}}_3$ to both partitions. With this, we simultaneously partition the subedges in $\mathcal{E}_2^{\text{diff}}$ based on their endpoints. Finally, we compute the shortest watchman routes $(V_s^{\text{std}}, V_{s'}^{\text{std}})$ for this new set of points. The two endpoints of a subedge $ww' \in \mathcal{E}_2^{\text{diff}}$ are seen by the same watchman. Hence, by Lemma 13, the whole subedge is seen.

Furthermore, recall that we construct the standard partitioning by assigning every point to a starting point based on its distance to the starting point. Putting the points in $\tilde{\mathcal{S}}_3$ into both partitions does not contradict the bound $B = \max_{p \in \tilde{\mathcal{S}}_2 \cup \tilde{\mathcal{S}}_3} \min\left\{\|t_s^p\|, \|t_{s'}^p\|\right\}$ on the maximum length within each partition, since for every point $p \in \tilde{\mathcal{S}}_3$, $\min\left\{\|t_s^p\|, \|t_{s'}^p\|\right\} = \|t_s^p\| = \|t_{s'}^p\| \leq B$. Therefore, the bound of Lemma 5 also holds for the modified set $\tilde{\mathcal{S}}_2$, hence we get

$$\max\left\{\|V_s^{\text{std}}\|, \|V_{s'}^{\text{std}}\|\right\} \leq \left(1 + \frac{\pi}{2}\right) OPT.$$

Finally, as in Section 3.3, we choose the shortest set of watchman routes as our solution

$$(V_s, V_{s'}) = \arg\min\Big\{\max\left\{\|W_s\|, \|s'\|\right\}, \max\left\{\|s\|, \|W_{s'}\|\right\}, \max\left\{\|V_s^1 \cup V_s^{\text{std}}\|, \|V_{s'}^1 \cup V_{s'}^{\text{std}}\|\right\}\Big\}.$$

Since the set $\mathcal{T}$ contains at most $O(n)$ points, we obtain a set of two watchman routes for P of length at most $(2 + \pi/2)OPT$ in $O(n^6)$ time.

▶ **Theorem 14.** *The two routes V_s and $V_{s'}$ see all of P, can be computed in $O(n^6)$ time, and*

$$\max\left\{\|V_s\|, \|V_{s'}\|\right\} \leq \left(2 + \frac{\pi}{2}\right) OPT.$$

5 Adaptations for the Floating Variant

Nilsson and Packer [8] show how to compute two points s and s' having the property that: for any point p on the boundary of P, the shortest of the paths t_s^p and $t_{s'}^p$ has length bounded by $OPT^F/2$, where OPT^F is the min-max length of the optimum floating two-watchman route. Recall that in Section 4, we defined a set of points $\mathcal{T}$ on the boundary. Therefore, for the two points s and s', the stated property holds in particular for the points in $\mathcal{T}$. The computation of s and s' takes $O(n^8)$ time.

Using the method derived in Section 4, we split the points in $\mathcal{T}$ into $\mathcal{S}_1$ and $\mathcal{S}_2$ and can obtain a pair of watchman routes V_F and V_F' through the points s and s', respectively, in additional $O(n^6)$ time. These two routes consist of two subroutes $V_F = V_s^1 \cup V_s^{\text{std}}$ and

$V'_F = V^1_{s'} \cup V^{\text{std}}_{s'}$, where V^1_s and $V^1_{s'}$ see the points in $\mathcal{S}_1$, and the subroutes V^{std}_s and $V^{\text{std}}_{s'}$ see the points in $\mathcal{S}_2$. The length of each of V^1_s and $V^1_{s'}$ is bounded by $2 \cdot OPT^F$, since the optimal routes are not guaranteed to pass through s and s' but there is some point on each of the two optimal routes at a distance at most $OPT^F/2$ from s and s', respectively. The length of V^{std}_s and $V^{\text{std}}_{s'}$ is bounded by the length of the perimeter of its funnel, i.e., $(1 + \pi/2)OPT^F$; see Lemma 5. We therefore conclude

▶ **Corollary 15.** *For a polygon P, the pair (V_F, V'_F) of watchman routes that see all of P can be computed in $O(n^8)$ time, and*

$$\max\left\{\|V_F\|, \|V'_F\|\right\} \leq \left(3 + \frac{\pi}{2}\right) OPT^F,$$

where OPT^F is the maximum length of an optimal pair of watchman routes for the floating two-watchman route problem in P.

6 Conclusion

We have addressed the two-watchman route problem in simple polygons by providing an approximation algorithm that is based on a discretization of the polygon boundary. The hardness in finding a good solution lies in determining a partition of the polygon boundary pieces into two subsets, such that each piece is seen by one of the two watchmen. For more than two watchmen, to see a discrete point set, a generalization of our partitioning scheme could be a promising strategy. However, to see the whole polygon with three (or more) watchmen, seeing the polygon boundary is not sufficient. Obtaining such approximation algorithms therefore requires different methods.

Since optimal watchman routes for the min-sum objective behave differently than for the min-max objective, our strategy does not apply, except for the crude observation that the longest of two min-max optimal routes is at most as long as the longest of two min-sum optimal routes. This adds a multiplicative factor of 2 on our results for the min-max measure, giving an approximation factor of ≈ 7.142 for the anchored version, and a factor of ≈ 9.142 for the floating version – better than the currently best known factors of ≈ 13.844 for the anchored version and ≈ 11.939 for the floating version [8]. Nilsson and Packer's approximation factor for the anchored version is larger than for the floating version as their anchored routes result from enforcing the floating routes to pass through the starting points, while in our algorithms the anchored routes are constructed from the starting points. Finally, the computational complexity of the min-sum two-watchman route problem in simple polygons remains open.

References

1 Svante Carlsson and Håkan Jonsson. Guarding a treasury. In *Proceedings of the 5th Canadian Conference on Computational Geometry, Waterloo, Ontario, Canada*, pages 85–90. University of Waterloo, 1993.

2 Wei-pang Chin and Simeon C. Ntafos. Optimum watchman routes. *Inf. Process. Lett.*, 28(1):39–44, 1988. doi:10.1016/0020-0190(88)90141-X.

3 Wei-pang Chin and Simeon C. Ntafos. Shortest watchman routes in simple polygons. *Discr. Comput. Geom.*, 6:9–31, 1991. doi:10.1007/BF02574671.

4 Linda L. Deneen and Shashikant Joshi. Treasures in an art gallery. In *Proceedings of the 4th Canadian Conference on Computational Geometry, St. John's, Newfoundland, Canada*, pages 17–22. Memorial University of Newfoundland, 1992.

5 Barry Joe and R. B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT*, 27(4):458–473, 1987. doi:10.1007/BF01937271.

6 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

7 Joseph S. B. Mitchell and Erik L. Wynters. Watchman routes for multiple guards. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 126–129, 1991.

8 Bengt J. Nilsson and Eli Packer. Approximation algorithms for the two-watchman route in a simple polygon. *Algorithmica*, 86(9):2845–2884, 2024. doi:10.1007/S00453-024-01245-0.

9 Xuehou Tan. Fast computation of shortest watchman routes in simple polygons. *Inf. Process. Lett.*, 77(1):27–33, 2001. doi:10.1016/S0020-0190(00)00146-0.

10 Xuehou Tan, Tomio Hirata, and Yasuyoshi Inagaki. Corrigendum to "An Incremental Algorithm for Constructing Shortest Watchman Routes". *Int. J. Comput. Geom. Appl.*, 9(3):319–323, 1999. doi:10.1142/S0218195999000212.

11 Xuehou Tan and Bo Jiang. Efficient algorithms for touring a sequence of convex polygons and related problems. In *Proceedings of Theory and Applications of Models of Computation, TAMC 2017*, pages 614–627. Springer Verlag, Lecture Notes in Computer Science 10185, 2017. doi:10.1007/978-3-319-55911-7_44.

Strategy Repair in Reachability Games via a Graph Quotientation

Tiziana Calamoneri
Department of Computer Science, Sapienza University of Rome, Italy

Pierre Gaillard
Department of Computer Science, Sapienza University of Rome, Italy
LaBRI, Université de Bordeaux, France

Giacomo Paesani
Department of Computer Science, Sapienza University of Rome, Italy

Giuseppe Perelli
Department of Mathematics, Sapienza University of Rome, Italy

Abstract

Reachability Games over graphs (RGs) are a powerful modelling tool for synthesis and planning. The solutions to RGs are *strategies* that are used in program design. Sometimes, due to model deviation at execution time, specification updates, or simply a bug, the strategy provided as a solution to an RG no longer works. Strategy Repair aims to solve this problem by adjusting strategies with a minimum number of modifications. Such a minimisation requirement is motivated by the costs that one may incur when implementing the new strategy. To minimise implementation costs, one wants to reuse as much of the provided strategy as possible.

In the literature, Strategy Repair has been investigated from both theoretical and practical perspectives. First, it has been shown to be NP-complete. Second, two algorithmic approaches have been proposed to tackle the problem in practice, one provides an optimal solution, the other an approximated solution. Both approaches underutilise the graph-theoretical properties of games, which could significantly improve their performance and accuracy (in the case of approximation algorithms).

This paper provides a graph-theoretic characterisation of Strategy Repair that provably improves every algorithmic approach to solving the problem. It does so by introducing a new notion of *quotient* graph that allows us to identify and merge those vertices that are equivalent from the perspective of every solution to the problem. This way, solving Strategy Repair can be done in a reduced instance, which we call the quotient game. The approach not only reduces the problem's input size, but also improves the effectiveness of MustFix, an optimisation condition previously introduced for the problem. Besides the theoretical characterisation, we test our approach empirically by running experiments to demonstrate improvements over the quotient graph approach.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Paths and connectivity problems

Keywords and phrases Reachability Games, Strategy Repair, Strategic Reasoning, Automated Reasoning

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.12

Funding This paper was partially supported by *Sapienza* University of Rome (projects "Exploiting graph structure to cope with hard problems" [RM1241905ECC7E82], "Graphs as a model of complex real-life systems" [RM123188D7F7985D] and "ASGARD – Autonomous and Self-Governing Agent-Based Rule Design" [B83C23005800001]).

© Tiziana Calamoneri, Pierre Gaillard, Giacomo Paesani, and Giuseppe Perelli;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Reachability over graphs is a fundamental problem in Computer Science [6]. It serves as a modeling paradigm and as a solution technique for many aspects of Formal Methods, in particular for Automata Theory and Model Checking [2, 5, 15].

One of its basic formulations consists, for a given directed graph, to find the vertices from which there exists a path to a designated set of *target vertices*.

Such a setting can be used to model the behavior of an agent navigating within a *closed system*, that is, whose behavior is fully determined by the agent itself. However, in modern computing, we mostly deal with *open systems*, whose behavior is dependent on the interaction between the controlling agent and an external entity. To adapt the modeling to this scenario, reachability problems are generalized to Reachability Games (RGs) [11].

In an RG, some vertices in the directed graph are not controlled by the agent but by an external entity, *e.g.*, the environment or another agent, whose nature is adversarial, *i.e.*, whose aim is to prevent reaching a target vertex, if possible. In this scenario, it is not sufficient to find a path connecting the source vertex to the target, as the adversarial agent might deviate from the path through uncontrolled vertices. Therefore, solving a reachability game requires finding a *winning strategy*, that is, a function that prescribes to the agent the arc to traverse from each controlled vertex in such a way that, regardless of the adversarial decisions, a target vertex is guaranteed to be reached. This is a well-known and widely studied problem in Synthesis [3, 4, 8, 9, 16] and Planning [7, 13].

The solution to an RG, *i.e.*, a winning strategy σ_0 of the game, can be used to model the behavior of an agent acting in an open system. However, as noted in [12], at execution-time, models might deviate from the actual trajectory that stems from strategy execution. Also, there might be situations where the target vertices change to accommodate a modeling update. As a result, the strategy σ_0 computed as the solution to the RG might become inapplicable to the problem, leaving the agent unable to take appropriate actions to achieve its correct behavior. In other words, σ_0 is no longer winning for the game in place.

For this reason, the authors of [12] introduced *Strategy Repair* (SR), designed to address the problem of adjusting a given strategy to make it winning. Such an adjustment involves reconfiguring the strategy to select different arcs for the minimum number of vertices. This minimization requirement is motivated by the fact that every modification to the strategy incurs an implementation cost that must be kept limited. In addition to formally defining Strategy Repair in its basic form, the authors of [12] present theoretical and practical results related to it. First, they show that the problem is NP-complete *via* a reduction from Vertex Cover. Next, they introduce two algorithmic approaches. The first one provides an optimal solution to the problem and runs in time exponential in the input size. The second one provides an approximated solution and runs in time polynomial in the input size.

Despite extensive investigation of the problem from theoretical and practical standpoints, the work in [12] fails to fully exploit the graph-theoretical aspects of RGs and strategies. Their approach first identifies a set of vertices from which the repair algorithm must operate, then repairs the broken vertices, aiming to minimize the number of modifications.

Our paper contributes by working between these two steps. It does so by introducing a new notion of *quotient* over the set of vertices that depends on the strategy σ_0 that needs to be repaired. In doing so, it allows us to identify and collapse vertices that are *equivalent* from the perspective of Strategy Repair. As a result, we are able to introduce a *quotient game*, whose set of vertices is more compact compared to the original, but also equivalent in terms of solutions to the SR problem. Therefore, this allows us to solve an instance of SR by solving its quotient, whose size is reduced due to this compactification.

In addition to proving its theoretical properties, we conducted experiments comparing Strategy Repair algorithms with and without the quotient transformation, showing improvements in running time and accuracy for the approximated approaches.

As a by-product of this construction, we observed in the experiments that adopting the quotient game approach also improves the effect of MustFix, an optimization condition introduced in [12]. Therefore, we also briefly discuss this effect from a theoretical perspective.

Related Work

Games on graphs constitute the main reasoning methods for designing policies in Synthesis and Planning. Therefore, results on Strategy Repair are readily transferable to these fields. However, so far no notion of policy repair has emerged in the literature before [12]. The closest is the one of *plan repair* [14, 17, 19], concerning the problem of repairing a sequence of actions in a deterministic environment, i.e., a closed system where the outcome is fully determined by the agent's choices.

Additionally, a notion of Game Repair has been introduced in [1]. However, the setting is significantly different from the one analyzed in this paper. Indeed, the problem of Game Repair arises when, in a Multi-Agent Game, the winning conditions of the agents make it impossible to find a solution. Therefore, the task is to redesign the agent's objectives themselves.

2 Preliminaries

We here introduce the basic notions and definitions related to *reachability games*.

A *2-player arena*, or simply an *arena*, is a directed graph $\mathcal{A} = (V, \mathrm{E})$ whose vertex set is partitioned into the two sets V_0 and V_1, where V_0 and V_1 are the sets of vertices controlled by Player 0 and Player 1, respectively. Moreover, every vertex is assumed to have at least one outgoing arc. A *reachability game* (or simply *game*) $\mathcal{G} = \langle \mathcal{A}, T \rangle$ is a perfect-information 2-player turn-based game played on an arena $\mathcal{A}$, where the players are provided with a set $T \subseteq V$ of vertices, called *terminal* vertices. A token is initially placed on a vertex, and the player controlling the vertex hosting the token is asked to select an arc from that vertex and move the token along it. The aim of Player 0 is to bring the token to a vertex in T. Before formally specifying the objectives for the two players, we need to introduce additional notation.

A *play* of $\mathcal{A}$ is walk, *i.e.*, an infinite sequence of vertices $\pi = \pi_0\pi_1\pi_2\ldots$ such that $(\pi_i, \pi_{i+1}) \in \mathrm{E}$, for every $i \in \mathbb{N}$. A *partial play* of π is a finite subsequence $\pi_i \ldots \pi_j$, for some integers i and j with $i \leq j$.

A *strategy* (for Player 0) is a function $\sigma : V_0 \to \mathrm{E}$ mapping Player 0 vertices to arcs, such that, for each $v \in V_0$, $\sigma(v)$ is an arc outgoing from v. Intuitively, a strategy describes which arc Player 0 selects whenever it is its turn. A play π is a *consistent* with a strategy σ if $\sigma(\pi_i) = (\pi_i, \pi_{i+1})$ for each $\pi_i \in V_0$.

We define a very natural distance between two strategies σ and σ' over the same game, that is

$$\mathsf{dist}(\sigma, \sigma') = |\{v \in V_0 \mid \sigma(v) \neq \sigma'(v)\}|.$$

It is proven in [12] that dist is in fact a distance.

A strategy σ is called *winning from* v if every play starting from v consistent with σ reaches a vertex in T. We say that a vertex v is *winning* if there exists a strategy winning from v. By $\mathrm{Win}(\mathcal{G})$, we denote the set of winning vertices v. A strategy σ is *winning* if it is winning from all vertices in $\mathrm{Win}(\mathcal{G})$.

Notice that we use strategies known as *memoryless strategies*, but more general strategies can be found in the literature, where the next arc is chosen based on the current partial play rather than solely on the current vertex. This restriction is without loss of generality, since for every reachability game there exists a memoryless winning strategy, which is computable in polynomial time [10, 11, 18].

▶ **Definition 1** (Reachability problem). *The Reachability problem takes as input a game* $\mathcal{G} = \langle \mathcal{A}, T \rangle$ *and requires finding a winning strategy for Player 0.*

An example of a reachability game is depicted in Figure 1.

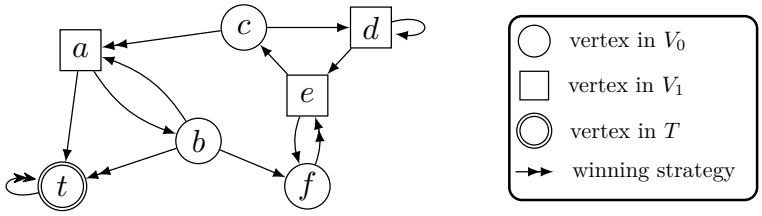


Figure 1 Example of a reachability game with target $T = \{t\}$ and $\mathrm{Win}(\mathcal{G}) = \{a, b, c, t\}$. Note that $a \in \mathrm{Win}(\mathcal{G})$ because it is a vertex of V_1 (so no arc of the strategy is expected to exit from it) and any arc coming out from a reaches a play consistent with the depicted winning strategy and leading to T; on the contrary, $e \notin \mathrm{Win}(\mathcal{G})$ because Player 1 can force the token to move only between e and f.

For a given game $\mathcal{G}$ and strategy σ, we define $\mathrm{Win}(\mathcal{G}, \sigma)$ as the set of vertices from which σ is winning for Player 0. Clearly, it always holds that $\mathrm{Win}(\mathcal{G}, \sigma) \subseteq \mathrm{Win}(\mathcal{G})$, with $\mathrm{Win}(\mathcal{G}, \sigma) = \mathrm{Win}(\mathcal{G})$ if and only if σ is winning for Player 0.

Given a game $\mathcal{G}$ and a (not necessarily winning) strategy σ_0, a strategy σ is a *closest winning strategy with respect to* σ_0, or simply *closest winning* when σ_0 is clear from the context, if it is winning in $\mathcal{G}$ and minimizes $\mathsf{dist}(\sigma_0, \sigma)$ among all winning strategies in $\mathcal{G}$.

We are finally ready to define the central problem of this work.

▶ **Definition 2** (Strategy Repair problem [12]). *The Strategy Repair problem takes as input a game* $\mathcal{G}$ *and a strategy* σ_0*, and requires finding a closest winning strategy with respect to* σ_0*.*

This problem requires minimizing the number of modifications that are required to turn a strategy σ_0 into a winning strategy σ for a given reachability game $\mathcal{G}$. The corresponding decision problem, instead, consists in fixing a given threshold $k \in \mathbb{N}$ and checking whether there exists a winning strategy σ with $\mathsf{dist}(\sigma_0, \sigma) \leq k$, and is NP-complete [12].

Note that reaching any vertex in $\mathrm{Win}(\mathcal{G}, \sigma_0)$ guarantees reaching T as well. So, given a reachability game $\mathcal{G} = \langle \mathcal{A}, T \rangle$, the aim of the Strategy Repair problem is to change σ_0 into another strategy σ which guarantees to reach any vertex in $\mathrm{Win}(\mathcal{G}, \sigma_0)$ from every other vertex in $\mathrm{Win}(\mathcal{G}) \setminus \mathrm{Win}(\mathcal{G}, \sigma_0)$. Therefore, the set $\mathrm{Win}(\mathcal{G}, \sigma_0)$ acquires the meaning of a new target set. For this reason, we denote from now on $\mathrm{Win}(\mathcal{G}, \sigma_0)$ by T'. Moreover, we define $\mathrm{Fr}(T')$ as the subset of arcs in the cut from $\mathrm{Win}(\mathcal{G}) \setminus T'$ to T' and coming out from a vertex in V_0. An arc in $\mathrm{Fr}(T')$ is called a *frontier arc* and the origin of a frontier arc is called a *frontier vertex*. Observe that for an arc $(u, v) \in \mathrm{Fr}(T')$, it holds that $\sigma_0(u) \neq (u, v)$ since $u \notin T'$.

3 Quotient Game

In this section, we introduce a new notion of quotient game of an instance $(\mathcal{G}, \sigma_0)$ of SR. We then proceed by proving that such a quotient identifies the vertices of $\mathcal{G}$ that are equivalent from the perspective of repairing σ_0. Such a result is summarized in the statement of Theorem 4, whose proof is obtained by the application of some lemmas introduced here. Finally, we discuss the effect of the quotient construction on MustFix: a selection condition introduced in [12] as an optimization for their algorithmic approach. We show that the quotient construction increases the number of times MustFix is triggered across all algorithmic approaches, thereby improving performance. We start by providing some useful notation and definitions.

Given a strategy σ, a vertex $u \in V_0$ and an arc (u, v) outgoing from u, we define the strategy $\sigma_{(u,v)}$ exactly the same as σ except in u, where $\sigma_{(u,v)}(u) = (u, v)$.

To iteratively repair σ_0 to a winning strategy, a frontier arc (u, v) at the time must be selected to replace $\sigma_0(u)$. Such replacement ensures that u enters the winning set of the modified strategy $\sigma_{0(u,v)}$. A key part of the repair process is deciding which frontier arc to select. The aim of the construction introduced in this work is to facilitate this selection.

When a frontier arc (u, v) is selected to replace $\sigma_0(u)$, the vertex u might not be the only one that becomes winning for the modified strategy, since some vertices from which every play consistent with σ_0 leads to either u or T' could exist. For this reason, we consider the *repair set of* u, denoted $\mathsf{Repair}(u)$ and defined as the set of vertices that indirectly become winning when σ_0 is modified, changing $\sigma_0(u)$ to a frontier arc. Note that if u is the origin of different frontier arcs, the same set $\mathsf{Repair}(u)$ is obtained regardless of which frontier arc is chosen. So given any frontier arc (u, v), we can formally define $\mathsf{Repair}(u) = \mathrm{Win}\left(\mathcal{G}, \sigma_{0(u,v)}\right) \setminus T'$.

We say that a frontier vertex u is a *base* if for every frontier vertex v, it does not hold that $\mathsf{Repair}(u) \subsetneq \mathsf{Repair}(v)$. In other words, u is a base if its repair set is maximal with respect to inclusion among the repair sets of frontier vertices. Two bases u and u' are equivalent if and only if $\mathsf{Repair}(u) = \mathsf{Repair}(u')$. Instead of referring to an equivalence class, we elect one base to represent the class itself. Let Bases be the set of elected bases. In the following, we denote by R the union of the repair sets of the bases, *i.e.*, $R = \cup_{u \in \mathsf{Bases}} \mathsf{Repair}(u)$.

Note that the repair sets of vertices in Bases are a partition of R. Indeed, suppose, by contradiction, there is a vertex $v \in \mathsf{Repair}(u) \cap \mathsf{Repair}(u')$, for two non-equivalent u and u' in Bases. It is impossible that u' lies on a partial play consistent with σ_0 from v to u (and vice versa), because in this case it would be $\mathsf{Repair}(u') \subseteq \mathsf{Repair}(u)$, so either u or u' would not be a base.

We now construct a reachability game $\mathcal{Q}(\mathcal{G}, \sigma_0)$ induced from $\mathcal{G}$ and σ_0, where each of these repair sets is represented by a single vertex. To avoid confusion, vertices of the quotient game are denoted by q_u, where u is a vertex of $\mathcal{G}$.

▶ **Definition 3** (Quotient game). *Let $\langle \mathcal{G}, \sigma_0 \rangle$ be an instance of Strategy Repair. The reachability game $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is a game $\langle \mathcal{A}^q, T^q \rangle$ with $\mathcal{A}^q = (V^q, \mathrm{E}^q)$ defined as follows:*

- *V^q is partitioned into ${V_0}^q$ and ${V_1}^q$ such that:*
  - ${V_0}^q = \{q_u \mid u \in V_0 \setminus R\} \cup \{q_u \mid u \in \mathsf{Bases}\}$;
  - ${V_1}^q = \{q_u \mid u \in V_1 \setminus R\}$;
- *E^q is the set of arcs of $\mathcal{Q}(\mathcal{G}, \sigma_0)$ obtained as follows. For each $(u, v) \in \mathrm{E}$ with $u \in (V \setminus R) \cup \mathsf{Bases}$, $(q_u, q_v) \in \mathrm{E}^q$ if $v \notin R$, and $(q_u, q_{v'}) \in \mathrm{E}^q$ otherwise, where v' is the elected base such that $v \in \mathsf{Repair}(v')$.*
- *$T^q = \{q_u \mid u \in T'\}$.*

Strictly speaking, $\mathcal{Q}(\mathcal{G}, \sigma_0)$ also depends on the elected bases: in particular, the frontier arcs of $\mathcal{Q}(\mathcal{G}, \sigma_0)$ may vary depending on which base we elect for each equivalence class. Nevertheless, note that this detail does not imply substantial changes to the execution or quality of the proposed approach.

For every $u \in V$, we say that q_u *represents* u in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ if $q_u \in V^q$ while q_v *represents* u if $v \in \mathsf{Bases}$ and $u \in \mathsf{Repair}(v)$.

A strategy σ in $\mathcal{G}$ is *compatible* with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ if it is equal to σ_0 when restricted to $\mathsf{Repair}(u) \setminus \{u\}$, for every base u, *i.e.*, $\sigma(v) = \sigma_0(v)$ for every $v \in V_0 \cap (\mathsf{Repair}(u) \setminus \{u\})$. Let σ be a strategy compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, we define the *projected strategy* σ^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ as follows: for each $q_u \in {V_0}^q$ with $\sigma(u) = (u, v)$, then $\sigma^q(q_u) = (q_u, q_{v'})$, where $q_{v'}$ represents v in $\mathcal{Q}(\mathcal{G}, \sigma_0)$.

Intuitively, if $\sigma(u) = (u, v)$ is an arc entering the set $\mathsf{Repair}(v')$, the projected strategy directly takes the arc $(q_u, q_{v'})$ since $q_{v'}$ represents the set $\mathsf{Repair}(v')$ which has been collapsed in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. We say that a strategy σ in $\mathcal{G}$ compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is an *elevation* of a strategy σ^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ if σ^q is the projection of σ. Note that an elevation of a strategy in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is not necessarily unique.

Figure 2 shows an example of the construction of a quotient game, where σ_0^q is the projected strategy of σ_0 and, reciprocally, σ_0 is an elevation of σ_0^q.

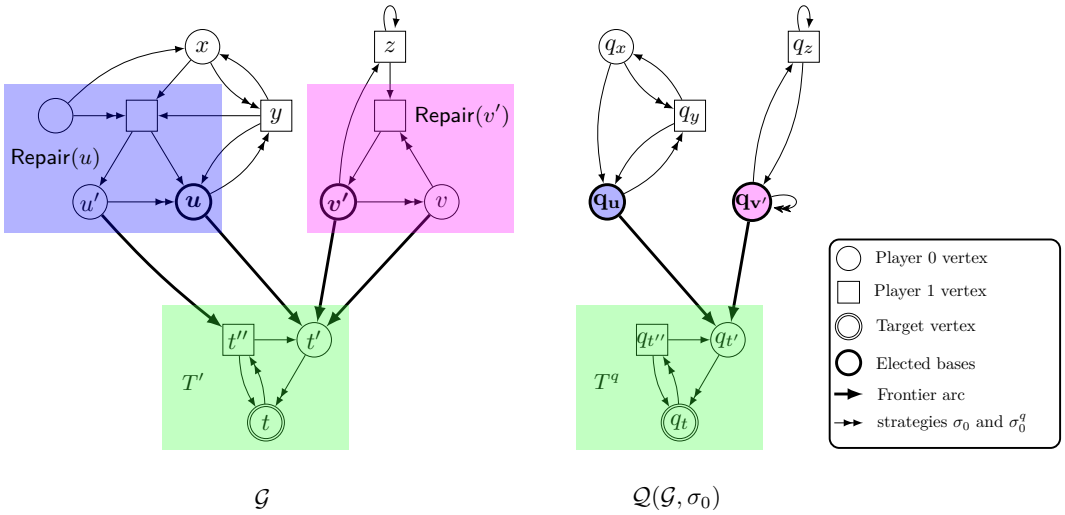


Figure 2 Construction of $\mathcal{Q}(\mathcal{G}, \sigma_0)$. $T' = \{t, t', t''\}$; the frontier vertices are u, u', v and v'; u, v, v' are bases; and $\mathsf{Bases} = \{u, v'\}$ is highlighted by thick vertices. Note v and v' are equivalent bases. In $\mathcal{Q}(\mathcal{G}, \sigma_0)$, $\mathsf{Repair}(u)$ is represented by q_u and $\mathsf{Repair}(v')$ by $q_{v'}$. Moreover, there is a loop on $q_{v'}$ because $\sigma_0(v') = (v', v)$ and $v \in \mathsf{Repair}(v')$.

Similarly to what has been done with strategies, we now define the projection and elevation for plays.

We say that a play π in $\mathcal{G}$ is *compatible* with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ if it is consistent with σ_0 within the set $\mathsf{Repair}(u) \setminus \{u\}$, for every base u. Let π be a play compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, we define the *projected play* π^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ obtained from π as follows: for every integer i, the partial play $\pi_i \pi_{i+1}$ is replaced by

- $q_u q_v$, where q_u represents π_i and q_v represents π_{i+1}, if $u \neq v$;
- nothing, otherwise.

Moreover, an *elevation* of a play π^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is a play π compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ such that the projection of π is π^q. An example of all these definitions is shown in Figure 3.

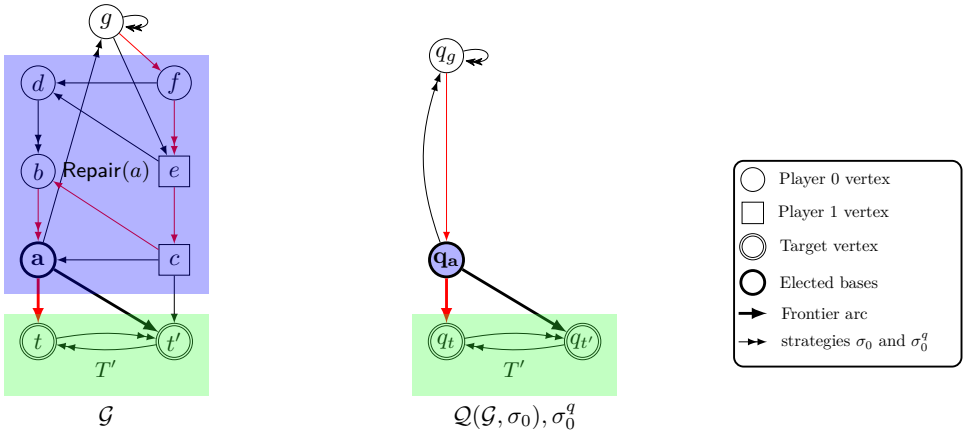


Figure 3 A path π, marked in red, in $\mathcal{G}$ compatible with $\mathcal{Q}(\mathcal{G},\sigma_0)$, and the corresponding path π^q, again marked in red, in $\mathcal{Q}(\mathcal{G},\sigma_0)$. π^q is the projection of π in $\mathcal{Q}(\mathcal{G},\sigma_0)$, and π is one of the possible elevations of π^q in $\mathcal{G}$.

We can now state the main result of this section.

▶ **Theorem 4.** *Given a game $\mathcal{G}$ and a strategy σ_0, let σ^q be a closest winning strategy with respect to σ_0^q in $\mathcal{Q}(\mathcal{G},\sigma_0)$. Any elevation σ of σ^q in $\mathcal{G}$ is a closest winning strategy with respect to σ_0.*

As a consequence of Theorem 4, Strategy Repair on input $(\mathcal{G},\sigma_0)$ can be solved in the following way: first, we construct $\mathcal{Q}(\mathcal{G},\sigma_0)$, then we solve Strategy Repair on input $(\mathcal{Q}(\mathcal{G},\sigma_0),\sigma_0^q)$, and finally return an elevation of the previously computed solution.

The rest of the section is devoted to introducing auxiliary results useful for proving Theorem 4. We begin with the following lemma, which plays a key role in showing that moving from $(\mathcal{G},\sigma_0)$ to the induced quotient game preserves solutions.

▶ **Lemma 5.** *Given a game $\mathcal{G}$ and a strategy σ_0, there exists a closest winning strategy σ_* with respect to σ_0 that is compatible with $\mathcal{Q}(\mathcal{G},\sigma_0)$.*

Proof. Let σ be a closest winning strategy with respect to σ_0. We define a strategy σ_* as follows. For each $u \in \mathsf{Bases}$, if there exists $v \in V_0 \cap (\mathsf{Repair}(u) \setminus \{u\})$ such that $\sigma(v) \neq \sigma_0(v)$, then for each $v \in V_0 \cap (\mathsf{Repair}(u) \setminus \{u\})$ such that $\sigma(v) \neq \sigma_0(v)$ set $\sigma_*(v) = \sigma_0(v)$ and pick $\sigma_*(u) \in \mathrm{Fr}(T')$ which is always possible since u is a frontier vertex. For any other vertex $u \in V_0$, set $\sigma_*(u) = \sigma(u)$. By construction, it follows that σ_* is compatible with $\mathcal{Q}(\mathcal{G},\sigma_0)$.

Now, we show that the strategy σ_* is winning. Let $v \in \mathrm{Win}(\mathcal{G})$ and consider any play π from v consistent with σ_*. If π is also consistent with σ, then it reaches a vertex in T since σ is a winning strategy. Otherwise, let i be the smallest integer such that $\sigma_*(\pi_i) \neq \sigma(\pi_i)$. By construction of σ_*, there is a vertex $u \in \mathsf{Bases}$ such that $\pi_i \in \mathsf{Repair}(u)$, and then π reaches T', either through the frontier arc $\sigma_*(u)$ or through an arc outgoing from a vertex of $V_1 \cap \mathsf{Repair}(u)$, because σ_* and σ_0 are equal in $\mathsf{Repair}(u) \setminus \{u\}$. Once π is in T', notice that σ_* and σ are equal in this set by construction, and σ must be equal to σ_0 in this set, otherwise it would not be a closest winning strategy. Therefore, π is consistent with σ_0, which ensures to reach the target from $T' = \mathrm{Win}(\mathcal{G},\sigma_0)$. This shows that $v \in \mathrm{Win}(\mathcal{G},\sigma_*)$ and therefore that $\mathrm{Win}(\mathcal{G}) \subseteq \mathrm{Win}(\mathcal{G},\sigma_*)$, proving that σ_* is a winning strategy.

Removing the differences between σ and σ_0 in σ_* allows us to state that: if $\sigma(v) = \sigma_0(v)$ for every $v \in \mathsf{Repair}(u) \setminus \{u\}$, then $\mathsf{dist}(\sigma_0,\sigma_*) = \mathsf{dist}(\sigma_0,\sigma)$ within $\mathsf{Repair}(u)$. Otherwise, *i.e.*, when there exists $v \in \mathsf{Repair}(u) \setminus \{u\}$ such that $\sigma(v) \neq \sigma_0(v)$, $\mathsf{dist}(\sigma_0,\sigma_*) \leq \mathsf{dist}(\sigma_0,\sigma) - 1$

inside $\mathsf{Repair}(u) \setminus \{u\}$ for every $u \in \mathsf{Bases}$, while setting $\sigma_*(u)$ to a frontier arc increases the distance by exactly one. Hence, $\mathsf{dist}(\sigma_0, \sigma_*) \leq \mathsf{dist}(\sigma_0, \sigma)$ in $\mathcal{G}$, so σ_* is a closest winning strategy with respect to σ_0. ◀

The following lemmas establish results that facilitate the transfer of plays, strategies, winning regions, winning strategies, and distances between games and their quotient games.

▶ **Lemma 6.** *Given a game $\mathcal{G}$ and a strategy σ_0, if π is a play consistent with a strategy σ that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, then π^q is consistent with σ^q.*

Proof. The play π is consistent with σ which is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, hence π is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ and the projection π^q is therefore well defined.

Let i be any integer such that $\pi^q_i \in V^q_0$, and let $u \in V_0$ such that $q_u = \pi^q_i$. Since u is a vertex of π, there exists an integer j such that $u = \pi_j$. Since π is consistent with σ and $\pi_j \in V_0$, $\sigma(\pi_j) = (\pi_j, \pi_{j+1})$.

Note that π_{j+1} is not represented by q_{π_j} since $\pi_j \notin R$ or $\pi_j \in \mathsf{Bases}$. Denote by q_v the vertex of V^q_0 that represents π_{j+1}. Thus, we have that $\sigma^q(\pi^q_i) = \sigma^q(q_{\pi_j}) = (q_{\pi_j}, q_v) = (\pi^q_i, q_v)$. Moreover, since $\pi_j\pi_{j+1}$ is a partial play of π, then $\pi^q_i q_v$ is a partial play of π^q. This means that $\pi^q_{i+1} = q_v$, and so $\sigma^q(\pi^q_i) = (\pi^q_i, \pi^q_{i+1})$. ◀

▶ **Lemma 7.** *Given a game $\mathcal{G}$ and a strategy σ_0, let σ^q be a strategy in $\mathcal{Q}(\mathcal{G}, \sigma_0)$ and σ be an elevation of σ^q. For a vertex $u \in V$, denote by q_v the representative of u in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. If π^q is a play starting from q_v that is consistent with σ^q, then there exists an elevation π of π^q starting from u that is consistent with σ.*

Proof. First, we iteratively construct a play π starting from the partial play of length 0 u as follows. If $u \neq v$, then $u \in \mathsf{Repair}(v)$ and π contains any partial play from u to v within $\mathsf{Repair}(v)$ that is consistent with σ_0. Otherwise, we set $\pi_0 = u$.

Now, for every integer i, we construct partial plays in $\mathcal{G}$ corresponding to the partial play $\pi^q_i\pi^q_{i+1}$ in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Denote by x and y the vertices of V such that $q_x = \pi^q_i$ and $q_y = \pi^q_{i+1}$. If $y \notin R$, then π contains the partial play xy. Suppose now $y \in R$.

We distinguish two more cases depending on which player controls x. If $x \in V_0$, then say $\sigma(x) = (x, z)$, it holds that $z \in \mathsf{Repair}(y)$ since σ is an elevation σ^q and $\sigma^q(q_x) = (q_x, q_y)$. So π contains the partial play xz and any partial play from z to y within $\mathsf{Repair}(y)$ that is consistent with σ_0. Otherwise, that is, if $x \in V_1$, then there exists an arc (x, z), for some $z \in \mathsf{Repair}(y)$, and π contains the partial play xz and any partial play from z to y within $\mathsf{Repair}(y)$ that is consistent with σ_0.

Now, we prove π is an elevation of π^q. π is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ because all partial plays of π within $\mathsf{Repair}(u') \setminus \{u'\}$, for some $u' \in \mathsf{Bases}$, are plays consistent with σ_0 by construction. Moreover, the projection of π is exactly π^q since some specific partial plays of π mimic the arcs of π^q.

We conclude the proof by showing that π is consistent with σ. Let a and b be two consecutive vertices in π, with $a \in V_0$. If $a \in R \setminus \mathsf{Bases}$, then, since π is consistent with σ_0 within every repair set by construction, with the possible exception of the elected base, $\sigma_0(a) = (a, b)$. Finally, suppose $a \notin R$ or $a \in \mathsf{Bases}$. Then by construction $\sigma(a) = (a, b)$. ◀

▶ **Lemma 8.** *Given a game $\mathcal{G}$ and a strategy σ_0, the winning vertices of the quotient game $\mathcal{Q}(\mathcal{G}, \sigma_0)$ are precisely the representatives of the winning vertices in $\mathcal{G}$.*

Proof. Let $u \in \mathrm{Win}(\mathcal{G})$ and denote by q_v the representative of u in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Note that it is possible that $v = u$. By Lemma 5, there exists a closest winning strategy σ_* with respect to σ_0, that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$. In particular, $u \in \mathrm{Win}(\mathcal{G}, \sigma_*)$: every play starting

from u and consistent with σ_* eventually reaches T'. Let π^q be any play starting from q_v that is consistent with σ^q_*. Let π be an elevation of π^q starting from u and consistent with σ, obtained from the application of Lemma 7. Let i be the minimum integer such that $\pi_i \in T'$. This means that π^q starts from q_v and reaches q_{π_i}, which belongs to T^q since $\pi_i \in T'$, and so the target. This proves that every representative of a winning vertex in $\mathcal{G}$ is winning in $\mathcal{Q}(\mathcal{G}, \sigma_0)$.

Let $q_u \in \mathrm{Win}(\mathcal{Q}(\mathcal{G}, \sigma_0))$. Consider a winning strategy σ^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. In particular, $q_u \in \mathrm{Win}(\mathcal{Q}(\mathcal{G}, \sigma_0), \sigma^q)$: every play starting from q_u and consistent with σ^q eventually reaches T^q. We show that, for every v represented by q_u, $v \in \mathrm{Win}(\mathcal{G})$. Let σ be any elevation of σ^q and π be any play consistent with σ starting from v. By Lemma 6, π^q is consistent with σ^q. Since π^q is a play starting from q_u that eventually reaches T^q, there exists an integer i such that $\pi^q_i \in T^q$. Let z be the vertex of V represented by π^q_i. Thus there is an integer j such that $\pi_j = z$, which belongs to T' since $q_z \in T^q$. This proves that the winning region of $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is made of vertices of the quotient game representing vertices of $\mathcal{G}$ that are winning. ◀

▶ **Lemma 9.** *Given a game $\mathcal{G}$ and a strategy σ_0. If a strategy σ of $\mathcal{G}$ that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ is winning, then σ^q is winning in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Moreover, if a strategy σ^q of $\mathcal{Q}$ is winning, then any elevation σ of σ^q is winning.*

Proof. Let σ be a winning strategy of $\mathcal{G}$ that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Let $q_u \in \mathrm{Win}(\mathcal{Q}(\mathcal{G}, \sigma_0))$. By Lemma 8, q_u represents a set of vertices of $\mathrm{Win}(\mathcal{G})$. Let π^q be any play starting from q_u consistent with σ^q. Let π be an elevation of π^q starting from u and consistent with σ, obtained from the application of Lemma 7. Let i be the minimum integer such that $\pi_i \in T'$. If $i = 0$, then $q_u = \pi^q_0 \in T^q$. So from now on, we can assume $i \geq 1$. In particular, $\pi_{i-1} \notin T'$. Thus, by definition, π^q contains the partial play $q_w q_{\pi_i}$, where q_w represents π_{i-1}, since $w \neq \pi_i$. This means that π^q starts from q_u and reaches the target. This proves that σ^q is winning.

Let σ^q be a winning strategy of $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Let $u \in \mathrm{Win}(\mathcal{G})$. Let π be any play starting from u that is consistent with σ. By Lemma 6, π^q is a play consistent σ^q. Since π^q is a play starting from q_v that eventually reaches T^q, where q_v represents u in $\mathcal{Q}(\mathcal{G}, \sigma_0)$, there exists an integer i such that $\pi^q_i \in T^q$. If $i = 0$, then $v = \pi_0 \in T'$.

So from now on, we can assume $i \geq 1$. In particular, $\pi^q_{i-1} \notin T^q$. Denote with q_w and q_z the vertices of V^q that represent π^q_{i-1} and π^q_i, respectively. Clearly $w \neq z$. Thus, there is an integer j such that π_j is represented by q_w and π_{j+1} is represented by q_z. Then $\pi_{j+1} \in T'$. This means that σ is winning from u. Thus, σ is winning in $\mathcal{G}$. ◀

▶ **Lemma 10.** *Given a game $\mathcal{G}$ and a strategy σ_0, if a strategy σ is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, then $\mathsf{dist}_{\mathcal{Q}}(\sigma^q, \sigma^q_0) \leq \mathsf{dist}_{\mathcal{G}}(\sigma, \sigma_0)$ holds. Moreover, if a strategy σ is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ and is also a closest winning strategy with respect to σ_0, then $\mathsf{dist}_{\mathcal{Q}}(\sigma^q, \sigma^q_0) = \mathsf{dist}_{\mathcal{G}}(\sigma, \sigma_0)$.*

Proof. Let σ be a strategy that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Let $u \in V_0$, if $u \in R \setminus \mathsf{Bases}$, then $\sigma_0(u) = \sigma(u)$, since σ is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Otherwise, $q_u \in {V_0}^q$. Assume $\sigma_0(u) = \sigma(u) = (u, v)$ for some $v \in V$. If $v \notin R$, then $\sigma^q_0(q_u) = \sigma^q(q_u) = (q_u, q_v)$. Otherwise, that is, if $v \in R$, then $u \in \mathsf{Bases}$ and $v \in \mathsf{Repair}(u)$. Indeed, since $\sigma_0(u) = (u, v)$, then $u \in \mathsf{Repair}(v')$, where $v' \in \mathsf{Bases}$ and $v \in \mathsf{Repair}(v')$ and so $u \in R$. Moreover, $u \in \mathsf{Bases}$ because $q_u \in V^q_0$. Finally, $v \in \mathsf{Repair}(u)$ since otherwise u would not be in Bases. In that case, $\sigma^q_0(u) = \sigma^q(u) = (q_u, q_u)$. This proves that $\sigma_0(u) = \sigma(u)$ implies that $\sigma^q_0(u) = \sigma^q(u)$. It follows that $\mathsf{dist}_{\mathcal{Q}}(\sigma^q, \sigma^q_0) \leq \mathsf{dist}_{\mathcal{G}}(\sigma, \sigma_0)$.

Assume now that σ is also a closest winning strategy with respect to σ_0. Let $q_u \in {V_0}^q$, such that $\sigma_0^q(q_u) = \sigma^q(q_u) = (q_u, q_v)$ for some $q_v \in V^q$. If $v \in V \setminus R$, then $\sigma_0(u) = \sigma(u) = (u, v)$. For the rest of this proof we assume $v \in \mathsf{Bases}$. Let $\sigma_0(u) = (u, w)$ for some $w \in \mathsf{Repair}(v)$. Then $u \in \mathsf{Repair}(v)$, so $u \in R$ and therefore $u \in \mathsf{Bases}$ since $q_u \in {V_0}^q$. Hence $v = u$ and thus $\sigma_0^q(q_u) = \sigma^q(q_u) = (q_u, q_u)$. Since σ is a winning strategy for $\mathcal{G}$ that is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$, by Lemma 9, σ^q is winning in $\mathcal{Q}(\mathcal{G}, \sigma_0)$, which contradicts the fact that $\sigma^q(q_u) = (q_u, q_u)$. It follows that $\mathsf{dist}_{\mathcal{G}}(\sigma, \sigma_0) = \mathsf{dist}_{\mathcal{Q}}(\sigma^q, \sigma_0^q)$. ◀

We are finally ready to prove the main result of this section.

▶ **Theorem 4.** *Given a game $\mathcal{G}$ and a strategy σ_0, let σ^q be a closest winning strategy with respect to σ_0^q in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Any elevation σ of σ^q in $\mathcal{G}$ is a closest winning strategy with respect to σ_0.*

Proof. Let σ be any elevation of σ^q in $\mathcal{G}$. By Lemma 9 applied to σ^q, σ is winning. Now, we show that σ is a closest winning strategy with respect to σ_0. Let σ_* be the strategy, obtained by Lemma 5, that is a closest winning strategy with respect to σ_0 and compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$. By Lemma 9 applied to σ_*, the strategy σ_*^q is winning in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. Since σ^q is a closest winning strategy with respect to σ_0^q, it holds that $\mathsf{dist}_{\mathcal{Q}(\mathcal{G},\sigma_0)}(\sigma^q, \sigma_0^q) \leq \mathsf{dist}_{\mathcal{Q}(\mathcal{G},\sigma_0)}(\sigma_*^q, \sigma_0^q)$. By Lemma 5, σ_* is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ and it holds $\mathsf{dist}_{\mathcal{G}}(\sigma_*, \sigma_0) = \mathsf{dist}_{\mathcal{Q}(\mathcal{G},\sigma_0)}(\sigma_*^q, {\sigma_0}^q)$, by Lemma 10 applied to σ_*. By the definition of an elevation, it is also true that σ is compatible with $\mathcal{Q}(\mathcal{G}, \sigma_0)$ and thus $\mathsf{dist}_{\mathcal{Q}(\mathcal{G},\sigma_0)}(\sigma^q, \sigma_0^q) = \mathsf{dist}_{\mathcal{Q}(\mathcal{G},\sigma_0)}(\sigma, \sigma_0)$, by Lemma 10 applied to σ. Therefore, we obtain $\mathsf{dist}_{\mathcal{G}}(\sigma_0, \sigma) \leq \mathsf{dist}_{\mathcal{G}}(\sigma_0, \sigma_*)$, which implies that σ is a closest winning strategy with respect to σ_0. ◀

The MustFix condition on quotient games

Let $\mathcal{G}$ be a reachability game. Given an arc $e = (v_1, v_2) \in \mathrm{E}$, we denote by $\mathcal{G}_e$ the game induced from $\mathcal{G}$ by removing every arc (v_1, v_2') with $v_2' \neq v_2$. Therefore given a vertex $u \in V_0$, $\mathcal{G}_{\sigma_0(u)}$ is the game defined as $\mathcal{G}$ where Player 0 is forced to follow $\sigma_0(u)$ from u.

Given $\mathcal{G}$ and σ_0, we say that a frontier vertex u satisfies the MustFix condition if $u \notin \mathrm{Win}\left(\mathcal{G}_{\sigma_0(u)}\right)$, *i.e.*, if Player 0 cannot win from u with any strategy that maps u to $\sigma_0(u)$. In other words, the choice $\sigma_0(u)$ must be necessarily modified. It is known [12] that individuating such vertices improves the running time of algorithms and the accuracy of heuristics-based algorithms. Notice that if u satisfies the MustFix condition, then u is a base. Indeed, if there existed another vertex u' such that $u \in \mathsf{Repair}(u')$, then keeping the choice $\sigma_0(u)$ and taking the frontier arc from u' would yield a winning play from u, contradicting the fact that u satisfies the MustFix condition. For the same reason, no other base u' equivalent to u exists, since this would again imply $u \in \mathsf{Repair}(u')$ and lead to the same contradiction. Therefore, for such a vertex u, q_u is a well defined vertex in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. We now prove that if a vertex satisfies the MustFix condition in $\mathcal{G}$ then its representative also satisfies it in the game $\mathcal{Q}(\mathcal{G}, \sigma_0)$.

▶ **Proposition 11.** *Given a game $\mathcal{G}$ and a strategy σ_0, let $u \in V$. If $(u, v) \in \mathrm{Fr}(\mathrm{Win}\,(\mathcal{G}, \sigma_0))$ and $u \notin \mathrm{Win}\left(\mathcal{G}_{\sigma_0(u)}\right)$, then $q_u \notin \mathrm{Win}\left(\mathcal{Q}(\mathcal{G}, \sigma_0)_{\sigma_0^q(q_u)}\right)$.*

Proof. Let u be a frontier vertex such that $u \notin \mathrm{Win}\left(\mathcal{G}_{\sigma_0(u)}\right)$. Thanks to the previous remark, we know that q_u is a well-defined vertex in $\mathcal{Q}(\mathcal{G}, \sigma_0)$. If $q_u \in \mathrm{Win}\left(\mathcal{Q}(\mathcal{G}, \sigma_0)_{\sigma_0^q(q_u)}\right)$, then Player 0 has a strategy σ^q for $\mathcal{Q}(\mathcal{G}, \sigma_0)$ winning from q_u and such that $\sigma^q(q_u) = \sigma_0^q(q_u)$. An elevation of σ^q can be used in $\mathcal{G}_{\sigma_0(u)}$, and is also winning in $\mathcal{G}_{\sigma_0(u)}$ from u by Lemma 9, which contradicts $u \notin \mathrm{Win}\left(\mathcal{G}_{\sigma_0(u)}\right)$. ◀

In addition to preserving the occurrences of vertices satisfying the MustFix condition, the construction of the game $\mathcal{Q}(\mathcal{G}, \sigma_0)$ may cause vertices to satisfy this condition even though no vertex of $\mathcal{G}$ satisfies it. This is fundamental for reducing the running time of the exact algorithm and improving the accuracy of a polynomial-time algorithm.

In the example of Figure 4, the frontier vertices of $\mathcal{G}$ are a, b, c and d, yet none of them satisfies the MustFix condition. Indeed, $\sigma_{0(b,t)}$ is winning from a in $\mathcal{G}_{\sigma_0(a)}$, and $\sigma_{0(a,t),(c,t)}$ is winning from b in $\mathcal{G}_{\sigma_0(b)}$. Hence, neither a nor b satisfies the MustFix condition. The vertices c and d do not satisfy the MustFix condition for symmetric reasons. However, the game $\mathcal{Q}(\mathcal{G}, \sigma_0)$ has two frontier vertices, q_b and q_c, both of which now satisfy the MustFix condition. Indeed, q_b is not winning for any strategy σ^q such that $\sigma^q(q_b) = (q_b, q_e)$. Therefore, q_b satisfies the MustFix condition, and so does q_c by symmetry.

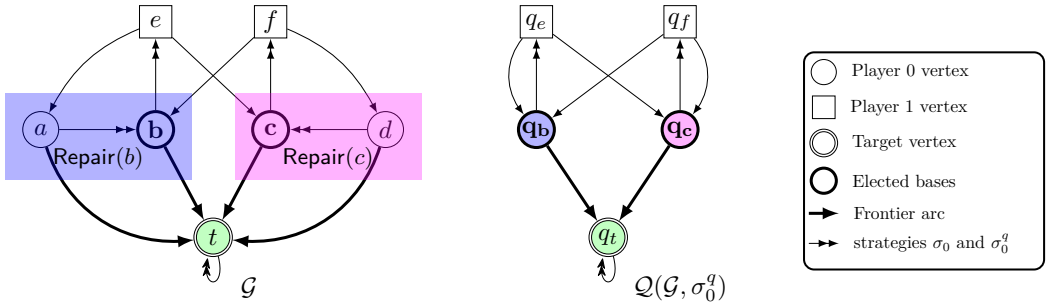


Figure 4 An example of construction of $\mathcal{Q}(\mathcal{G}, \sigma_0)$ where some vertices satisfying the MustFix condition are introduced.

4 Experiments

In this section, we show that our new quotient game concept can substantially improve the performance of known algorithms for the strategy repair problem, in terms of both running time and accuracy (*i.e.*, the ratio of the cardinalities of the approximate and exact solutions). The algorithms we consider are the (exponential time) exact algorithm, $\mathcal{A}_{ex}$, and the heuristic $\mathcal{A}_h$, described in [12], which are the only known in the literature, to the best of our knowledge. Clearly, we need to modify the algorithms so they can operate on the quotient graph. In particular, the quotient graph must be recomputed each time the current solution is updated, but this does not affect the correctness of the resulting solutions, thanks to Theorem 4. In the following, we use $\mathcal{A}_{ex}^{\mathcal{G}}$, $\mathcal{A}_h^{\mathcal{G}}$, $\mathcal{A}_{ex}^{\mathcal{Q}}$, and $\mathcal{A}_h^{\mathcal{Q}}$ to refer to the original algorithms for the input strategy repair problem and their modifications for the quotient game.

All tests have been run on an Intel Core i7-8700 CPU @ 3.20GHz, with 32GB of RAM and running Ubuntu 20.04.06 LTS.

For all the sets of experiments, we run the algorithms on many values of $|V|$ (reported in the first column of each table) and, for each fixed value of $|V|$, we generated N instances (N is set to 100 by default) either as random instances, or as instances deduced by vertex-cover instances.

Namely, to generate a *random instance* of a given size, each vertex is first assigned to either V_0 or V_1 with equal probability (0.5 each). In addition, each vertex is independently included in the target set T with probability 0.05, yielding an expected target size of 5% of $|V|$. For each vertex, its out-degree d is uniformly drawn, and d out-neighbors are then selected uniformly at random. We select d either between 1% and 5% of $|V|$, generating *sparse random instances*, or between 40% and 50% of $|V|$, generating *dense random instances*.

The initial strategy σ_0 to be repaired is constructed as follows. For each vertex $v \in V_0$, if v has at least one out-neighbor that does not belong to the target T, one such out-neighbor is chosen uniformly. Otherwise, an arbitrary out-neighbor is selected. This construction of σ_0 increases the chances that a number of arcs in σ_0 will require repair.

For the *vertex-cover instances*, an undirected graph with $|V|$ vertices and edge density of parameter p is generated as follows: for each pair of vertices $\{i, j\}$, the edge $\{i, j\}$ is included independently with probability p. The corresponding strategy repair instance is then obtained via the reduction described in [12]. We conducted two series of experiments, one with $p = \frac{2}{|V|}$ and one with $p = 0.1$, yielding instances referred as *sparse* and *dense vertex-cover instances*, respectively.

We introduced a timeout, set to 15 minutes by default. In all tables, the "No. of timeouts" column records the number of instances that were interrupted by a timeout. Moreover, in all tables, the running times of the algorithms executed on the quotient games also include the time necessary to pass from $\mathcal{G}$ to $\mathcal{Q}$.

We performed a first set of experiments on random instances, comparing the running times of the exact algorithm in its two versions $\mathcal{A}_{ex}^{\mathcal{G}}$ and $\mathcal{A}_{ex}^{\mathcal{Q}}$; the results are listed in Table 1. It is quite clear that introducing the quotient approach approximately halves the running time. It is worth noting that, when $|V| = 100$ and $|V| = 200$, the average time is much higher for $\mathcal{A}_{ex}^{\mathcal{G}}$ than for $\mathcal{A}_{ex}^{\mathcal{Q}}$; this particular behavior can be explained with the presence of some particularly time-consuming instances: there are two instances with $|V| = 100$ and running times of approximately 38 and 115 seconds, respectively, and two instances with $|V| = 200$ whose the running time of the first one is approximately 56 seconds while the second one does not output a solution because it exceeds the timeout.

Table 1 Comparison of running times of $\mathcal{A}_{ex}^{\mathcal{G}}$ and $\mathcal{A}_{ex}^{\mathcal{Q}}$ in sparse random instances.

	Time (sec)	
$\|V\|$	$\mathcal{A}_{ex}^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$
100	1.7	1.4e−2
200	0.67	2.0e−2
300	6.3e−2	5.5e−2
400	0.14	0.12
500	0.64	0.23
600	1.2	0.41
700	1.4	0.67
800	2.0	0.98
900	2.9	1.4
1000	3.7	1.8
2000	64.6	14.1
3000	316.6	56.7

In a second set of experiments, we also introduced the heuristic to compare the solution's accuracy, in addition to running times; the results are listed in Table 2. While the running times of $\mathcal{A}_{ex}^{\mathcal{Q}}$ and $\mathcal{A}_{h}^{\mathcal{Q}}$ are approximately the same, again they are about halved when running $\mathcal{A}_{h}^{\mathcal{Q}}$ with respect to $\mathcal{A}_{h}^{\mathcal{G}}$ (see second column of Table 2). For this set of experiments, we used a timeout of 15 minutes, which was reached only by $\mathcal{A}_{h}^{\mathcal{G}}$ on instances of size 4000; these instances therefore require the use of the quotient approach.

The third column records the number of times a MustFix condition is exploited, the fourth column represents the distance of the found solution from σ_0, and the fifth column relates the optimal solution (found by $\mathcal{A}_{ex}$) with the solutions found by the heuristic. The accuracy is,

in any case, extremely high, and it tends to increase while $|V|$ grows; the reason is that, when the graph is larger, there are more candidates to find a frontier vertex satisfying the MustFix condition, in which case we are sure not to make an unnecessary modification. We can see this because the number of vertices satisfying the MustFix condition approaches the distance obtained. We observe that $\mathcal{A}_h^{\mathcal{G}}$ takes generally a longer time and finds vertices satisfying the MustFix condition slightly less frequently, inducing sometimes a negligible increase in the distance from the optimal solution.

Table 2 Comparison of $\mathcal{A}_{ex}^{\mathcal{Q}}$, $\mathcal{A}_h^{\mathcal{Q}}$ and $\mathcal{A}_h^{\mathcal{G}}$ on sparse random instances.

		Time (sec)			No. MustFix			Distance			Accuracy	
$\|V\|$	N	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$
100	1000	8.9e−3	2.4e−3	1.6e−3	13.99	13.98	13.47	14.19	14.45	14.45	0.9842	0.9820
200	1000	0.44	2.1e−2	1.7e−2	42.61	42.60	42.23	42.64	42.67	42.70	0.9994	0.9986
300	1000	7.1e−2	7.1e−2	6.3e−2	69.12	69.12	68.86	69.12	69.12	69.13	1	0.9998
400	1000	0.16	0.16	0.17	93.94	93.94	93.73	93.93	93.93	93.94	1	0.9998
500	1000	0.28	0.28	0.40	118.9	118.9	118.7	118.9	118.9	118.9	1	0.9999
600	1000	0.49	0.49	0.83	142.4	142.4	142.2	142.4	142.4	142.4	1	1
700	1000	0.80	0.75	1.5	166.4	166.4	166.2	166.4	166.4	166.4	1	1
800	1000	1.1	1.0	2.3	190.3	190.3	190.1	190.3	190.3	190.3	1	1
900	1000	1.6	1.5	2.8	241.2	214.2	214.0	214.2	214.2	214.2	1	1
1000	1000	2.1	2.0	4.3	237.8	237.8	237.6	237.8	237.8	237.8	1	1
1500	5	7.0	6.8	21.3	363.2	363.2	363.8	363.2	363.2	363.2	1	1
2000	5	14.8	14.8	77.7	458.6	458.6	458.6	458.6	458.6	458.6	1	1
2500	5	38.4	33.5	201	593.8	593.8	593.6	593.8	593.8	593.8	1	1
3000	5	71.3	58.3	311	723.0	723.0	723.0	723.0	723.0	723.0	1	1
4000	5	140	128	TO	933.8	933.8	-	933.8	933.8	-	1	-

The next set of experiments is the same as the previous one, but takes as input dense random instances. In this case, we obtain very similar results to the ones for sparse random instances in Table 2, except that the running times are higher: again, the running times are almost the same for $\mathcal{A}_{ex}^{\mathcal{Q}}$ and $\mathcal{A}_h^{\mathcal{Q}}$, and longer for $\mathcal{A}_h^{\mathcal{G}}$, and the accuracy is 1 for both polynomial algorithms. The results are presented in Table 3.

Table 3 Comparison of $\mathcal{A}_{ex}^{\mathcal{Q}}$, $\mathcal{A}_h^{\mathcal{Q}}$ and $\mathcal{A}_h^{\mathcal{G}}$ on dense random instances.

		Time (sec)			No. MustFix			Distance			Accuracy	
$\|V\|$	N	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$	$\mathcal{A}_h^{\mathcal{Q}}$	$\mathcal{A}_h^{\mathcal{G}}$
100	100	1.8e−2	1.8e−2	1.6e−2	24.06	24.06	23.85	24.06	24.06	24.06	1	1
200	100	0.10	0.10	0.16	47.83	47.83	47.65	47.83	47.83	47.83	1	1
300	100	0.28	0.28	0.65	71.09	71.09	70.92	71.09	71.09	71.09	1	1
400	100	0.59	0.60	1.8	96.07	96.07	95.84	96.07	96.07	96.07	1	1
500	100	1.0	1.0	4.4	118.5	118.5	118.3	118.5	118.5	118.5	1	1
600	100	1.7	1.8	9.3	144.2	144.2	144.0	144.2	144.2	144.2	1	1
700	100	2.8	2.7	14.8	167.2	167.2	167.0	167.2	167.2	167.2	1	1
800	100	4.1	4.0	21.6	190.9	190.9	190.7	190.9	190.9	190.9	1	1
900	100	5.7	5.5	35.6	213.9	213.9	213.7	213.9	213.9	213.9	1	1
1000	5	8.2	8.2	49.3	243.4	243.4	243.2	243.4	243.4	243.4	1	1
2000	5	57.6	57.6	1075	477.4	477.4	477.2	477.4	477.4	477.4	1	1

We changed the instance type for the fourth set of experiments to sparse vertex-cover instances. Table 4 shows that the running times of $\mathcal{A}_{ex}^{\mathcal{Q}}$ and $\mathcal{A}_{h}^{\mathcal{Q}}$ are comparable and that the accuracy of $\mathcal{A}_{h}^{\mathcal{Q}}$ is very close to 1, showing a very good performance of our methodology based on the quotient game; on the contrary, a comparison between $\mathcal{A}_{h}^{\mathcal{Q}}$ and $\mathcal{A}_{h}^{\mathcal{G}}$ highlights that not only $\mathcal{A}_{h}^{\mathcal{G}}$ is much slower (about one order of magnitude) but its accuracy is lower by a factor which is about 6/7. Note that we did not measure the running times of $\mathcal{A}_{ex}^{\mathcal{G}}$ as most instances reached the timeout without producing a solution.

Table 4 Comparison of $\mathcal{A}_{ex}^{\mathcal{Q}}$, $\mathcal{A}_{h}^{\mathcal{Q}}$ and $\mathcal{A}_{h}^{\mathcal{G}}$ on input generated from sparse vertex-cover instances.

	Time (sec)			No. MustFix			Accuracy	
$\lvert V\rvert$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$
100	2.0e−2	1.9e−2	6.9e−2	38.66	38.63	0	0.999	0.856
200	0.12	0.11	0.56	78.13	78.07	0	0.998	0.864
300	0.34	0.34	2.0	117.7	117.6	0	0.999	0.845
400	0.82	0.77	4.9	156.92	156.9	0	0.999	0.853
500	1.5	1.4	9.8	196.3	196.3	0	0.999	0.850
600	2.2	2.2	17	235.28	235.27	0	1	0.851
700	3.6	3.5	28	274.45	274.42	0	0.999	0.848
800	5.4	5.2	42	313.11	313.10	0	0.999	0.848
900	7.9	7.7	61	352.40	352.39	0	1	0.848
1000	11.2	10.9	86	392.86	392.83	0	0.999	0.849

Our last set of experiments has been run on dense vertex-cover instances, and the results are shown in Table 5.

In this case, the running times are much higher, and we could not run the exact algorithm on instances with more than 70 vertices without timing out. For such small instance sizes, the running times of both heuristics are very similar; however, we can observe a better accuracy for $\mathcal{A}_{h}^{\mathcal{Q}}$. Nevertheless, it is worth noting that $\mathcal{A}_{h}^{\mathcal{Q}}$ shows slightly lower accuracy on dense instances than on sparse ones.

Table 5 Comparison of $\mathcal{A}_{ex}^{\mathcal{Q}}$, $\mathcal{A}_{h}^{\mathcal{Q}}$ and $\mathcal{A}_{h}^{\mathcal{G}}$ on input generated from dense vertex-cover instances.

	Time (sec)			No. MustFix			Accuracy	
$\lvert V\rvert$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$	$\mathcal{A}_{ex}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$	$\mathcal{A}_{h}^{\mathcal{Q}}$	$\mathcal{A}_{h}^{\mathcal{G}}$
10	1.8e−4	1.8e−4	2.1e−4	2.61	2.60	0	0.99	0.98
20	1.2e−3	9.3e−4	1.4e−3	7.59	7.53	0	0.99	0.88
30	9.8e−3	3.6e−3	5.5e−3	13.1	12.3	0	0.97	0.78
40	0.14	1.2e−2	1.6e−2	18.3	15.2	0	0.92	0.74
50	2.1	3.0e−2	3.8e−2	23.7	16.6	0	0.89	0.73
60	28.1	6.4e−2	7.6e−2	29.4	17.0	0	0.87	0.73
70	155	0.12	0.13	35.0	18.1	0	0.86	0.74

We conclude this section observing that, both in Table 4 and in Table 5, no vertices satisfying the MustFix condition are found when running $\mathcal{A}_{h}^{\mathcal{G}}$; this is due to a result from [12], stating that the vertex-cover instances do not contain vertices satisfying the MustFix condition. On the contrary, the quotient graph construction generates vertices with the MustFix condition, therefore improving both the running time and the accuracy of the solution.

5 Conclusion and Future Work

In this work, we analyzed the graph-theoretical aspects of Strategy Repair (SR) to design a more effective graph-based algorithmic approach to its solution. We did it by introducing a new quotient construction tailored to the repair process. Specifically, we defined a notion of vertex equivalence parametrized by the strategy σ_0 to be repaired, enabling the identification and shrinking of the set of vertices that are indistinguishable from the perspective of SR into a single vertex. This construction yields a quotient game whose vertex set is more compact than that of the original instance, while remaining provably solution-preserving with respect to SR. Consequently, solving SR on the quotient game is equivalent to solving it on the original game, but on a size-reduced instance, thereby improving the computational profile of the repair procedure.

Beyond establishing the correctness and invariance properties of the quotient transformation, we conducted several experimental evaluations. First, we compared known algorithms for solving SR applied to instances with and without quotienting. Second, we measured the scalability of the approach over random (sparse) instances of SR. Third, we tested the approach over two classes of Vertex Cover, sparse and dense, looking for instances that supposedly make the problem harder in practice.

The results indicate consistent improvements in running time and, for approximation-based variants, the heuristics in solution quality. These findings suggest that the quotient construction is not merely a theoretical abstraction, but a practically effective preprocessing step that enhances the scalability and robustness of existing SR techniques.

Notice that this quotient idea is algorithm-independent, meaning that it can be applied to any approach solving Strategy Repair, yielding analogous benefits in terms of performance and accuracy. It might also open the door to the design of new approaches dedicated to fully exploiting the properties of quotient games that have not yet been explored.

In addition, we plan to take other directions in the future. First, we plan to extend the equivalence notion to broader classes of objectives or multi-agent settings and generalize the applicability of the approach to a broader class of Strategy Repair problems. Second, we aim to integrate quotient-based reductions into incremental or dynamic repair frameworks to support real-time adaptation in reactive systems. From a practical perspective, embedding the quotient transformation into toolchains for controller synthesis, reactive verification, or large-scale game solving may significantly enhance their scalability.

Finally, a tighter complexity analysis of SR under quotienting could characterize the precise parameter regimes in which compactification yields asymptotic improvements.

References

1 Shaull Almagor, Guy Avni, and Orna Kupferman. Repairing multi-player games. In *Proc. CONCUR 2015*, volume 42 of *LIPIcs*, pages 325–339. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.325.

2 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

3 Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Finite LTL synthesis as planning. In *Proc. ICAPS 2018*, pages 29–38. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17790.

4 Alberto Camacho, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith. LTL realizability via safety and reachability games. In *Proc. IJCAI 2018*, pages 4683–4691. AAAI Press, 2018. doi:10.24963/ijcai.2018/651.

5 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2002.

6 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

7 Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. Strong cyclic planning revisited. In *Proc. ECP 1999*, volume 1809 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 1999. doi:10.1007/10720246_3.

8 Giuseppe De Giacomo, Marco Favorito, Jianwen Li, Moshe Y. Vardi, Shengping Xiao, and Shufang Zhu. LTLf synthesis as AND-OR graph search: Knowledge compilation at work. In *Proc. IJCAI 2022*, pages 2591–2598. ijcai.org, 2022. doi:10.24963/ijcai.2022/359.

9 Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *Proc. IJCAI 2015*, pages 1558–1564. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/223.

10 Volker Diekert and Manfred Kufleitner. Reachability games and parity games. In *Proc. ICTAC 2022*, pages 14–35. Springer, 2022. doi:10.1007/978-3-031-17715-6_3.

11 Nathanaël Fijalkow. *Games on Graphs: From Logic and Automata to Algorithms*. Cambridge University Press, 2026.

12 Pierre Gaillard, Fabio Patrizi, and Giuseppe Perelli. Strategy repair in reachability games. In *Proc. ECAI 2023*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 780–787. IOS Press, 2023. doi:10.3233/FAIA230344.

13 Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *Proc. ICAPS 2018*, pages 88–96. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17732.

14 Alfonso Gerevini and Ivan Serina. Efficient plan adaptation through replanning windows and heuristic goals. *Fundam. Informaticae*, 102(3-4):287–323, 2010. doi:10.3233/FI-2010-309.

15 Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000. doi:10.1145/333979.333987.

16 Farhad Nawaz, Shaoting Peng, Lars Lindemann, Nadia Figueroa, and Nikolai Matni. Reactive temporal logic-based planning and control for interactive robotic tasks. In *Proc. IROS 2024*, pages 12108–12115. IEEE, 2024. doi:10.1109/IROS58592.2024.10802335.

17 Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artif. Intell.*, 76(1-2):427–454, 1995. doi:10.1016/0004-3702(94)00082-C.

18 Dominique Perrin and Jean-Eric Pin. *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*. Elsevier Morgan Kaufmann, 2004.

19 Enrico Scala and Pietro Torasso. Proactive and reactive reconfiguration for the robust execution of multi modality plans. In *Proc. ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 783–788. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-783.

Faster Algorithms for Shortest Unique or Absent Substrings

Panagiotis Charalampopoulos
King's College London, UK

Manal Mohamed
King's College London, UK

Solon P. Pissis
The Cyprus Institute, Nicosia, Cyprus
CWI, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Hilde Verbeek
CWI, Amsterdam, The Netherlands

Wiktor Zuba
University of Warsaw, Poland

Abstract

We revisit two well-known algorithmic problems on strings: computing a *shortest unique substring* (SUS) and a *shortest absent substring* (SAS) in a string S of length n. Both problems admit folklore $\mathcal{O}(n)$-time solutions using the suffix tree of S. However, for small alphabets, this complexity is not necessarily optimal in the word RAM model, where a string of length n over alphabet $[0, \sigma)$ can be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time.

We present an $\mathcal{O}(n \log \sigma / \sqrt{\log n})$-time algorithm for computing a SUS in S. This algorithm decomposes the problem according to the length and the period of the sought substring and uses several tools and techniques, such as synchronizing sets, the analysis of runs, and wavelet trees, to reduce the computation of a SUS to a simple geometric problem. Further, we adapt this algorithm and combine it with an efficient construction of de Bruijn sequences in order to obtain an $\mathcal{O}(n \log \sigma / \sqrt{\log n})$-time algorithm for computing a SAS in S.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, unique substrings, absent substrings, absent words

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.13

Related Version *Full Version*: `https://arxiv.org/abs/2605.04826` [10]

Funding *Hilde Verbeek*: Supported by a Constance van Eeden Fellowship.

1 Introduction

Given a string S over an alphabet Σ, a string P is called *unique* in S if it occurs exactly once in S as a substring. A unique substring P of S is called a *shortest unique substring* (SUS) of S if there is no shorter string that is unique in S. Similarly, a string P over Σ is said to be *absent* from S if it does not occur in S as a substring. A string P that is absent from S is called a *shortest absent substring* (SAS) of S if no shorter string that is absent from S exists. For example, for string $S = \mathtt{gcattgcgtaggt}$ over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$, we have that $\mathtt{ag}$ is a SUS of S and $\mathtt{aa}$ is a SAS of S.

Shortest unique substrings (SUSs) and shortest absent substrings (SASs) have a wide range of applications in bioinformatics, information retrieval, and data compression. In bioinformatics, SUSs are employed in alignment-free sequence comparison methods [22],

© Panagiotis Charalampopoulos, Manal Mohamed, Solon P. Pissis, Hilde Verbeek, and Wiktor Zuba; licensed under Creative Commons License CC-BY 4.0

whereas in information retrieval they are used to extract minimal text snippets from a document collection containing a query term [31]. Note that the notion of SUS used in [31] is position-dependent: for a given position in the input string, one seeks a SUS covering that position. In contrast, in this work we consider the global variant of the problem, where the goal is to compute a shortest substring that is unique in the entire string. SASs are likewise of significant importance in bioinformatics and data compression. They provide highly specific genomic signatures for pathogens such as SARS-CoV-2 [32], thereby facilitating the development of rapid diagnostic assays and targeted therapeutics [34]. Moreover, SASs are utilized in alignment-free sequence comparison methods [8] and constitute the foundational concept of compression schemes based on antidictionaries [13].

It is a classical exercise to compute a SUS or a SAS of a string S of length n in $\mathcal{O}(n)$ time using the suffix tree of S [37]. In particular, all SUSs and SASs are naturally encoded in the suffix tree of S; see Figure 1 for an illustration.

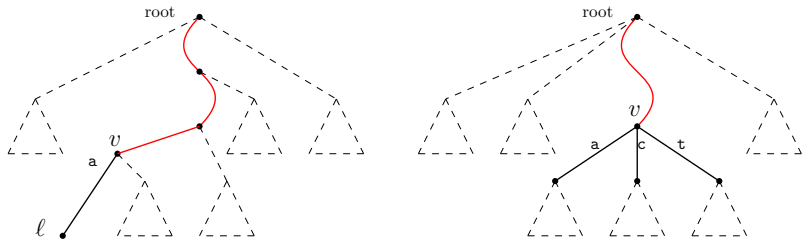


Figure 1 A schematic illustration of two suffix trees. Left, the root-to-v path-label, appended with letter a, is a *unique substring* that occurs as a prefix of the suffix represented by the leaf ℓ. Right, the root-to-v path-label, appended with letter $\mathtt{g} \notin \{\mathtt{a}, \mathtt{c}, \mathtt{t}\}$, yields an *absent substring*.

Here, we consider the unit-cost word RAM model of computation with word size $w = \Theta(\log n)$ for inputs of size n, and a standard instruction set including arithmetic operations, bitwise Boolean operations, and shifts. We measure the space complexity of our algorithms in terms of the number of machine words used. A *packed representation* of a string S over an integer alphabet $\Sigma = [0, \sigma)$ stores $\Theta(\log_\sigma n)$ letters per machine word (possibly apart from the last one), thus representing S in $\mathcal{O}(1 + |S|/\log_\sigma n)$ machine words. A string given in this representation is referred to as a *packed string*.

A large body of work exploits bit-level parallelism in the word RAM model to accelerate classical string processing tasks when the input consists of packed strings. In particular, the problems for which speed-ups have been obtained include pattern matching [1, 3, 4, 5, 7, 20, 21, 30], text indexing [6, 18, 36], computing palindromes [12], constructing longest common extension data structures [25], computing a longest common substring [9], constructing the BWT [25], computing Lempel-Ziv (LZ77) factorizations [16, 27], counting squares and locating runs [11], computing Lyndon arrays [2], computing covers [33], and constructing compressed suffix trees and compressed suffix arrays [26].

In this paper, we ask the following question for a string of length n:

Can a SUS (or a SAS) be computed in $o(n)$ time in the packed setting?

Our results. We answer this question in the affirmative, improving upon the folklore linear-time solutions for computing a single SUS or SAS in the packed setting.

▶ **Theorem 1** (SUS). *Given a packed string S of length n over an integer alphabet $[0, \sigma)$, a shortest unique substring of S can be computed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time.*

▶ **Theorem 2** (SAS). *Given a packed string S of length n over an integer alphabet $[0, \sigma)$, a shortest absent substring of S can be computed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time.*

Our techniques and paper organization. We obtain Theorem 1 by decomposing the problem based on the *length* and the *period* of the sought substring of S. For short substrings, we employ tabulation. For medium-length aperiodic substrings, we modify an existing technique for longest common substrings [9] that is based on an efficient construction of wavelet trees [29]. For long aperiodic substrings, we combine sampling via string synchronizing sets [17, 25] with a heavy-light decomposition [35] of two compacted tries T and T^R. Then, using pairs of heavy paths, we reduce the search to a 2D geometric problem that, as we show, underlies the computation of SUSs. For periodic substrings, we exploit the runs in S [23]: we group them by their Lyndon roots and sparse-Lyndon roots [11] and use this grouping to reduce the search to another instance of said geometric problem. See Section 3.

We obtain Theorem 2 using the same framework, augmented with a new efficient construction of de Bruijn sequences [14] in packed representation. See Section 4.

We start in Section 2 with the necessary preliminaries. Any materials omitted due to space constraints are provided in the full version [10].

Other related work. Kempa and Kociumaka [28] studied the hardness hierarchy for problems whose fastest known word-RAM algorithms run in $\mathcal{O}(n\sqrt{\log n})$ time on inputs of $\Theta(n)$ machine words. This class includes string processing problems such as constructing the BWT [25], computing a longest common substring [9], and computing LZ77 factorizations [27]. For these problems, they showed that the known $\mathcal{O}(n/\sqrt{\log n})$-time algorithms for *binary* strings – equivalently, $\mathcal{O}(n\sqrt{\log n})$-time algorithms when the input size is measured in machine words – are conditionally optimal, with conditional lower bounds established via non-trivial reductions to a variant of dictionary matching. Computing a SUS (Theorem 1) or a SAS (Theorem 2) for binary strings shares the same time complexity. It remains an interesting open problem whether one can improve upon our algorithms or prove that they are conditionally optimal. Let us note that proving a conditional lower bound for our problem using the framework of Kempa and Kociumaka [28] seems challenging. The reductions in said work highlight as the underlying hardness in the studied problems the task of looking for an occurrence of some (sub)string P in a string S (e.g., as in the longest common substring problem). The problems we study here are of a different flavor: in SUS we look for substrings that do not occur elsewhere in S while in SAS we look for strings that do not occur at all in S.

2 Preliminaries

Strings. An *alphabet* Σ is a finite set of elements called *letters*. A *string* $S = S[0]S[1]\cdots S[n-1]$ of *length* $|S| = n$ is a sequence of n letters from Σ. We refer to each $i \in [0, n)$ as a *position* of S. We consider throughout an *integer* alphabet $\Sigma = [0, \sigma)$ with $\sigma = n^{\mathcal{O}(1)}$. A string P is a *substring* of a string S if we have $P = S[i]\cdots S[i + |P| - 1]$ for some position i of S. In this case, we say that P *occurs* at position i of S. Such an occurrence is called a *fragment* of S; we denote it by either $S[i\,..\,i + |P|)$ or $S[i\,..\,i + |P| - 1]$. The set of the starting positions of the *occurrences* of a string P in S is denoted by $\mathrm{Occ}(P, S)$. A *prefix* of S is a fragment of the form $S[0\,..\,j)$, and a *suffix* of S is a fragment of the form $S[i\,..\,n)$. A substring P of S is called *proper* if $P \neq S$. The *reverse* of string S, which we denote by S^R, is defined as $S^R = S[n-1]S[n-2]\cdots S[0]$. The *concatenation* of two strings S and S' is denoted by $S \cdot S'$ and the concatenation of k copies of a string S is denoted by S^k.

▶ **Definition 3** (Period). *An integer $p > 0$ is a* period *of a string P if $P[i] = P[i + p]$, for all $i \in [0, |P| - p)$. The smallest period of P is called* the period *of P and is denoted by $\mathsf{per}(P)$. A string P is called* periodic *if $\mathsf{per}(P) \leq |P|/2$ and* aperiodic *otherwise.*

▶ **Example 4.** For string $P = \texttt{abaaabaaabaaaba}$, we have $\mathsf{per}(P) = 4$. Note that 8 is also a period of P. Since $\mathsf{per}(P) = 4 \leq |P|/2 = 15/2$, P is periodic.

▶ **Definition 5** (Run). *A fragment $F = S[i\,..\,j]$ of a string S is called a* run *of S if $\mathsf{per}(F) \leq |F|/2$, and extending F either to the left or to the right (if possible) would result in an increase of its period, that is, $S[i-1] \neq S[i-1+\mathsf{per}(F)]$ (or $i = 0$) and $S[j+1] \neq S[j+1-\mathsf{per}(F)]$ (or $j = |S|-1$).*

▶ **Definition 6** (Lyndon Root). *The* Lyndon root *of a run F, denoted by $\mathsf{Lroot}(F)$, is defined as the lexicographically smallest rotation of the prefix $F[0\,..\,\mathsf{per}(F))$ of F.*

▶ **Definition 7** (Lyndon Representation). *The Lyndon representation of a run F is a quadruple $\mathsf{Lrepr}(F) = (\lambda, e, \alpha, \beta)$ such that:*

- *$\lambda = \mathsf{Lroot}(F)$, and*
- *$F = P \cdot \lambda^e \cdot T$, where P is a (possibly empty) suffix of λ with $|P| = \alpha < |\lambda|$, and T is a (possibly empty) prefix of λ with $|T| = \beta < |\lambda|$.*

▶ **Example 8.** Let $S = \underline{\texttt{abaaabaaabaaaba}}\texttt{b}$. The *underlined* fragment $F = S[0\,..\,14] = \texttt{abaaabaaabaaaba}$ is a run with $\mathsf{per}(F) = 4 \leq |F|/2 = 15/2$. Observe that extending F to the right yields the fragment $S[0\,..\,15] = S$, whose period is 14; hence extending F increases its period. Moreover, we have $\mathsf{Lroot}(F) = \texttt{aaab}$ and $\mathsf{Lrepr}(F) = (\texttt{aaab}, 3, 2, 1)$.

▶ **Definition 9** (τ-Run). *A run R of a string S is called a* τ-run*, for an integer $\tau > 0$, if $|R| \geq 3\tau - 1$ and $\mathsf{per}(R) \leq \frac{1}{3}\tau$.*

▶ **Lemma 10** (Lemma 2.8 [9]). *Let S be a string of length n over an integer alphabet $[0, \sigma)$, with $\sigma = n^{\mathcal{O}(1)}$, and let $\tau > 0$ be an integer. Then S contains $\mathcal{O}(n/\tau)$ τ-runs. Moreover, if $\tau \leq \frac{1}{4}\log_\sigma n$, all τ-runs in S can be computed and grouped by their Lyndon roots in $\mathcal{O}(n/\tau)$ time. Within the same time bound, for each τ-run, we can compute the two leftmost occurrences of its Lyndon root.*

▶ **Definition 11** (Shortest Unique Substring (SUS)). *A string P is a* unique substring *of a string S if and only if $|\mathrm{Occ}(P, S)| = 1$. A unique substring P of S is called a* shortest unique substring *of S if there is no string P' with $|P'| < |P|$ that is a unique substring of S.*

▶ **Definition 12** (Shortest Absent Substring (SAS)). *A string P is* absent *from a string S if and only if $|\mathrm{Occ}(P, S)| = 0$. An absent string P of S is called a* shortest absent substring *of S, if there is no string P' with $|P'| < |P|$ that is absent from S.*

String synchronizing sets. We next define a powerful sampling mechanism.

▶ **Definition 13** (String Synchronizing Set [25]). *For a string S of length n and a positive integer $\tau \in [1, \lfloor \frac{n}{2} \rfloor]$, a set $\mathbf{Sync} \subseteq [0, n-2\tau]$ is a* τ-synchronizing set *of S if it satisfies:*

1. *Consistency: For $i, j \in [0, n-2\tau]$, if $S[i\,..\,i+2\tau) = S[j\,..\,j+2\tau)$, then $i \in \mathbf{Sync}$ if and only if $j \in \mathbf{Sync}$.*
2. *Density: For $i \in [0, n-3\tau+1]$, $\mathbf{Sync} \cap [i, i+\tau) = \emptyset$ if and only if $\mathsf{per}(S[i\,..\,i+3\tau-1)) \leq \frac{1}{3}\tau$.*

▶ **Theorem 14** ([17]). *A string $S \in [0, \sigma)^n$ can be preprocessed in $\mathcal{O}(\frac{n}{\log_\sigma n})$ time so that, given $\tau \in [1, \lfloor n/2 \rfloor]$, a τ-synchronizing set $\mathbf{Sync}$ of S of size $|\mathbf{Sync}| < \frac{70n}{\tau}$ can be constructed in $\mathcal{O}(n/\tau)$ time.*

Problem definitions. We now formally define the problems in scope.

Shortest Unique Substring
Input: A packed string S of length n over an integer alphabet $\Sigma = [0, \sigma)$.
Output: $i, j \in \mathbb{Z}_{\geq 0}$ such that $P = S[i\,..\,j]$ is a shortest unique substring of S.

Shortest Absent Substring
Input: A packed string S of length n over an integer alphabet $\Sigma = [0, \sigma)$.
Output: $i, j \in \mathbb{Z}_{\geq 0}$ and $c \in \Sigma$ such that $P = S[i\,..\,j] \cdot c$ is a shortest absent substring of S.

The folklore $\mathcal{O}(n)$-time solutions to the above problems are sketched in [10]. In [10], we also show that any Shortest Unique Substring instance of length n over alphabet $[0, \sigma)$ can be reduced in $\mathcal{O}(\frac{n}{\log_\sigma n})$ time to an instance of length $\mathcal{O}(n \log \sigma)$ over a binary alphabet. This implies that the binary alphabet constitutes a hardest case for computing a SUS. In particular, any algorithm designed for the binary case can be applied to general alphabets via this reduction, without any increase in the asymptotic running time.

In what follows, we assume that all considered strings are given in packed representation. When it is clear from the context, we may omit the term *packed*.

3 Computing a Shortest Unique Substring

We decompose the problem into four cases based on the length ℓ and the period p of a SUS:

1. **Short case**: $\ell \leq \frac{1}{5} \log_\sigma n$ (see [10]);
2. **Medium aperiodic case**: $\ell \in (\frac{1}{5} \log_\sigma n, 2^{\sqrt{\log n}})$ and $p > \frac{1}{45} \log_\sigma n$ (see [10]);
3. **Long aperiodic case**: $\ell \geq \log^4 n$ and $p > \frac{1}{9} \log^4 n$ (see Section 3.2);
4. **Periodic case**: $\ell \geq 3\tau$ and $p \leq \frac{1}{3}\tau$, for $\tau = \lfloor \frac{1}{15} \log_\sigma n \rfloor$ or $\tau = \lfloor \frac{1}{3} \log^4 n \rfloor$ (see Section 3.3). Note that when $\tau = 0$, this case is obsolete because the period of any string is positive.

We defer the proofs for the *short* (see Lemma 15) and *medium aperiodic* (see Corollary 16) cases to the full version [10]: the former case is standard, and for the latter one we draw heavily from previous work [9]. For those cases, we assume familiarity with *suffix trees* and *wavelet trees*; a formal exposition of these is also given in [10].

▶ **Lemma 15** ([10]). *Given an instance of Shortest Unique Substring, in $\mathcal{O}(n/\log_\sigma n)$ time, we can either compute a SUS of S of length at most $\ell := \lfloor \frac{1}{5} \log_\sigma n \rfloor$ if one exists, or conclude that no SUS of length at most ℓ exists.*

▶ **Corollary 16** ([10]). *Given an instance of Shortest Unique Substring, we can compute a SUS P of S in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time if it has length $|P| \in [\frac{1}{5} \log_\sigma n, 2^{\sqrt{\log n}}]$ and period $p > \frac{1}{45} \log_\sigma n$.*

3.1 SUS as a Skyline Problem

In this section, we introduce and solve a simple geometric problem that serves as a subroutine for computing a SUS in both the long aperiodic and the periodic cases.

▶ **Definition 17** (Domination in $\mathbb{Z}^2_{\geq 0}$). *Let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be two points in $\mathbb{Z}^2_{\geq 0}$. We say that p_1 is* dominated by *p_2 if $x_1 \leq x_2$ and $y_1 \leq y_2$.*

▶ **Definition 18** (Shadow). *Let P be a multiset of points in $\mathbb{Z}^2_{\geq 0}$. The* shadow *of a point $p \in P$ is the set of points in $\mathbb{Z}^2_{\geq 0}$ that are dominated by p but not dominated by any $p' \in P \setminus \{p\}$.*

▶ **Definition 19** (Skyline). *Let P be a multiset of points in $\mathbb{Z}^2_{\geq 0}$. The* primary skyline *of P is the union of the shadows of all points in P.*

▶ **Example 20.** The primary skyline of $\{(7,3),(7,3)\}$ is the empty set.

In the following, we formalize the Minimum Skyline Point problem, explain its relevance to finding a SUS of S, and present a linear-time solution; see Figure 2 for an example.

Minimum Skyline Point
Input: A multiset P of points in $\mathbb{Z}^2_{\geq 0}$.
Output: A point $(x, y) \in \mathbb{Z}^2_{\geq 0}$ that lies in the primary skyline of P and minimizes $x + y$, if one exists.

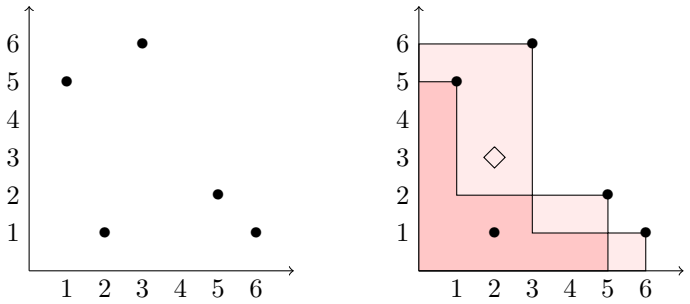


Figure 2 A set P (left) and its primary skyline (right). The union of the shadows of the points in P consists of all points within the shaded polygon but outside its heavily shaded part – the points in the heavily shaded part are dominated by multiple points from P. For this instance of Minimum Skyline Point, the output would be point $(2,3)$, shown with a rhombus.

▶ **Lemma 21.** *Any instance of Minimum Skyline Point can be solved in $\mathcal{O}(|P|)$ time if the points in P are given as a list sorted with respect to one of the two coordinates.*

Proof. We first establish two claims.

▷ Claim 22. Suppose that the primary skyline is nonempty and let $(x^\star, y^\star)$ be a point in the primary skyline with minimal sum of coordinates. Then, either $x^\star = 0$ or there exists a point $(x, y) \in P$ with $x = x^\star - 1$.

Proof. Assume, toward a contradiction, that the claim is false; that is, that we have $x^\star > 0$ and there is no point $(x, y) \in P$ with $x = x^\star - 1$. Let $p^\star = (x^\star, y^\star)$. By the definition of the primary skyline, $p^\star$ must be dominated by exactly one point in P. Now consider the point $p_{\text{left}} = (x^\star - 1, y^\star)$, which, due to the minimality of the coordinate-wise sum of $p^\star$, must be dominated by at least two points in P. Therefore, there exists a point $p' \in P$ that dominates p_{left} and does not dominate $p^\star$. Let $p' = (x', y')$. Since p' does not dominate $p^\star$, we must have either $x' < x^\star$ or $y' < y^\star$. We obtain a contradiction in each case:

- If $x' < x^\star$, then by our assumption that no point $(x, y) \in P$ has $x = x^\star - 1$, we must have $x' \leq x^\star - 2$. Hence, $x' < x^\star - 1$, and thus p' does not dominate p_{left}, a contradiction.
- If $y' < y^\star$, then p' clearly does not dominate p_{left}, a contradiction. ◁

▷ Claim 23. Suppose that the primary skyline is nonempty and let $(x^\star, y^\star)$ be a point in the primary skyline with minimal sum of coordinates. Let $P' = \{(x, y) \in P \mid x \geq x^\star\}$ and let p_{unique} be the unique point of P' that dominates $(x^\star, y^\star)$. If $|P'| = 1$, we have $y^\star = 0$. Else, we have $y^\star = y' + 1$, where y' is the maximum y-coordinate of a point in $P' \setminus \{p_{\text{unique}}\}$.

Proof. By definition, $p^\star := (x^\star, y^\star)$ is dominated by exactly one point, namely p_{unique}. Since points are only dominated by points weakly to their right, we have that p_{unique} must be in P'. We distinguish between two cases:

Case 1: $|P'| = 1$. We have $P' = \{p_{\text{unique}}\}$. If $y^\star > 0$, then the point $p_{\text{down}} = (x^\star, y^\star - 1)$ is in $\mathbb{Z}^2_{\geq 0}$ and the only point of P that dominates it is p_{unique}. Thus, p_{down} lies in the primary skyline, but its coordinate sum $x^\star + y^\star - 1$ is smaller than that of $p^\star$, contradicting the minimality of $p^\star$. Hence, $y^\star = 0$.

Case 2: $|P'| \geq 2$. Let $P_{\text{other}} = P' \setminus \{p_{\text{unique}}\}$, and let y' be the maximum y-coordinate of a point in P_{other}. Since $p^\star$ is dominated uniquely by p_{unique}, every point $(x'', y'') \in P_{\text{other}}$ satisfies $y'' < y^\star$. Hence, $y' < y^\star$. Conversely, since p_{unique} dominates $p^\star$, its y-coordinate is at least $y^\star$. Therefore, the point $(x^\star, y^\star)$ is dominated by exactly one point in P' if and only if $y' < y^\star$. By the minimality of $p^\star$, we have $y^\star = y' + 1$. ◁

Algorithm. We assume, without loss of generality, that the points in P are sorted by x-coordinate in non-decreasing order. We scan P from right to left, while maintaining $y_{\max}$ and y', the two largest y-coordinates encountered among processed points (where $y_{\max} \geq y'$). We initialize $y_{\max}$ and y' to -1, indicating that no processed point has contributed a valid y-coordinate yet.

We iterate through P by grouping points with the same x-coordinate. Let x_i denote the current x-coordinate of the group currently being processed, and let $P_{\text{curr}} = \{(x, y) \in P \mid x = x_i\}$ be the set of points at this coordinate. Before processing the points in P_{curr}, we evaluate the candidate x-coordinate $x^\star = x_i + 1$. Because we have scanned from right to left, all points $(x, y) \in P$ such that $x \geq x^\star$ have already been processed. Let P' denote this set of previously processed points. We determine the corresponding $y^\star$ following Claim 23:

- If $|P'| = 1$, we set $y^\star = 0$.
- If $|P'| \geq 2$ and $y_{\max} > y'$, we set $y^\star = y' + 1$.
- Otherwise (if $|P'| = 0$ or $y_{\max} = y'$), the primary skyline does not intersect the set $\{(x^\star, y) \mid y \in \mathbb{Z}_{\geq 0}\}$, so we skip this candidate.

After evaluating $y^\star$, we update $y_{\max}$ and y' using the y-values in P_{curr} so that they remain the two largest y-coordinates among all processed points.

Finally, after all points in P have been processed, we perform a last check for the candidate $x^\star = 0$ using the final values of $y_{\max}$ and y'. The algorithm maintains the candidate $(x^\star, y^\star)$ that minimizes $x^\star + y^\star$ and returns it as a witness. Each point of P is processed exactly once using $\mathcal{O}(1)$ simple operations, thus the algorithm runs in $\mathcal{O}(|P|)$ time. ◀

Intuition for application to SUS. In the long aperiodic case (Section 3.2) and in the periodic case (Section 3.3), we show that a SUS of S can be found by solving several instances of the Minimum Skyline Point problem. The core idea is to transform the SUS problem into a geometric one. Let us consider one such instance. We first identify a set $\mathcal{S}$ of carefully-selected substrings of S, each anchored around a *common fragment* $F = S[i\,..\,j]$. Each substring $U \in \mathcal{S}$ is then represented by an integer point (x, y), where x and y are the lengths of the extensions of U to the left and the right, respectively, relative to the common fragment F.

▶ **Example 24.** Let $S = \texttt{abracadabra}$, $F = S[4\,..\,6] = \texttt{cad}$, and $\mathcal{S} = \{\texttt{bracada}, \texttt{cadabra}\}$. The substrings are represented as points by the lengths of their extensions relative to F: $\texttt{bracada} \mapsto (3, 1)$ (since $\texttt{bra}$ is of length 3 and $\texttt{a}$ is of length 1), and $\texttt{cadabra} \mapsto (0, 4)$ (since the left extension is empty and $\texttt{abra}$ is of length 4).

Since strings in $\mathcal{S}$ are substrings of S anchored around a common fragment, each point maps to one substring of S. Then, a string $U \in \mathcal{S}$ is a substring of string $V \in \mathcal{S}$ if and only if the point representing U is dominated by the point representing V. To find a candidate SUS in $\mathcal{S}$, we look for a minimal extension anchored around F that is contained within exactly one substring from $\mathcal{S}$. This is equivalent to solving a Minimum Skyline Point instance: finding an integer point $(x^\star, y^\star)$ that lies in the primary skyline of the points representing $\mathcal{S}$ that minimizes $x^\star + y^\star$. The total length of the resulting SUS is then $|F| + x^\star + y^\star$.

3.2 Long Aperiodic Case

This section addresses the computation of a SUS in the case when $\ell \geq \log^4 n$ and $p > \frac{1}{9}\log^4 n$. Let us start with a high-level overview of our solution for this case. We construct a string synchronizing set consisting of *anchor* positions in S that enable the identification of identical sufficiently long aperiodic patterns. These anchors are then used to construct two tries whose root-to-leaf paths represent occurrences of these patterns. SUSs are found by locating the shortest paths that occur only once. We achieve this by formulating several instances of the Minimum Skyline Point problem based on these tries.

We apply Theorem 14 on S with $\tau := \lfloor \frac{1}{3}\log^4 n \rfloor$ and denote the resulting τ-synchronizing set by **Sync**. The following proposition is crucial: if a sufficiently long, aperiodic pattern occurs twice in the string, then every anchor within one occurrence must have a corresponding anchor at the same relative position in the other occurrence. Therefore, to find a unique pattern, we must locate an anchor whose surrounding fragments – those immediately preceding and succeeding it – do not occur around any other anchor and have minimal total length.

▶ **Proposition 25** (follows by Definition 13)**.** *Suppose that a substring X of S with length $|X| = m \geq 3\tau$ and period $\mathsf{per}(X) > \frac{1}{3}\tau$ occurs at distinct positions i and j in S. Then for all $q \in [0, m - 2\tau]$, we have $i + q \in \mathbf{Sync}$ if and only if $j + q \in \mathbf{Sync}$.*

We construct two tries T and T^R. For every anchor position $i \in \mathbf{Sync}$, we insert $S[i\,..\,n)$ into T and $S[0\,..\,i)^R$ into T^R. In both tries, the leaves are labeled with their corresponding anchor positions. For efficiency, we implement them as *compacted* tries, where nodes with a single child are suppressed and are called *implicit*; all other nodes (i.e., the root, branching nodes, and leaves) are maintained and are called *explicit*. For any explicit node v, let $L(v)$ denote the set of leaf labels in the subtree rooted at v, let $\mathsf{str}(v)$ denote the concatenation of the edge labels from the root to v, and let $\mathsf{sd}(v)$ denote the string depth of v. Note that these definitions extend naturally to implicit nodes. If an occurrence of a pattern P contains an anchor $i \in \mathbf{Sync}$, then there exist nodes u in T^R and v in T such that $P = \mathsf{str}(u)^R \cdot \mathsf{str}(v)$ and $i \in L(u) \cap L(v)$. Moreover, the size $|L(u) \cap L(v)|$ equals the number of occurrences of P, provided that P is sufficiently long and aperiodic. We formalize our task as follows; for convenience, for the remainder of this section, we assume that the parent of the root node is itself.

Two Trees SUS
Input: Two rooted trees T_1 and T_2, each with size at most N and with leaves uniquely labeled from $[0, N)$, a weight function w with range $\mathbb{Z}_{\geq 0}$ where $w(u) > w(v)$ for every strict ancestor v of every node u, and an integer k.
Output: A pair (u, v) of nodes u in T_1 and v in T_2 minimizing $w(\mathsf{parent}(u)) + \max(w(\mathsf{parent}(v)), k-1)$, subject to $|L(u) \cap L(v)| = 1$ and $w(v) \geq k$ (if they exist).

▶ **Lemma 26.** *Consider a Shortest Unique Substring instance and let* $\tau := \lfloor \frac{1}{3} \log^4 n \rfloor$. *If S has a SUS of length $\ell \geq 3\tau$ and period $p > \frac{1}{3}\tau$, we can reduce its computation in $\mathcal{O}(n/\log_\sigma n)$ time to an instance of Two Trees SUS with $N = \mathcal{O}(n/\tau)$ and $k = 2\tau$.*

Proof. We begin by constructing a τ-synchronizing set **Sync** of S. Using this set, we construct two compacted tries T and T^R. For every anchor $i \in$ **Sync**, we insert the suffix $S[i\,..\,n)$ into T and the reversed prefix $S[0\,..\,i)^R$ into T^R, labeling the corresponding leaves with the anchor i in both tries. We make all implicit nodes in T at string depth $k-1$ explicit. Hence, we have $N = \mathcal{O}(|\mathbf{Sync}|) = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log^4 n)$.

Any fragment P in S that contains at least one anchor $i \in$ **Sync** can be decomposed into $P = \mathsf{str}(u)^R \cdot \mathsf{str}(v)$, where u is a node in T^R and v is a node in T, both of which can be implicit. If P has length at least 3τ and period greater than $\frac{1}{3}\tau$, then by the definition of τ-synchronizing sets, all occurrences of P have anchors at the same relative positions. Specifically, any occurrence of P starting at position j must have $j + q \in$ **Sync** if and only if the original occurrence at position i had an anchor at the same offset (Proposition 25). Thus, if the node pair (u, v) represents the fragment P, the number of occurrences of P is exactly $|L(u) \cap L(v)|$. To find a SUS, we must find a pair (u, v) of (possibly implicit) nodes such that $|L(u) \cap L(v)| = 1$ (uniqueness), $\mathsf{sd}(v) \geq 2\tau$ (to satisfy the anchor offset property), and $\mathsf{sd}(u) + \mathsf{sd}(v)$ is minimized.

To do so, we set the weights of nodes in the two trees as follows. In both tries, we set the weight of the root node to 0 and assign to each other node u a weight $w(u) := \mathsf{sd}(u)$. We thus create an instance of Two Trees SUS with $k = 2\tau$. If the optimal node pair (u, v) satisfies $w(\mathsf{parent}(u)) + w(\mathsf{parent}(v)) + 2 < 3\tau$, we conclude that S does not have a SUS of length $\ell \geq 3\tau$ and period $p > \frac{1}{3}\tau$. (Note that since we have made all nodes with string depth $k-1$ explicit, for any node v with $w(v) \geq k$, v's parent has weight at least $k-1$, and hence $\max(w(\mathsf{parent}(v)), k-1) = w(\mathsf{parent}(v))$.) Otherwise, we return as a SUS the string $\mathsf{str}(u)^R[0\,..\,w(\mathsf{parent}(u))] \cdot \mathsf{str}(v)[0\,..\,w(\mathsf{parent}(v))]$ – the indices of the occurrence of this string in S can be inferred from the common leaf label in the subtrees of u and v.

The time complexity depends on the construction of the τ-synchronizing set, which requires $\mathcal{O}(n/\log_\sigma n)$ time using Theorem 14, and the construction of the tries. For the tries, we first construct an LCE data structure over S in $\mathcal{O}(n/\log_\sigma n)$ time [25] supporting $\mathcal{O}(1)$-time LCP queries. We sort the N suffixes of S in $\mathcal{O}(N \log N)$ time using merge sort, performing each comparison with an LCP query and a letter comparison. Given the sorted list of suffixes and the LCE data structure, the tries can then be constructed in $\mathcal{O}(N)$ time [24]. The total time for constructing the tries is $\mathcal{O}(n/\log_\sigma n) + \mathcal{O}(N \log N) \subseteq \mathcal{O}(n/\log_\sigma n)$. ◀

3.2.1 Solving the Two Trees SUS Problem

We solve Two Trees SUS as follows. We first decompose the trees T_1 and T_2 into *heavy paths*. For each pair of heavy paths, one from T_1 and one from T_2, we then construct a Minimum Skyline Point instance, which we solve in linear time using Lemma 21.

Heavy-light decomposition. We first recall the widely-used heavy-light decomposition [35].

▶ **Definition 27** (Heavy-Light Decomposition [35]). *Consider a rooted tree $\mathcal{T}$. We obtain a* heavy-light decomposition *of $\mathcal{T}$ by marking each edge as either heavy or light as follows. For every internal node of $\mathcal{T}$, the outgoing edge leading to the child with the largest number of descendants is marked as heavy, while all other outgoing edges are marked as light; ties are resolved arbitrarily. A maximal path of heavy edges is a* heavy path.

A heavy-light decomposition can be constructed in linear time [35]. The following fact holds for any heavy-light decomposition:

▶ **Fact 28** (Lemma 1 [35]). *In a tree with N leaves, any root-to-leaf path intersects at most $\log N$ heavy paths in the decomposition.*

Furthermore, we can construct a *heavy-path tree* that encodes the ancestral relations among all heavy paths and leaves. This is done by contracting every heavy edge, such that all remaining edges are light. In this auxiliary structure, each node corresponds to a contracted heavy path or a leaf. By Fact 28, the heavy-path tree has height at most $\log N$, which allows for efficient traversal across the original tree structure.

A reduction using pairs of heavy paths. We first note that any internal node of a tree belongs to exactly one heavy path in its heavy-light decomposition. We compute the heavy-light decompositions for T_1 and T_2 and consider every pair of heavy paths h_1 from T_1 and h_2 from T_2, provided that they share at least one leaf label.

For a heavy path h, let $L(h)$ denote the set of all leaves descending from the root of h. We define the function $d_h : L(h) \to \mathbb{Z}_{\geq 0}$, where for some leaf $\ell \in L(h)$, $d_h(\ell)$ denotes the weight of ℓ's lowest ancestor within h. Namely, $d_h(\ell)$ denotes the maximum weight of a node on h that is an ancestor of leaf ℓ. With these definitions, we have the following observation:

▶ **Observation 29.** *Given a heavy path h, a node v on h, and a leaf ℓ descending from the root of h, we have that $\ell \in L(v)$ if and only if $w(v) \leq d_h(\ell)$.*

To solve Two Trees SUS, we show that it suffices to solve a Minimum Skyline Point instance for every pair of heavy paths (h_1, h_2) sharing at least one leaf label. By Fact 28, each leaf belongs to at most $\log N$ sets $L(h)$ per tree. We can thus construct each subset $L(h_1) \cap L(h_2)$ by enumerating, for each leaf label ℓ, all $\log^2 N$ pairs of heavy paths above it.

▶ **Lemma 30.** *Any instance of Two Trees SUS can be solved in $\mathcal{O}(N \log^3 N)$ time.*

Proof. We construct the heavy-light decompositions of T_1 and T_2 in $\mathcal{O}(N)$ time [35]. This process partitions each tree into a set of disjoint heavy paths.

For every leaf label $\ell \in [0, N)$ present in both T_1 and T_2, we identify all pairs of heavy paths (h_1, h_2) such that h_1 lies on the root-to-leaf path in the first tree and h_2 lies on the root-to-leaf path in the second tree. Since any root-to-leaf path intersects at most $\mathcal{O}(\log N)$ heavy paths (Fact 28), there are $\mathcal{O}(\log^2 N)$ such pairs for each of the N labels. For each pair, we generate a tuple $(h_1, h_2, d_{h_1}(\ell), d_{h_2}(\ell))$, where d_{h_1} and d_{h_2} are retrieved from the weights of the light-edge endpoints. All tuples are generated in $\mathcal{O}(N \log^2 N)$ time in total and are stored in one list. We sort the list, using the heavy path identifiers (h_1, h_2) as primary keys and the value $d_{h_1}(\ell)$ as the secondary key. This ensures that all tuples corresponding to the same pair (h_1, h_2) appear consecutively, and are then ordered by $d_{h_1}(\ell)$. Using merge sort, this step takes $\mathcal{O}(N \log^2 N \cdot \log(N \log^2 N)) = \mathcal{O}(N \log^3 N)$ time. Let us denote the sublist corresponding to the pair (h_1, h_2) of heavy paths by $\mathcal{H}(h_1, h_2)$.

▷ Claim 31. Consider an instance of Two Trees SUS, in which the output nodes are restricted to a given pair of heavy paths h_1 and h_2 from T_1 and T_2. Given the list $\mathcal{H}(h_1, h_2)$, we can reduce this instance in $\mathcal{O}(g)$ time to an instance of the Minimum Skyline Point problem over a multiset of points of size $\mathcal{O}(g)$, where $g = |\mathcal{H}(h_1, h_2)|$.

Proof. Recall that in the definition of Two Trees SUS, the integer k is given as a minimum weight on one of the returned nodes. Let $\mathcal{L} = L(h_1) \cap L(h_2)$. Assume that $\mathcal{L} \neq \emptyset$; otherwise, the instance has no solution. We wish to find nodes $u \in h_1$ and $v \in h_2$ minimizing $w(\mathsf{parent}(u)) + \max(w(\mathsf{parent}(v)), k-1)$ such that $|L(u) \cap L(v)| = 1$ and $w(v) \geq k$. By Observation 29, this reduces to finding the minimum $(x_1, x_2) \in \mathbb{Z}^2_{\geq 0}$ such that exactly one leaf $\ell \in \mathcal{L}$ satisfies $d_{h_1}(\ell) \geq x_1$ and $d_{h_2}(\ell) \geq x_2$. This is equivalent to solving a Minimum Skyline Point on the multiset $P' = \{(d_{h_1}(\ell), d_{h_2}(\ell)) \mid \ell \in \mathcal{L}\} \sqcup \{(\infty, k-1), (\infty, k-1)\}$, thus finding a solution $(x^\star, y^\star)$ with minimal $x^\star + y^\star$ – the extra point that we insert in the multiset twice ensures that if the primary skyline is not empty, then $y^\star \geq k$. Given $\mathcal{H}(h_1, h_2)$, we can construct this instance in $\mathcal{O}(g)$ time. Finally, to obtain a solution to Two Trees SUS, we select the nodes u in h_1 and v in h_2 satisfying $w(\mathsf{parent}(u)) + 1 = x^\star$ and either $w(\mathsf{parent}(v)) + 1 = y^\star$ or $w(\mathsf{parent}(v)) + 1 \leq k = y^\star \leq w(v)$. ◁

We apply Claim 31 to each pair of heavy paths that share at least one leaf label, and solve each instance of the Minimum Skyline Point problem using Lemma 21 in time linear in the number of points. Since the total number of tuples across all pairs of heavy paths is $\mathcal{O}(N \log^2 N)$, the total time required for applications of Claim 31 and Lemma 21 is $\mathcal{O}(N \log^2 N)$. We maintain the global minimum value of $w(\mathsf{parent}(u)) + \max(w(\mathsf{parent}(v)), k-1)$ found across all pairs of heavy paths and return a witness pair of nodes as the final solution. ◀

3.2.2 Wrapping Up

The final complexity of the long aperiodic case is determined by the combination of the reduction to the Two Trees SUS problem and the subsequent application of the heavy path based Minimum Skyline Point algorithm.

▶ **Lemma 32.** *Given an instance of Shortest Unique Substring, we can compute a SUS of S in $\mathcal{O}(n/\log_\sigma n)$ time if it has length $\ell \geq \log^4 n$ and period $p > \frac{1}{9}\log^4 n$.*

Proof. Using Lemma 26, we reduce the long aperiodic case to an instance of the Two Trees SUS problem. The number of leaf labels (anchors) is $N = \mathcal{O}(|\mathbf{Sync}|) = \mathcal{O}(n/\log^4 n)$. As established in the reduction, constructing the synchronizing set, the LCE data structure, and the two compacted tries T and T^R takes $\mathcal{O}(n/\log_\sigma n)$ time. We then solve the resulting Two Trees SUS instance using the algorithm from Lemma 30. The time required is $\mathcal{O}(N \log^3 N)$ and as N is in $\mathcal{O}(n/\log^4 n)$, we obtain a running time of $\mathcal{O}(n/\log n)$. Therefore, both the reduction and the solver fit within the target time bound of $\mathcal{O}(n/\log_\sigma n)$. The algorithm identifies a pair (u, v) minimizing $w(\mathsf{parent}(u)) + \max(w(\mathsf{parent}(v)), k-1)$ subject to the uniqueness and length constraints. Since the result of the Two Trees SUS problem provides the minimal unique extension for any pattern overlapping an anchor, the resulting substring is a valid SUS for the long aperiodic case. ◀

3.3 Periodic Case

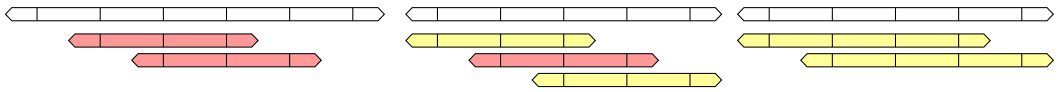
In this section, we address the SUS computation in highly periodic substrings of S. Specifically, we consider substrings of S of medium length with period at most $\frac{1}{45}\log_\sigma n$, as well as long substrings of S with period at most $\frac{1}{9}\log^4 n$. These two cases are handled using τ-runs with $\tau = \lfloor \frac{1}{15}\log_\sigma n \rfloor$ and $\tau = \lfloor \frac{1}{3}\log^4 n \rfloor$, respectively.

For the first group of τ-runs (where $\tau = \lfloor \frac{1}{15} \log_\sigma n \rfloor$), we employ their standard Lyndon representation (Definition 7). These runs can be efficiently computed and grouped by means of Lemma 10. Unfortunately, Lemma 10 is not applicable when $\tau = \lfloor \frac{1}{3} \log^4 n \rfloor$. To deal with this, we utilize the recently introduced *sparse-Lyndon representation* [11] that allows for efficiently grouping the τ-runs for $\tau = \lfloor \frac{1}{3} \log^4 n \rfloor$ according to their Lyndon roots, but representing via their sparse-Lyndon root. We formalize this discussion in Lemma 33.

▶ **Lemma 33** (Proposition 36 [11]). *For any string S of length n over an integer alphabet $[0, \sigma)$, with $\sigma = n^{\mathcal{O}(1)}$, all runs in S can be computed and grouped by equal Lyndon roots in $\mathcal{O}(n/\log_\sigma n)$ time. For runs with period at most $2\lfloor \frac{1}{18} \log_\sigma n \rfloor$, we compute their standard Lyndon representations, while for runs with larger periods we compute their sparse-Lyndon representations.*

To streamline the subsequent analysis, we hereafter slightly *abuse terminology*: the terms τ-run, Lyndon root, and Lyndon representation will be used uniformly for the two classes of runs considered in this section. In the context of the second class ($\tau = \lfloor \frac{1}{3} \log^4 n \rfloor$), these terms implicitly refer to their sparse-Lyndon counterparts.

This unified terminology is justified by the fact that both representations share analogous properties and can be handled identically within our algorithmic framework. We group all τ-runs by Lyndon root in $\mathcal{O}(n/\log_\sigma n)$ time using Lemma 33; the only difference is that for some groups, we compute the Lyndon representation of runs, while for others, we compute the sparse-Lyndon representation. For further details and a thorough discussion of sparse-Lyndon representations, we refer the reader to [11].



(a) $e < e' - 2$ (Case 1). **(b)** $e = e' - 2$ (Case 2). **(c)** $e = e' - 1$ (Case 3).

Figure 3 Cases 1–3 of Lemma 34, illustrating the potential alignment of a run with exponent e within a run with exponent e'. Red alignments are guaranteed to be contained within the longer run; yellow alignments are contained only if the extensions of the runs satisfy specific inequalities.

▶ **Lemma 34.** *Let F and F' be two runs in S with the same Lyndon root λ, where $\mathsf{Lrepr}(F) = (\lambda, e, \alpha, \beta)$ and $\mathsf{Lrepr}(F') = (\lambda, e', \alpha', \beta')$. We can determine whether F is a* unique substring *of F' as follows:*

1. *If $e < e' - 2$, then F is a substring of F', but it is not unique.*
2. *If $e = e' - 2$, then F is a substring of F'. It is unique if and only if $\alpha > \alpha'$ and $\beta > \beta'$.*
3. *If $e = e' - 1$, then F is a unique substring of F' if and only if* either *$\alpha > \alpha'$* or *$\beta > \beta'$, but not both. (Note: if both $\alpha > \alpha'$* and *$\beta > \beta'$ hold, F is not a substring of F'.)*
4. *If $e = e'$, then F is a unique substring of F' if and only if $\alpha \leq \alpha'$ and $\beta \leq \beta'$.*
5. *If $e > e'$, then F is not a substring of F'.*

Proof. We analyze each case separately by aligning the occurrences of the Lyndon roots in F and F' in all possible ways and comparing their extensions; see Figure 3 for illustrative examples. This allows us to determine if no alignment of F is contained within F' (i.e., F is not a substring of F'), exactly one alignment is contained (unique substring), or multiple alignments are contained (not unique).

1. If $e < e' - 2$, we can align the first occurrence of the Lyndon root of F with either the second or third occurrence of the Lyndon root of F'. In both cases, both extensions of F coincide with complete occurrences of the Lyndon root of F', meaning that F is contained in both alignments and is therefore not a unique substring.
2. If $e = e' - 2$, there are three possible alignments of the Lyndon roots. In the second alignment, both extensions of F coincide with complete occurrences of the Lyndon root of F', so F is a substring of F'. In the first alignment, the left extensions of the runs are aligned, meaning that F is contained in this alignment unless $\alpha > \alpha'$. Similarly, in the third alignment, the same reasoning applies to the right extensions. Therefore, F is a unique substring of F' if and only if $\alpha > \alpha'$ and $\beta > \beta'$.
3. If $e = e' - 1$, there are two possible alignments. In the first alignment, the left extensions of the runs are aligned while the right extension of F coincides with a complete occurrence of the Lyndon root of F'; thus, F is contained in this alignment if $\alpha \leq \alpha'$. The same reasoning applies symmetrically for the second alignment. Therefore, F is a unique substring of F' if and only if either $\alpha > \alpha'$ or $\beta > \beta'$, but not both.
4. If $e = e'$, there is only one possible alignment of the Lyndon roots. For F to be a substring of F', both extensions of F must be shorter than or equal to those of F'; that is, $\alpha \leq \alpha'$ and $\beta \leq \beta'$.
5. If $e > e'$, any alignment will result in an extension of F' being aligned with a Lyndon root of F; therefore, F cannot possibly be a substring of F'. ◀

For the following definitions, let $\mathcal{R}_\lambda$ be the set of τ-runs in S sharing the same Lyndon root λ. Let $r := |\lambda|$, and let $e_\lambda^{\max}$ denote the maximum exponent among all τ-runs in $\mathcal{R}_\lambda$.

▶ **Definition 35** (Mapping $\mathfrak{h}$). *Let $R \in \mathcal{R}_\lambda$ be a run with $\mathsf{Lrepr}(R) = (\lambda, e, \alpha, \beta)$. We define the function $\mathfrak{h}$, which maps runs to sets of up to three points as follows:*

- *If $e < e_\lambda^{\max} - 2$, then $\mathfrak{h}(R) = \emptyset$;*
- *If $e = e_\lambda^{\max} - 2$, then $\mathfrak{h}(R) = \{(\alpha, \beta)\}$;*
- *If $e = e_\lambda^{\max} - 1$, then $\mathfrak{h}(R) = \{(r + \alpha, \beta), (\alpha, r + \beta)\}$;*
- *If $e = e_\lambda^{\max}$, then $\mathfrak{h}(R) = \{(2r - 1, \beta), (\alpha, 2r - 1), (r + \alpha, r + \beta)\}$.*

▶ **Definition 36** (Mapping $\mathfrak{g}$). *Let $p = (x, y)$ be an integer point in $[0, 2r - 1]^2$. We define the function $\mathfrak{g}$ as a mapping from such points to candidate runs as follows:*

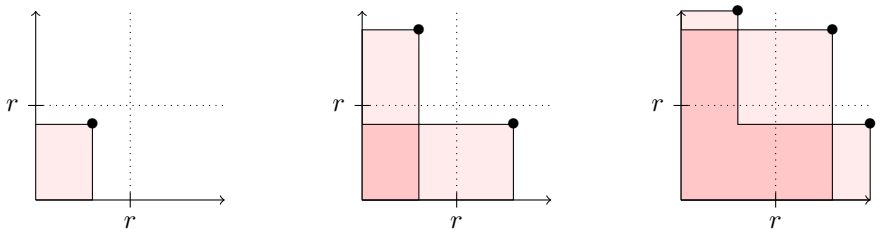
- *$\mathfrak{g}(p)$ has Lyndon root λ;*
- *The left and right extensions of $\mathfrak{g}(p)$ are $\alpha' = x \bmod r$ and $\beta' = y \bmod r$, respectively;*
- *The exponent e' of $\mathfrak{g}(p)$ is determined by the quadrant: $e' = e_\lambda^{\max}$, if ($x \geq r$ and $y \geq r$); $e' = e_\lambda^{\max} - 1$, if ($x \geq r$ and $y < r$) or ($x < r$ and $y \geq r$); $e' = e_\lambda^{\max} - 2$, if ($x < r$ and $y < r$).*

▶ **Example 37.** Let $\mathcal{R}_\lambda$ be a set of runs sharing the Lyndon root $\lambda = \mathtt{ab}$ with period $r = |\lambda| = 2$. Assume that $\mathcal{R}_\lambda$ consists of three runs, with a maximum exponent of $e_\lambda^{\max} = 6$:

- $\mathsf{Lrepr}(R_1) = (\mathtt{ab}, 6, 1, 1) \implies \mathfrak{h}(R_1) = \{(3, 1), (1, 3), (3, 3)\}$;
- $\mathsf{Lrepr}(R_2) = (\mathtt{ab}, 5, 0, 1) \implies \mathfrak{h}(R_2) = \{(2, 1), (0, 3)\}$;
- $\mathsf{Lrepr}(R_3) = (\mathtt{ab}, 4, 1, 0) \implies \mathfrak{h}(R_3) = \{(1, 0)\}$.

The geometric domain is $[0, 3]^2$, as $2r - 1 = 3$. Applying the mapping function $\mathfrak{h}$ to all runs in $\mathcal{R}_\lambda$ generates the point set $P = \{(3, 1), (1, 3), (3, 3), (2, 1), (0, 3), (1, 0)\}$.

Consider the point $(2, 2)$. Since $x \geq r$ and $y \geq r$ (specifically, $2 \geq 2$), the mapping $\mathfrak{g}(2, 2)$ assigns the exponent $e' = 6$ and extensions $\alpha' = 2 \bmod 2 = 0$ and $\beta' = 2 \bmod 2 = 0$. The mapping $\mathfrak{g}$ thus yields the run: $\mathfrak{g}(2, 2) = (\lambda, 6, 0, 0)$. This representation corresponds to the string $\lambda^6 = (\mathtt{ab})^6 = \mathtt{abababababab}$. Note that the point $(2, 2)$ lies on the primary



(a) $e = e_\lambda^{\max} - 2$. **(b)** $e = e_\lambda^{\max} - 1$. **(c)** $e = e_\lambda^{\max}$.

Figure 4 The mapping $\mathfrak{h}$ from runs to sets of points in $[0, 2r-1]^2$. The light shaded part represents the primary skyline, while dark shaded regions indicate areas dominated by multiple points, where no unique substring can exist.

skyline of P and is minimal in terms of $x + y$. This geometric property ensures that the corresponding run $\mathfrak{g}(2,2)$ belongs to exactly one periodic alignment among the runs in $\mathcal{R}_\lambda$ (specifically within R_1), making `abababababab` a valid SUS candidate.

▶ **Lemma 38.** *Let $R \in \mathcal{R}_\lambda$ be a run with $\mathsf{Lrepr}(R) = (\lambda, e, \alpha, \beta)$, and let $p = (x, y)$ be a point in the domain $[0, 2r-1]^2$. Then p is in the primary skyline of $\mathfrak{h}(R)$ if and only if $\mathfrak{g}(p)$ is a unique substring of R.*

Proof. Let $\mathfrak{g}(p) = R'$, where $\mathsf{Lrepr}(R') = (\lambda, e', \alpha', \beta')$. By the definition of $\mathfrak{g}$, we have $\alpha' = x \bmod r$ and $\beta' = y \bmod r$. The quadrant containing p determines the candidate exponent e', while the set of points $\mathfrak{h}(R)$ (and thus its primary skyline) is determined by the run exponent e; see Figure 4. We verify the correspondence with the cases in Lemma 34 by distinguishing how e compares to e'.

- **Case $e' = e$:** The point p can be dominated by exactly one point from $\mathfrak{h}(R)$ with relative coordinates (α, β). Hence, p belongs to the primary skyline if and only if it is dominated by this single point, which occurs when $\alpha' \le \alpha$ and $\beta' \le \beta$. This corresponds exactly to the condition for $\mathfrak{g}(p)$ to be a unique substring of R (Lemma 34: Case 4).
- **Case $e' = e - 1$:** In this case, p can be dominated by at most two points from $\mathfrak{h}(R)$. Point p is dominated by the first point if $\alpha' \le \alpha$ and by the second if $\beta' \le \beta$. Thus, p belongs to the primary skyline if it is dominated by exactly one of these points. This requires either $\alpha' \le \alpha$ or $\beta' \le \beta$ but not both. Again, this matches the condition for $\mathfrak{g}(p)$ to be a unique substring of R (Lemma 34: Case 3).
- **Case $e' = e - 2$:** This occurs only when $e = e_\lambda^{\max}$ and p is in the lower-left quadrant $(x, y < r)$. Here, p is always dominated by the point $(r + \alpha, r + \beta)$ of $\mathfrak{h}(R)$. For p to be in the primary skyline, it must *not* be dominated by the other two points $(2r - 1, \beta)$ and $(\alpha, 2r - 1)$. This lack of dominance occurs if and only if $\alpha' > \alpha$ and $\beta' > \beta$. This happens exactly when $\mathfrak{g}(p)$ is a unique substring of R (Lemma 34: Case 2). ◀

▶ **Lemma 39.** *Given a set $\mathcal{R}_\lambda$ of runs sharing the same Lyndon root λ, a SUS among these runs can be found by solving a Minimum Skyline Point instance containing at most $3|\mathcal{R}_\lambda|$ points. The corresponding set of (unsorted) points can be constructed in $\mathcal{O}(|\mathcal{R}_\lambda|)$ time.*

Proof. Let $e_\lambda^{\max}$ denote the maximum exponent among all runs in $\mathcal{R}_\lambda$. By Lemma 34, we know that any SUS within these runs must have an exponent equal to $e_\lambda^{\max} - 2$, $e_\lambda^{\max} - 1$ or $e_\lambda^{\max}$. Such a substring thus corresponds to $\mathfrak{g}(p)$ for some point $p \in [0, 2r-1]^2$, with $r = |\lambda|$.

By Lemma 38, p must lie on the primary skyline of $\mathfrak{h}(R)$ for some $R \in \mathcal{R}_\lambda$. Moreover, p cannot belong to the primary skylines of multiple runs; otherwise, $\mathfrak{g}(p)$ would not be unique overall. Hence, p must be dominated by exactly one point among all sets $\mathfrak{h}(R)$, which can be identified by solving the Minimum Skyline Point instance on the multiset $\bigsqcup_{R\in\mathcal{R}_\lambda} \mathfrak{h}(R)$. Since $|\mathfrak{h}(R)| \leq 3$ for every R, this instance has size $\mathcal{O}(|\mathcal{R}_\lambda|)$. Finally, if the resulting point from this instance is (x, y), it corresponds to a substring of length $r \cdot (e_\lambda^{\max} - 2) + x + y$. ◀

▶ **Lemma 40.** *Given an instance of Shortest Unique Substring and an integer τ, such that $\tau = \lfloor \frac{1}{15} \log_\sigma n \rfloor$ or $\tau = \lfloor \frac{1}{3} \log^4 n \rfloor$, we can compute a SUS of S in $\mathcal{O}(n/\log_\sigma n)$ time if it has length $\ell \geq 3\tau$ and period $p \leq \frac{1}{3}\tau$.*

Proof. By Lemma 10 and Lemma 33, all runs in S can be computed and grouped by their Lyndon roots in $\mathcal{O}(n/\log_\sigma n)$ time. Each τ-run is encoded in $\mathcal{O}(1)$ space. Next, using Lemma 39, we construct Minimum Skyline Point instances for all groups. Since each τ-run generates a constant number of points, the total number of points across all instances is $\mathcal{O}(n/\tau)$. These points are computed in $\mathcal{O}(n/\tau)$ time as per Lemma 10. To solve all Minimum Skyline Point instances in linear time (Lemma 21), the points for each $\mathcal{R}_\lambda$ instance must be sorted along one axis. This is achieved by globally sorting all generated points using bucket sort. Because the coordinates x and y for any point in an instance for root λ are bounded by $2|\lambda| - 1 < 2\tau$, the bucket sort can be performed in $\mathcal{O}(n/\tau + \tau)$ time. After sorting, the points are regrouped per their Lyndon roots. For either chosen value of τ, we have $\mathcal{O}(n/\tau + \tau) = \mathcal{O}(n/\log_\sigma n)$. This yields a total running time of $\mathcal{O}(n/\log_\sigma n)$. ◀

3.4 Putting It All Together

▶ **Theorem 1** (SUS). *Given a packed string S of length n over an integer alphabet $[0, \sigma)$, a shortest unique substring of S can be computed in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ time.*

Proof. We have described algorithms to find SUSs with differing lengths and periods: Lemma 15 handles any length up to $\frac{1}{5}\log_\sigma n$ with any period; Corollary 16 handles lengths in the range $[\frac{1}{5}\log_\sigma n, 2^{\sqrt{\log n}}]$ with period greater than $\frac{1}{45}\log_\sigma n$; and Lemma 32 handles lengths above $\log^4 n$ with period greater than $\frac{1}{9}\log^4 n$. Finally, with parameter values $\tau := \lfloor \frac{1}{15}\log_\sigma n \rfloor$ and $\tau := \lfloor \frac{1}{3}\log^4 n \rfloor$, Lemma 40 handles the larger length ranges with smaller periods. We run each of the algorithms and return the globally shortest substring output across all instances. The total running time is $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ which is asymptotically dominated by the complexity of the medium-length aperiodic case (Corollary 16). ◀

4 Computing a Shortest Absent Substring

We begin by defining an auxiliary problem closely related to Shortest Unique Substring.

Shortest Exclusive Substring
Input: Two strings S_1 and S_2 with $n = \|S_1\| + \|S_2\|$ over an integer alphabet $\Sigma = [0, \sigma)$.
Output: A shortest substring of S_1 that does not occur in S_2 (if one exists).

Similar to the solution for Shortest Unique Substring, we solve the Shortest Exclusive Substring problem by decomposing it into four cases based on the length ℓ and the period p of the sought substring of S_1, obtaining the following result.

▶ **Theorem 41** ([10]). *Any instance of Shortest Exclusive Substring can be solved in $\mathcal{O}(n \log \sigma/\sqrt{\log n})$ time when S_1 and S_2 are given in packed representation.*

The proof of Theorem 41 is deferred to [10]. We next present an efficient construction of de Bruijn sequences in the packed setting, which may be of independent interest.

▶ **Definition 42** (De Bruijn Sequence [14]). *A* de Bruijn sequence *of order k over an alphabet Σ of size σ is a string of length $\sigma^k + k - 1$ in which every string from Σ^k occurs exactly once.*

▶ **Lemma 43.** *A packed de Bruijn sequence of order k over the integer alphabet $[0, \sigma)$ for $\sigma \geq 2$ can be constructed in $\mathcal{O}(\frac{\sigma^k}{k}) = \mathcal{O}(\frac{n}{\log_\sigma n})$ time, where n is the length of the sequence. In particular, we can construct its prefix of length ℓ in $\mathcal{O}(\frac{\ell}{\log_\sigma n} + 1)$ time.*

Proof. A *Lyndon word* is a string that is lexicographically strictly smaller than all of its proper suffixes. As noted by Fredricksen and Maiorana [19], the concatenation of all Lyndon words whose length divides k, listed in lexicographical order, forms a de Bruijn sequence. Duval [15] provided an algorithm to generate all Lyndon words of length at most k in lexicographical order, generating each word by modifying the previous one.

We achieve the stated running time in the packed setting, by observing that $k \leq \lfloor \log_\sigma n \rfloor$, and using $\mathcal{O}(1)$ machine words to iterate over all Lyndon words in $[0, \sigma)^{\leq k}$.

In what follows, we describe Duval's algorithm [15] without proving its correctness; we only explain how it can be performed efficiently in the packed setting. We will return a string S, initialized as 0. We then maintain a length-k string w initialized as 0^k and a binary string w' whose ith bit is set if and only if $w[i] = \sigma - 1$ (we update w' together with w). We repeatedly apply the following steps of the generation loop while $w[0] \neq \sigma - 1$:

1. Locate $j = \max\{i : w[i] \neq \sigma - 1\}$. This is done by finding the rightmost 0 in w' in $\mathcal{O}(1)$ time using standard bitwise operations.
2. Replace $w[j]$ with $w[j] + 1$ in $\mathcal{O}(1)$ time.
3. The prefix $w[0\,..\,j]$ is now a Lyndon word. If $(j + 1)$ divides k, we append $w[0\,..\,j]$ to S. This takes $\mathcal{O}(1)$ time as the appended string fits into $\mathcal{O}(1)$ machine words.
4. Replace w with the length-k prefix of $(w[0\,..\,j])^\infty$. This can be naively performed using $\mathcal{O}(k/(j+1))$ operations. Consider a potential function $\pi(w) = |\{i : w[i] = \sigma - 1\}|$, noting that Step 2 increases $\pi(w)$ by at most 1. Conversely, in Step 4, values $w[i] = \sigma - 1$ for $i > j$ are replaced with $w[i \bmod (j+1)]$. Since $w[0] < \sigma - 1$ (the algorithm terminates when $w[0]$ reaches $\sigma - 1$), $\Omega(k/(j + 1))$ copies of $\sigma - 1$ are overwritten by setting $w = (w[0\,..\,j])^\infty$. As $\pi(w) \geq 0$ at all times, the total number of operations we perform in Step 4 is asymptotically upper bounded by the number of times Step 2 is executed, and hence the amortized running time of Step 4 is $\mathcal{O}(1)$.

After the last iteration, we complete the sequence by appending 0^{k-1}. Observe that, for each integer ℓ there are at most $\frac{\sigma^\ell}{\ell}$ Lyndon words of length ℓ, and hence at most $\sum_{\ell \leq k} \frac{\sigma^\ell}{\ell} \leq 3\frac{\sigma^k}{k}$ Lyndon words of length at most k. This proves that there are $\mathcal{O}(\frac{\sigma^k}{k})$ iterations of the loop, and the algorithm takes $\mathcal{O}(\frac{n}{\log_\sigma n})$ time (where $n = \sigma^k + k - 1$ and $k \leq \lfloor \log_\sigma n \rfloor$) as claimed. To output a length-ℓ prefix of this sequence we terminate the algorithm once the length of the output sequence reaches ℓ, potentially removing up to $k - 1$ excessive letters from the final Lyndon word. Since $k/\log_\sigma n = \mathcal{O}(1)$, the total time complexity is

$$\mathcal{O}\left(\frac{\ell + k}{\log_\sigma n} + 1\right) = \mathcal{O}\left(\frac{\ell}{\log_\sigma n} + 1\right). \qquad \blacktriangleleft$$

▶ **Theorem 2** (SAS). *Given a packed string S of length n over an integer alphabet $[0, \sigma)$, a shortest absent substring of S can be computed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time.*

Proof. Let $k := \lfloor \log_\sigma n \rfloor + 1$. Since $\sigma^k > n$, the length of any SAS is at most k. We first check for a SAS of length at most $k - 1$ by applying Theorem 41 to a packed de Bruijn sequence S_1 of order $k - 1$ (Lemma 43) and $S_2 := S$. If this yields a substring of length

at most $k-1$, we are done. Otherwise, if $\sigma^k = \mathcal{O}(n)$, we create a full packed de Bruijn sequence of order k and repeat the process to find a SAS of length k. If $\sigma^k = \omega(n)$, we instead generate a prefix S'_1 of a de Bruijn sequence of order k of length $n+1$ in $\mathcal{O}(\frac{n}{\log_\sigma n})$ time. S'_1 contains $n-k+2$ distinct substrings of length k. Since S_2 has at most $n-k+1$ such substrings, at least one substring of S'_1 must be absent from S_2. We find this witness of length k using Theorem 41 for S'_1 and S_2. The calls to Theorem 41 dominate the total running time, which is $\mathcal{O}\left(\frac{n\log\sigma}{\sqrt{\log n}}\right)$. ◀

References

1 Ricardo A. Baeza-Yates. Improved string searching. *Softw. Pract. Exp.*, 19(3):257–271, 1989. doi:10.1002/SPE.4380190305.

2 Hideo Bannai and Jonas Ellert. Lyndon arrays in sublinear time. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.14.

3 Djamal Belazzougui. Worst case efficient single and multiple string matching in the RAM model. In *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010*, volume 6460 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2010. doi:10.1007/978-3-642-19222-7_10.

4 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Towards optimal packed string matching. *Theoretical Computer Science*, 525:111–129, 2014. doi:10.1016/j.tcs.2013.06.013.

5 Philip Bille. Fast searching in packed strings. *Journal of Discrete Algorithms*, 9(1):49–56, 2011. doi:10.1016/j.jda.2010.09.003.

6 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 6:1–6:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.6.

7 Dany Breslauer, Leszek Gasieniec, and Roberto Grossi. Constant-time word-size string matching. In *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Proceedings*, volume 7354 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 2012. doi:10.1007/978-3-642-31265-6_7.

8 Panagiotis Charalampopoulos, Maxime Crochemore, Gabriele Fici, Robert Mercas, and Solon P. Pissis. Alignment-free sequence comparison using absent words. *Inf. Comput.*, 262:57–68, 2018. doi:10.1016/J.IC.2018.06.002.

9 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, and Solon P. Pissis. Faster algorithms for longest common substring. *ACM Trans. Algorithms*, 2025. doi:10.1145/3774754.

10 Panagiotis Charalampopoulos, Manal Mohamed, Solon P. Pissis, Hilde Verbeek, and Wiktor Zuba. Faster algorithms for unique or absent substrings. *CoRR*, 2026. doi:10.48550/arXiv.2605.04826.

11 Panagiotis Charalampopoulos, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Counting distinct square substrings in sublinear time. In *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025*, volume 345 of *LIPIcs*, pages 36:1–36:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.MFCS.2025.36.

12 Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest palindromic substring in sublinear time. In *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022*, volume 223 of *LIPIcs*, pages 20:1–20:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CPM.2022.20.

13 Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Sergio Salemi. Data compression using antidictionaries. *Proc. IEEE*, 88(11):1756–1768, 2000. doi:10.1109/5.892711.

14 Nicolaas Govert de Bruijn. A combinatorial problem. *Proc. Koninklijke Nederlandse Akademie V. Wetenschappen*, 49:758–764, 1946. URL: http://resolver.tudelft.nl/uuid:f9a56551-402a-4428-98e6-1469e38e64c1.

15 Jean-Pierre Duval. Génération d'une section des classes de conjugaison et arbre des mots de lyndon de longueur bornée. *Theor. Comput. Sci.*, 60:255–283, 1988. doi:10.1016/0304-3975(88)90113-2.

16 Jonas Ellert. Sublinear time Lempel-Ziv (LZ77) factorization. In *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Proceedings*, volume 14240 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2023. doi:10.1007/978-3-031-43980-3_14.

17 Jonas Ellert and Tomasz Kociumaka. Time-optimal construction of string synchronizing sets. In *43rd International Symposium on Theoretical Aspects of Computer Science, STACS 2026*, LIPIcs, pages 36:1–36:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.STACS.2026.36.

18 Johannes Fischer and Paweł Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Proceedings*, volume 9133 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2015. doi:10.1007/978-3-319-19929-0_14.

19 Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de bruijn sequences. *Discret. Math.*, 23(3):207–210, 1978. doi:10.1016/0012-365X(78)90002-X.

20 Kimmo Fredriksson. Shift-or string matching with super-alphabets. *Information Processing Letters*, 87(4):201–204, 2003. doi:10.1016/S0020-0190(03)00296-5.

21 Szymon Grabowski and Kimmo Fredriksson. Bit-parallel string matching under hamming distance in o(n[m/w]) worst case time. *Inf. Process. Lett.*, 105(5):182–187, 2008. doi:10.1016/J.IPL.2007.08.021.

22 Bernhard Haubold, Nora Pierstorff, Friedrich Möller, and Thomas Wiehe. Genome comparison without alignment using shortest unique substrings. *BMC Bioinform.*, 6:123, 2005. doi:10.1186/1471-2105-6-123.

23 Costas S. Iliopoulos, Dennis W. G. Moore, and William F. Smyth. A characterization of the squares in a fibonacci string. *Theor. Comput. Sci.*, 172(1-2):281–291, 1997. doi:10.1016/S0304-3975(96)00141-7.

24 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001*, volume 2089 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2001. doi:10.1007/3-540-48194-X_17.

25 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

26 Dominik Kempa and Tomasz Kociumaka. Breaking the $O(n)$-barrier in the construction of compressed suffix arrays and suffix trees. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5122–5202. SIAM, 2023. doi:10.1137/1.9781611977554.CH187.

27 Dominik Kempa and Tomasz Kociumaka. Lempel-Ziv (LZ77) factorization in sublinear time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 2045–2055. IEEE, 2024. doi:10.1109/FOCS61266.2024.00122.

28 Dominik Kempa and Tomasz Kociumaka. On the hardness hierarchy for the $o(n\sqrt{\log n})$ complexity in the word RAM. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 290–300. ACM, 2025. doi:10.1145/3717823.3718291.

29 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theor. Comput. Sci.*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.

30 Gonzalo Navarro and Mathieu Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Proceedings*, volume 1448 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1998. doi:10.1007/BFB0030778.

31 Jian Pei, Wush Chi-Hsuan Wu, and Mi-Yen Yeh. On shortest unique substring queries. In *29th IEEE International Conference on Data Engineering, ICDE 2013*, pages 937–948. IEEE Computer Society, 2013. doi:10.1109/ICDE.2013.6544887.

32 Diogo Pratas and Jorge Miguel Silva. Persistent minimal sequences of SARS-CoV-2. *Bioinform.*, 36(21):5129–5132, 2021. doi:10.1093/BIOINFORMATICS/BTAA686.

33 Jakub Radoszewski and Wiktor Zuba. Computing string covers in sublinear time. In *String Processing and Information Retrieval - 31st International Symposium, SPIRE 2024, Proceedings*, volume 14899 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2024. doi:10.1007/978-3-031-72200-4_21.

34 Raquel M. Silva, Diogo Pratas, Luísa Castro, Armando J. Pinho, and Paulo Jorge S. G. Ferreira. Three minimal sequences found in ebola virus genomes and absent from human DNA. *Bioinform.*, 31(15):2421–2425, 2015. doi:10.1093/BIOINFORMATICS/BTV189.

35 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

36 Takuya Takagi, Shunsuke Inenaga, Kunihiko Sadakane, and Hiroki Arimura. Packed compact tries: A fast and efficient data structure for online string processing. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 100-A(9):1785–1793, 2017. doi:10.1587/TRANSFUN.E100.A.1785.

37 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

Orthogonal Strip Partitioning of Polygons: Lattice-Theoretic Algorithms and Lower Bounds

Jaehoon Chung
Korea Institute for Advanced Study (KIAS), Seoul, Republic of Korea

Abstract

We study a variant of a polygon partition problem, introduced by Chung, Iwama, Liao, and Ahn [ISAAC'25]. Given orthogonal unit vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$ and a polygon P with n vertices, we partition P into connected pieces by cuts parallel to $\mathbf{v}$ such that each resulting subpolygon has width at most one in direction $\mathbf{u}$. We consider the value version, which asks for the minimum number of strips, and the reporting version, which outputs a compact encoding of the cuts in an optimal strip partition.

We give efficient algorithms and lower bounds for both versions on three classes of polygons of increasing generality: convex, simple, and self-overlapping. For convex polygons, we solve the value version in $O(\log n)$ time and the reporting version in $O\big(h \log\big(1 + \frac{n}{h}\big)\big)$ time, where h is the width of P in direction $\mathbf{u}$. We prove matching lower bounds in the decision-tree model, showing that the reporting algorithm is input-sensitive optimal with respect to h. For simple polygons, we present $O(n \log n)$-time, $O(n)$-space algorithms for both versions and prove an $\Omega(n)$ lower bound. For self-overlapping polygons, we extend the approach for simple polygons to obtain $O(n \log n)$-time, $O(n)$-space algorithms for both versions, and we prove a matching $\Omega(n \log n)$ lower bound in the algebraic computation-tree model via a reduction from the δ-closeness problem.

Our approach relies on a lattice-theoretic formulation of the problem. We represent strip partitions as antichains of intervals in the Clarke–Cormack–Burkowski lattice, originally developed for minimal-interval semantics in information retrieval. Within this lattice framework, we design a dynamic programming algorithm that uses the lattice operations of meet and join. To the best of our knowledge, this is the first geometric application of the Clarke–Cormack–Burkowski lattice.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Polygon partitioning, Strip partition, Lattice, Self-overlapping curves

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.14

Related Version *Full Version*: `https://arxiv.org/abs/2604.15247`

Funding This work was supported in part by a KIAS Individual Grant AP106101 via the Center for Artificial Intelligence and Natural Sciences at Korea Institute for Advanced Study, and by the Center for Advanced Computation at Korea Institute for Advanced Study.

1 Introduction

Partitioning a polygon into the minimum number of pieces has been studied extensively under shape-class constraints, for example into convex, star-shaped, quadrilateral, or monotone pieces [1, 10, 15]. Much less is known for metric constraints that bound geometric measures, such as width, diameter, or perimeter. In contrast to the classical shape-based setting, the problem becomes significantly harder under such metric bounds. Imposing a unit upper bound on diameter or perimeter already makes the minimum partition problem NP-hard for simple polygons without holes [3]. Moreover, no polynomial-time algorithm is known for computing an optimal solution even when the input is a square or an equilateral triangle [2].

In ISAAC'25, Chung et al. [11] introduced a minimum partition problem with an upper-bounded width constraint, which limits the width of each partitioned piece in prescribed directions. Such a constraint is motivated by manufacturing and recycling, where materials must be cut or processed within width limits. For instance, Lan et al. [16] recycle wind

© Jaehoon Chung;
licensed under Creative Commons License CC-BY 4.0

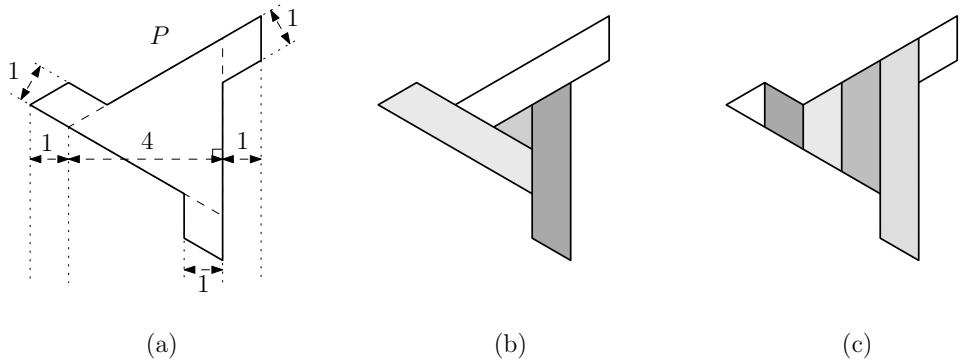


Figure 1 (a) The polygon P has a windmill shape with three arms extending from an equilateral triangle of height 4. (b) A minimum partition of P under width and cut constraints, $W = U = \mathbb{S}^+$. (c) A minimum orthogonal strip partition of P with $\mathbf{u} = (1, 0), \mathbf{v} = (0, 1)$.

turbine blades by first cutting the blades into panels small enough to fit the intake of crushing machines, and then crushing them into recyclates. Moreover, wind turbine blades are typically made of glass-fiber composites with directionally aligned fibers, so the cutting effort can depend on the cut direction [13].

From a theoretical perspective, the problem is related to Bang's conjecture, a long-standing open problem in convex geometry [5]. Chung et al. resolved a partition analogue of this conjecture, showing that for any convex body, there exists a direction in which an optimal partition can be achieved using only parallel cuts.

These applications and theoretical results motivate the study of minimum partitioning under an upper-bounded width constraint. Moreover, they suggest that it is natural to restrict attention to partitions in which all cuts are parallel to a single direction, which we call the *strip partition model*.

We consider three classes of input polygons: convex, simple, and self-overlapping, where a self-overlapping polygon generalizes a simple polygon by allowing the boundary to self-intersect. A *self-overlapping curve* was introduced by Shor and van Wyk [21] as the boundary of a topological disk immersed in $\mathbb{R}^2$. When this curve is polygonal, it serves as the boundary of a self-overlapping polygon. Unlike in the simple case, the boundary alone does not determine a unique interior or visibility relation. Following Shor and van Wyk, we define visibility with respect to a triangulation of the curve, so that partitions of the polygon are well defined.

In this work, we study orthogonal strip partitioning of polygons and present efficient algorithms with lower bounds for each class of input polygons.

Orthogonal strip partition problem. We first recall the partitioning problem under width and cut constraints introduced by Chung et al. [11]. Let P be a simple polygon, and let $\mathbb{S}^+ = \{(\cos\theta, \sin\theta) \mid 0 \le \theta < \pi\}$ denote the set of all unit vectors in the plane. For $\mathbf{v} \in \mathbb{S}^+$ and a set $X \subset \mathbb{R}^2$, let $\omega_{\mathbf{v}}(X)$ denote the width of X in direction $\mathbf{v}$, that is, the length of the projection of X onto a line parallel to $\mathbf{v}$. Given a subset $W \subseteq \mathbb{S}^+$, we say that X satisfies the *unit-width constraint* W if $\omega_{\mathbf{v}}(X) \le 1$ for some $\mathbf{v} \in W$. We also impose a *cut constraint* $U \subseteq \mathbb{S}^+$, meaning that every cut used to partition P must be aligned with some direction in U. Here, a cut is a line segment $c \subseteq P$ whose relative interior lies in the interior of P. Given a simple polygon P and sets $W, U \subseteq \mathbb{S}^+$, the goal is to find a partition of P into the minimum number of pieces such that every cut is aligned with a direction in U and each piece satisfies the unit-width constraint W.

The *orthogonal strip partition problem* is the special case with $W = \{\mathbf{u}\}$ and $U = \{\mathbf{v}\}$ for two orthogonal unit vectors $\mathbf{u}, \mathbf{v} \in \mathbb{S}^+$. A *strip* is a subpolygon $Q \subseteq P$ with $\omega_{\mathbf{u}}(Q) \leq 1$, and a strip partition divides P into strips using only cuts parallel to $\mathbf{v}$. Without loss of generality, we assume $\mathbf{u} = (1, 0)$ and $\mathbf{v} = (0, 1)$. Under this assumption, each cut is a vertical segment and every strip Q has width at most 1 in the horizontal direction. In this orthogonal setting, we regard a cut as a maximal vertical segment contained in P whose relative interior lies in the interior of P. See Figure 1 for an illustration.

Cut descriptors and lossless encoding. We define the *descriptor* of a cut as the pair of boundary edges of P on which its top and bottom endpoints lie, together with its x-coordinate. If a top or bottom endpoint lies on two incident edges, we choose the left edge, so the descriptor uniquely determines the position of the cut. A partition of P can be represented by the descriptors of its cuts. Our primary objective is to minimize the number of strips. In the reporting variant, we also compute the descriptors of all cuts in an optimal partition.

To store these descriptors succinctly, we fix a *lossless encoding* scheme for strip partitions. For each polygon P, the scheme assigns a codeword to each strip partition of P so that the partition can be reconstructed from the codeword without loss. Thus each strip partition of P is represented by a unique codeword.

The purpose of lossless encoding is to separate the combinatorial difficulty of finding an optimal partition from the cost of listing all cuts. For an integer $k \geq 1$, the optimal strip partition of $k \times 1$ rectangle consists of $k - 1$ cuts spaced one unit apart. If we store one descriptor per cut, any reporting algorithm requires $\Omega(k)$ time to output the explicit list. We therefore allow the algorithm to output a succinct codeword from which the full list of cut descriptors can be reconstructed without loss, for example by encoding the x-coordinate of the leftmost cut together with the number of cuts. This is called a *run-length* encoding [6]. Under this model, the cost is no longer dominated by explicit output size, but by distinguishing among combinatorially different optimal partitions.

▶ **Definition 1** (Orthogonal Strip Partition Problems). *An orthogonal strip partition of a polygon $P \subset \mathbb{R}^2$ is a partition of P into subpolygons (strips) using vertical cuts, such that each subpolygon has horizontal width at most 1. We consider three input domains $\mathcal{D} \in \{$convex, simple, self-overlapping$\}$. For $\mathcal{D} \in \{$convex, simple$\}$, an input instance consists of the vertices of P listed in counterclockwise order along its boundary. For $\mathcal{D} =$ self-overlapping, an instance consists of a triangulation of P. For each domain $\mathcal{D}$, we consider two versions:*

- Value version: *return the minimum number of strips.*
- Reporting version: *with respect to the fixed lossless encoding scheme, output any codeword representing an optimal strip partition of P.*

Let $\mathcal{D} \in \{\text{conv}, \text{simp}, \text{self}\}$ denote the type of P, corresponding to convex, simple, and self-overlapping, respectively. For a polygon P in domain $\mathcal{D}$, let $\text{OSP}_V^{\mathcal{D}}(P)$ and $\text{OSP}_R^{\mathcal{D}}(P)$ be the value and reporting versions of the orthogonal strip partition problem, respectively. We denote by $\mathsf{opt}(P)$ the minimum number of strips in an optimal strip partition of P.

Related work. The general version of the strip partition problem with arbitrary width and cut constraints $W, U \subseteq \mathbb{S}^+$, was first introduced by Chung et al. [11]. They studied structural properties in simple polygons, in particular the monotonicity of the minimum partition number under polygon containment. They proved a partition analogue of Bang's conjecture. If U contains every direction orthogonal to each $\mathbf{w} \in W$, then any convex body admits an optimal partition obtained by equally spaced cuts parallel to some direction $\mathbf{v} \in U$. Hence an optimal solution exists in the strip partition model.

Most algorithmic work for the strip partition model focuses on monotone variants, including the convex case. Liu [19] gave an $O(n \log n)$-time algorithm for partitioning a rectilinear polygon with rectilinear holes into x-monotone pieces using horizontal cuts, where n is the total number of vertices. Lee et al. [17] improved this to linear time for hole-free polygons. Lingas and Soltan [18] studied minimum convex partitions of polygons with holes by cuts restricted to a set U of directions; in particular, they give an $O(n \log n)$-time algorithm when $|U| = 1$ (parallel cuts).

Our results. We study orthogonal strip partitioning of convex, simple, and self-overlapping polygons, and give efficient algorithms with matching lower bounds in appropriate computational models, in particular the decision-tree and algebraic computation-tree models.

First, for a convex n-gon, the value version can be solved in $O(\log n)$ time and the reporting version in $O\big(h \log(1 + n/h)\big)$ time, where $h \geq 1$ is the horizontal width of the input polygon. As a function of h, this bound is $O(\log n)$ when $h = 1$ and approaches $O(n)$ as h increases. We prove matching lower bounds in the decision-tree model, showing that any algorithm needs $\Omega(\log n)$ time for the value version and $\Omega\big(h \log(1 + n/h)\big)$ time for the reporting version. Thus the reporting algorithm is input-sensitive optimal with respect to h.

Second, for simple polygons with n vertices, we use a lattice-theoretic formulation of strip partitions. We represent strip partitions as antichains of intervals in the Clarke–Cormack–Burkowski lattice, originally developed for minimal-interval semantics in information retrieval, and design a dynamic programming algorithm on a trapezoidal decomposition whose recurrence is expressed using the lattice operations of meet and join on these antichains. This yields an $O(n \log n)$-time, $O(n)$-space algorithm that solves both the value and reporting versions. We also show that, unlike for convex polygons, any algorithm requires $\Omega(n)$ accesses to the coordinates of the input vertices in the worst case. To the best of our knowledge, this is the first geometric application of the Clarke–Cormack–Burkowski lattice.

Finally, we extend this lattice-based framework to self-overlapping polygons given by a triangulation. We obtain an $O(n \log n)$-time, $O(n)$-space algorithm that solves both versions, and prove an $\Omega(n \log n)$ lower bound in the algebraic computation-tree model by a reduction from the δ-closeness problem. This shows that our algorithm is optimal in that model.

Because of page limits, the main body of the paper focuses on the lattice formulation (Section 3) and dynamic program (Section 4) for simple polygons. Omitted proofs and full details for the convex and self-overlapping cases are deferred to the full version.

2 Preliminaries

Let P be a polygon with n vertices in the plane, given as a list of vertices in counterclockwise order along its boundary. A *partition* of P is a set of connected pieces with pairwise disjoint interiors whose union equals P. The *cardinality* of a partition is the number of its pieces.

For a set $X \subseteq \mathbb{R}^2$, we denote by ∂X its boundary, by $\mathsf{int}(X)$ its interior, and by $\mathsf{cl}(X)$ its closure. We regard a polygon as the union of its interior and boundary; in particular, $\mathsf{cl}(P) = P$, ∂P is the boundary of P, and $\mathsf{int}(P)$ is its interior.

For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ denote its x- and y-coordinates, respectively. For any two points $p, q \in \mathbb{R}^2$, we use pq to denote the line segment connecting p and q, and write $|pq|$ for its length. We call pq a *cut* in P if $p, q \in \partial P$ and the interior points of pq lie in $\mathsf{int}(P)$. If ℓ is a vertical cut, we denote by $x(\ell)$ the x-coordinate of ℓ.

Let O be a totally ordered set. A subset $I \subseteq O$ is an *interval* of O if for all $x, y, z \in O$ with $x \leq z \leq y$, the condition $x, y \in I$ implies $z \in I$. A *closed interval* of O is an interval of the form $[a, b] = \{x \in O \mid a \leq x \leq b\}$. We denote by $\mathbf{I}_O$ the set of all closed intervals of O.

If $O \subseteq \mathbb{R}$ with the induced order, then for a closed interval $I = [a, b] \in \mathbf{I}_O$, we define its length by $|I| \coloneqq b - a$. The set O is *locally finite* if every closed interval $[a, b] \subseteq O$ contains only finitely many elements of O; for example, $\mathbb{Z}$ is locally finite, whereas $\mathbb{Q}$ and $\mathbb{R}$ are not.

We use the notation $[m] \coloneqq \{1, 2, \ldots, m\}$ for a positive integer m. For a finite set A, we use $|A|$ to denote its cardinality.

Computational model. We work in the real-RAM model, which supports unit-cost arithmetic $(+, -, \times, \div)$ and comparisons on real numbers. The n vertices of the input polygon P are given by real coordinates $(x_i, y_i)_{i=1}^n$. We allow a restricted use of floor function: for each input x_i, the integer part $\lfloor x_i \rfloor$ is also given and can be accessed in constant time, which increases the input size only by a constant factor. We do not allow the floor function on arbitrary real values, since this would make the model unrealistically powerful [20]. The integer labels do not increase the computational power for the problems we consider.

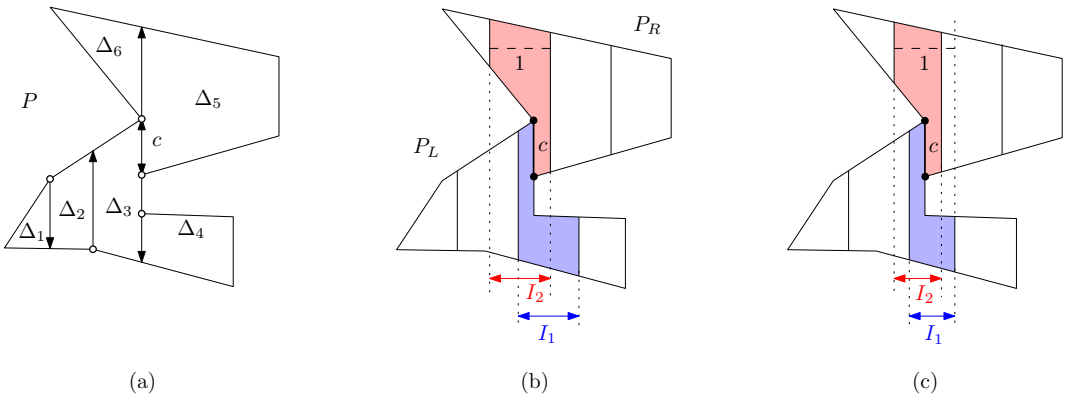
Our algorithms take floor/ceiling of differences between x-coordinates of input vertices. In our model, these are supported in constant time because the integer parts of all input x-coordinates are given. Let $(x_1, x_2, \ldots, x_n)$ be the x-coordinates of the vertices of P. For each $i \in [n]$, let $f_i = \lfloor x_i \rfloor$ (the integer part) and $r_i = x_i - f_i$ (the fractional part). Then, $\lfloor x_i - x_j \rfloor = f_i - f_j - \mathbf{1}[r_i < r_j]$, where $\mathbf{1}[\cdot]$ is the indicator function and its value is computed by a single comparison. For computing ceilings, we use the identity $\lceil x_i - x_j \rceil = -\lfloor x_j - x_i \rfloor$.

3 Lattice formulation for orthogonal strip partitions of simple polygons

In this section, P is a simple polygon with n vertices, stored in an array in counterclockwise order along its boundary. We give an $O(n \log n)$-time, $O(n)$-space algorithm for both $\mathrm{OSP}_V^{\mathrm{simp}}(P)$ and $\mathrm{OSP}_R^{\mathrm{simp}}(P)$.

We first compute a vertical trapezoidal decomposition of P and its dual graph in $O(n)$ time using the algorithm of Chazelle [9]; see Figure 2(a). Let $V = \{\Delta_1, \Delta_2, \ldots, \Delta_m\}$ be the set of trapezoids and let $G = (V, E)$ be its dual graph. Then $m = O(n)$ and G is a tree.

Let c be a vertical cut in the trapezoidal decomposition. Cutting P along c yields two subpolygons $P_L, P_R \subset P$, where P_L is the part of P lying to the left of c and P_R is the part lying to the right. Consider optimal strip partitions of P_L and P_R in which every strip has horizontal width at most 1. Taking the union of these two partitions, with c as a



(a) (b) (c)

Figure 2 (a) Vertical trapezoidal decomposition of P into $V = \{\Delta_1, \ldots, \Delta_6\}$, and decomposition of P into P_L and P_R by a vertical cut c of V. (b-c) Merging optimal strip partitions of P_L and P_R incident to c may or may not require an additional cut, depending on whether $|I_1 \cup I_2| \leq 1$ or > 1.

common boundary, yields a feasible strip partition of P, implying $\mathsf{opt}(P) \leq \mathsf{opt}(P_L)+\mathsf{opt}(P_R)$. Conversely, given an optimal partition of P, restricting it to P_L and P_R yields feasible strip partitions of these subpolygons. Thus, $\mathsf{opt}(P_L)+\mathsf{opt}(P_R)-1 \leq \mathsf{opt}(P) \leq \mathsf{opt}(P_L)+\mathsf{opt}(P_R)$, and we need to determine whether the lower or upper bound is achieved in a given instance.

To analyze this, observe that in any strip partition of P_L, there is exactly one strip incident to c. For P_L, let $\mathcal{I}_c(P_L)$ denote the set of x-intervals obtained as follows: for each optimal strip partition of P_L, take the strip incident to c and project it onto the x-axis, and include the resulting interval in $\mathcal{I}_c(P_L)$. We define $\mathcal{I}_c(P_R)$ for P_R analogously. Every interval in $\mathcal{I}_c(P_L)$ and in $\mathcal{I}_c(P_R)$ contains $x(c)$, since its corresponding strip is incident to c.

When gluing optimal partitions of P_L and P_R along c, the total number of strips decreases by one only when the two strips incident to c can be merged into a single strip without violating the width constraint. In terms of x-intervals, this is equivalent to checking whether there exist intervals $I_1 \in \mathcal{I}_c(P_L)$ and $I_2 \in \mathcal{I}_c(P_R)$ such that $|I_1 \cup I_2| \leq 1$. See Figure 2.

▶ **Observation 2.** *We have $\mathsf{opt}(P) = \mathsf{opt}(P_L)+\mathsf{opt}(P_R)-1$ if and only if there exist intervals $I_1 \in \mathcal{I}_c(P_L)$ and $I_2 \in \mathcal{I}_c(P_R)$ such that $|I_1 \cup I_2| \leq 1$.*

Moreover, it suffices to consider only inclusion-minimal intervals in each family $\mathcal{I}_c(P_L)$ and $\mathcal{I}_c(P_R)$. If $(I_1, I_2) \in \mathcal{I}_c(P_L) \times \mathcal{I}_c(P_R)$ satisfies $|I_1 \cup I_2| \leq 1$ and there exists $I_1' \in \mathcal{I}_c(P_L)$ with $I_1' \subsetneq I_1$, then $|I_1' \cup I_2| \leq 1$ also holds; by symmetry, the same argument applies to $\mathcal{I}_c(P_R)$. Thus, we may restrict attention to inclusion-minimal intervals in $\mathcal{I}_c(P_L)$ and $\mathcal{I}_c(P_R)$.

▶ **Observation 3.** *To decide whether there exist $I_1 \in \mathcal{I}_c(P_L)$ and $I_2 \in \mathcal{I}_c(P_R)$ with $|I_1 \cup I_2| \leq 1$, it suffices to consider only the inclusion-minimal intervals in $\mathcal{I}_c(P_L)$ and $\mathcal{I}_c(P_R)$.*

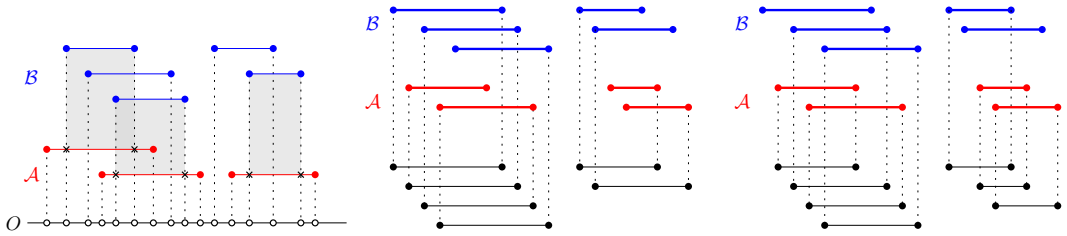
Algorithm overview and lattice formulation. Our algorithm performs a bottom-up dynamic program over the vertical trapezoidal decomposition of P and its dual tree G, rooted at an arbitrary node. For a node u of G, let Q be the subpolygon formed by the union of trapezoids in the subtree rooted at u. If u is not the root, we designate a vertical edge c of Q that serves as the interface to its parent. The associated subproblem for (Q, c) is to determine the minimum number $\mathsf{opt}(Q)$, together with the possible x-intervals of strips incident to c in optimal partitions of Q.

Observation 2 shows that, when two subpolygons are glued across c, the optimum of the combined region can be determined from the optimum of each subpolygon and the x-intervals of the strips incident to c in optimal partitions of the two subpolygons. Moreover, by Observation 3, it suffices to retain only the inclusion-minimal such intervals. Thus, for each subproblem (Q, c), we maintain the optimal value $\mathsf{opt}(Q)$ together with the inclusion-minimal intervals in $\mathcal{I}_c(Q)$.

We initialize these DP states at leaf trapezoids and combine them bottom-up at internal nodes, eventually reaching the root and obtaining $\mathsf{opt}(P)$. The crucial observation is that the update from child subproblems to a parent subproblem can be carried out entirely on these interval families. These interval families form an order-theoretic structure, namely the Clarke–Cormack–Burkowski (CCB) lattice. This lattice comes with two basic operations, meet($\wedge$) and join($\vee$). As we show next, the two basic operations naturally describe the two update situations of the dynamic program: gluing two incident strips into one, or introducing a new cut when such a gluing would violate the unit-width constraint.

3.1 Antichain completion and the Clarke–Cormack–Burkowski lattice

Let $(X, \leq)$ be a partially ordered set (poset). A subset $A \subseteq X$ is an *antichain* if no two distinct elements of A are comparable, that is, for all $a, b \in A$ with $a \neq b$, we have neither $a \leq b$ nor $b \leq a$. We denote by $\mathsf{AC}(X)$ the set of all antichains of X.



(a) $\mathcal{A} \preceq \mathcal{B}$ (b) $\mathcal{A} \wedge \mathcal{B}$ (c) $\mathcal{A} \vee \mathcal{B}$

Figure 3 Let $(\mathcal{E}_O, \preceq)$ be the antichain completion of $(\mathbf{I}_O, \supseteq)$. The figure illustrates the partial order, meet, and join operations for two antichains $\mathcal{A}, \mathcal{B} \in \mathcal{E}_O$: (a) $\mathcal{A} \preceq \mathcal{B}$; (b) $\mathcal{A} \wedge \mathcal{B}$; (c) $\mathcal{A} \vee \mathcal{B}$.

Boldi and Vigna [8] order antichains by their lower sets. For $\mathcal{A} \in \mathsf{AC}(X)$, its *lower set* is $\downarrow(\mathcal{A}) := \{x \in X \mid \exists y \in \mathcal{A} \text{ with } x \le y\}$. They define a partial order $\preceq$ on $\mathsf{AC}(X)$ by $\mathcal{A} \preceq \mathcal{B} \iff \downarrow(\mathcal{A}) \subseteq \downarrow(\mathcal{B})$. The poset $(\mathsf{AC}(X), \preceq)$ is called the *antichain completion* of $(X, \le)$.

Now let O be a totally ordered set, and let $\mathbf{I}_O$ be the set of all closed intervals of O. We order $\mathbf{I}_O$ by reverse inclusion, so $(\mathbf{I}_O, \supseteq)$ is a poset. We write $\mathcal{E}_O := \mathsf{AC}(\mathbf{I}_O)$ and view $(\mathcal{E}_O, \preceq)$ as the antichain completion of $(\mathbf{I}_O, \supseteq)$. For our purposes, it suffices to keep in mind the following description of $\mathcal{E}_O$ and $\preceq$. Each element $\mathcal{A} \in \mathcal{E}_O$ is a family of intervals in which no interval properly contains another. For $\mathcal{A}, \mathcal{B} \in \mathcal{E}_O$, the relation $\mathcal{A} \preceq \mathcal{B}$ means that every interval in $\mathcal{A}$ contains at least one interval in $\mathcal{B}$.

Clarke–Cormack–Burkowski (CCB) lattice. A poset is a *lattice* if every pair of elements has a unique least upper bound (join) and a unique greatest lower bound (meet). For a locally finite, totally ordered set O, Boldi and Vigna [8] show that the antichain completion $\mathcal{E}_O$ forms a complete lattice. This structure, known as the *Clarke–Cormack–Burkowski lattice*, is named after the work of Clarke, Cormack, and Burkowski on minimal-interval semantics in information retrieval [12].

Let O be a locally finite, totally ordered set, so $\mathcal{E}_O$ is a CCB lattice. Each $\mathcal{A} \in \mathcal{E}_O$ is a family of intervals of O in which no interval properly contains another. For a family $\mathcal{F}$ of intervals, let $\min_{\subseteq}(\mathcal{F}) := \{I \in \mathcal{F} \mid \nexists J \in \mathcal{F} \text{ with } J \subsetneq I\}$ be the set of inclusion-minimal intervals in $\mathcal{F}$. The join and meet of $\mathcal{A}, \mathcal{B} \in \mathcal{E}_O$ are defined, respectively, as

$$\mathcal{A} \vee \mathcal{B} := \min_{\subseteq}(\mathcal{A} \cup \mathcal{B}) \quad \text{and} \quad \mathcal{A} \wedge \mathcal{B} := \min_{\subseteq}\big\{[\min\{\ell, \ell'\}, \max\{r, r'\}] \mid [\ell, r] \in \mathcal{A},\ [\ell', r'] \in \mathcal{B}\big\}.$$

See Figure 3 for an illustration of the partial order, meet, and join of two antichains.

Boldi and Vigna also show that $\mathcal{E}_O$ is a *completely distributive* lattice: arbitrary meets distribute over arbitrary joins in this lattice. In particular, for any family $(\mathcal{A}_i)_{i \in I} \subseteq \mathcal{E}_O$ and any $\mathcal{B} \in \mathcal{E}_O$, we have $\left(\bigvee_{i \in I} \mathcal{A}_i\right) \wedge \mathcal{B} = \bigvee_{i \in I} (\mathcal{A}_i \wedge \mathcal{B})$.

Lattice-based representation of strip partitions. For a vertical cut c, the sets $\min_{\subseteq}(\mathcal{I}_c(P_L))$ and $\min_{\subseteq}(\mathcal{I}_c(P_R))$ are antichains of intervals, since no interval in each set properly contains another. By Observation 3, these antichains suffice to decide whether the strips incident to c can be merged. In fact, the equality $\mathsf{opt}(P) = \mathsf{opt}(P_L) + \mathsf{opt}(P_R) - 1$ holds if and only if the meet $\min_{\subseteq}(\mathcal{I}_c(P_L)) \wedge \min_{\subseteq}(\mathcal{I}_c(P_R))$ contains an interval of length at most 1. Thus, in our dynamic program, we only need to maintain, for each subpolygon Q with vertical edge c, the antichain $\mathsf{Ant}(Q, c) := \min_{\subseteq}\big(\mathcal{I}_c(Q)\big)$. In other words, $\mathsf{Ant}(Q, c)$ is the set of inclusion-minimal x-intervals of strips incident to c across all optimal strip partitions of Q.

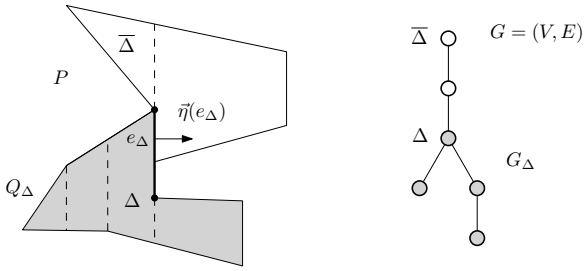


Figure 4 Subproblem structure $(Q_\Delta, e_\Delta, \vec{\eta}(e_\Delta))$ is defined for every non-root node Δ of $G = (V, E)$.

At first glance, these antichains appear to be families of intervals of $\mathbb{R}$, which is not locally finite. However, as we will show, once we fix some $\varepsilon \geq 0$, every interval that arises in our algorithm has endpoints of the form $x(p) + m$ or $x(p) + m \pm \varepsilon$, where p is a vertex of P and $m \in \mathbb{Z}$. Thus all endpoints lie in

$$\mathbb{Z}(P, \varepsilon) := \{\, x(p) + m - \varepsilon,\ x(p) + m,\ x(p) + m + \varepsilon \mid p \text{ is a vertex of } P,\ m \in \mathbb{Z} \,\},$$

a locally finite, totally ordered set. Consequently, every antichain maintained by our dynamic program lies in the CCB lattice $\mathcal{E}_{\mathbb{Z}(P,\varepsilon)}$. This lattice is completely distributive.

3.2 Subproblem structures in the refined dual tree

We now describe the subproblem structure underlying our dynamic program for simple polygons, formulated in terms of antichains of intervals.

We associate each subproblem with a pair (Q, e), where Q is a union of trapezoids from the vertical trapezoidal decomposition V and e is a vertical boundary edge of Q. (Recall that $\mathcal{I}_e(Q)$ denotes the family of x-intervals of all strips incident to edge e across all optimal strip partitions of Q.) We compress this family into an antichain by keeping only inclusion-minimal intervals: $\mathsf{Ant}(Q, e) := \min_{\subseteq}\big(\mathcal{I}_e(Q)\big)$. Every interval in $\mathsf{Ant}(Q, e)$ has length at most 1, and each such interval includes the common x-coordinate $x(e)$.

Let $G = (V, E)$ be the dual graph of the trapezoidal decomposition V, and fix an arbitrary leaf $\overline{\Delta} \in V$ as the root. For each node $\Delta \in V$, let G_Δ be the subtree of G rooted at Δ and let $Q_\Delta \subseteq P$ be the subpolygon obtained as the union of the trapezoids in G_Δ. A vertical cut of the trapezoidal decomposition that lies on ∂Q_Δ is called a *vertical interface* of Q_Δ.

Because G is a tree, all the vertical interfaces of Q_Δ lie on a single vertical edge of Q_Δ for $\Delta \neq \overline{\Delta}$. Let e_Δ denote this vertical edge. Note that e_Δ is contained in $\partial\Delta$. We define the *normal direction* of e_Δ, denoted $\vec{\eta}(e_\Delta)$, to be right if e_Δ lies on the right side of Δ, and left if it lies on the left side. We refer to the triple $(Q_\Delta, e_\Delta, \vec{\eta}(e_\Delta))$ as the *subproblem structure* associated with Δ. See Figure 4.

Refining G into a binary tree $\widetilde{G}$. We transform the dual graph G of V into a rooted binary tree $\widetilde{G}$ by locally refining each star centered at a trapezoid. Let $\Delta \in V$ be a trapezoid in V with subproblem structure $(Q_\Delta, e_\Delta, \vec{\eta}(e_\Delta))$, and assume without loss of generality that $\vec{\eta}(e_\Delta) = \mathsf{right}$. Suppose Δ has p children incident to its left side and q children on its right. Let $(\Delta_i^L, e_i^L, \vec{\eta}(e_i^L))_{i=1}^p$ (resp. $(\Delta_j^R, e_j^R, \vec{\eta}(e_j^R))_{j=1}^q$) be the subproblem structures associated with its left (resp. right) children. By construction, each $\vec{\eta}(e_i^L) = \mathsf{right}$ and each $\vec{\eta}(e_i^R) = \mathsf{left}$.

First, we construct a binary tree T_L whose leaves are the left children of Δ. If $p = 0$, we do not create T_L; if $p = 1$, then T_L is a single node Δ_1^L. For $p \geq 2$, we create $(p - 1)$ new internal nodes $u_2, \ldots, u_p$ and set each u_i as the parent of Δ_i^L and u_{i-1} for each $i = 2, \ldots, p$, with $u_1 := \Delta_1^L$. Symmetrically, we construct a binary tree T_R for the right children of Δ.

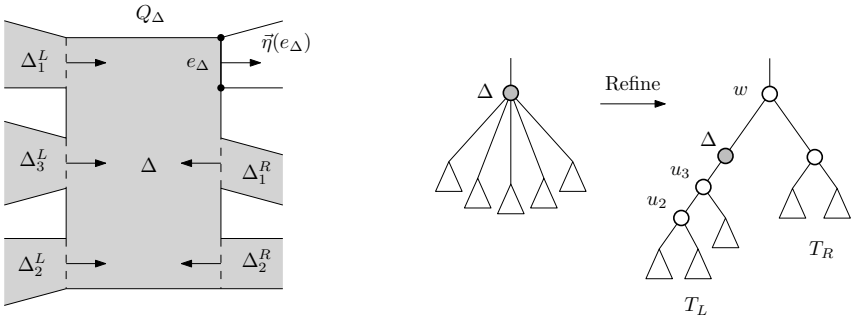


Figure 5 (a) The trapezoid Δ has three left children and two right children in the subtree G_Δ. (b) Refining the local star centered at Δ into a binary tree.

Next, we remove all edges from Δ to its children in G. If T_L exists (i.e., $p \geq 1$), we attach the root of T_L as the sole child of Δ. If T_R exists, let w be a new internal node whose children are the root of T_R and Δ; otherwise, let $w := \Delta$. If Δ has a parent z in G, we replace the edge between z and Δ by an edge between z and w. The case $\vec{\eta}(e_\Delta) = \mathsf{left}$ is handled symmetrically. See Figure 5 for an illustration.

Processing all trapezoids in postorder replaces every star subgraph in G with a binary tree. At each step, only the adjacency between Δ and its children changes; the subtrees rooted at those children remain intact. This refinement adds only $O(n)$ new internal nodes in total, and $\widetilde{G}$ can be constructed in $O(n)$ time. We refer to the original nodes of G as *trapezoid nodes* and the new internal nodes as *bridge nodes*. By construction, every trapezoid node in $\widetilde{G}$ has at most one child, and every bridge node has exactly two children.

Subproblem structures in $\widetilde{G}$. We lift the subproblem structures defined for nodes of G to the nodes of the refined tree $\widetilde{G}$. For each non-root node u, we associate a triple $(Q_u, e_u, \vec{\eta}(e_u))$, where Q_u is a weakly simple subpolygon of P, e_u is a vertical edge on ∂Q_u, and $\vec{\eta}(e_u) \in \{\mathsf{left}, \mathsf{right}\}$ is the outward normal direction of e_u with respect to Q_u. If u is a trapezoid node corresponding to a trapezoid $\Delta \in V$, we inherit its subproblem structure by setting $(Q_u, e_u, \vec{\eta}(e_u)) := (Q_\Delta, e_\Delta, \vec{\eta}(e_\Delta))$.

Suppose that u is a bridge node with children v and w. Assume that $(Q_v, e_v, \vec{\eta}(e_v))$ and $(Q_w, e_w, \vec{\eta}(e_w))$ are already defined. Let s be the minimal vertical segment spanning both e_v and e_w, and define $Q_u := Q_v \cup s \cup Q_w$. If $\vec{\eta}(e_v) = \vec{\eta}(e_w)$, then u is a *same-side bridge*. We set $e_u := s$ and $\vec{\eta}(e_u) := \vec{\eta}(e_v)$. In this case, Q_u has boundary edges that touch each other without crossing, so Q_u is only weakly simple.

If $\vec{\eta}(e_v) \neq \vec{\eta}(e_w)$, then u is a *cross-side bridge*. In this case, Q_u is always simple and one of e_v, e_w properly contains the other. If $e_v \subsetneq e_w$, we set $e_u := e_w \setminus e_v$ and $\vec{\eta}(e_u) := \vec{\eta}(e_w)$; otherwise, $e_u := e_v \setminus e_w$ and $\vec{\eta}(e_u) := \vec{\eta}(e_v)$. Figure 6(a-b) illustrates the subproblem structures at bridge nodes.

For the root node ρ of $\widetilde{G}$, which corresponds to the leaf $\overline{\Delta}$ of G, the region $Q_\rho = P$ has no vertical interface on its boundary. Then, we cannot directly define a subproblem structure for ρ. Let e be the vertical side of $\overline{\Delta}$ that is not incident to any other trapezoid of V. Note that e may be a single vertex if $\overline{\Delta}$ is a triangle. In this degenerate case, we conceptually replace e with a very short vertical segment, obtained by extending it slightly in both vertical directions. This extension preserves the x-interval of $\overline{\Delta}$, keeps P simple, and does not affect

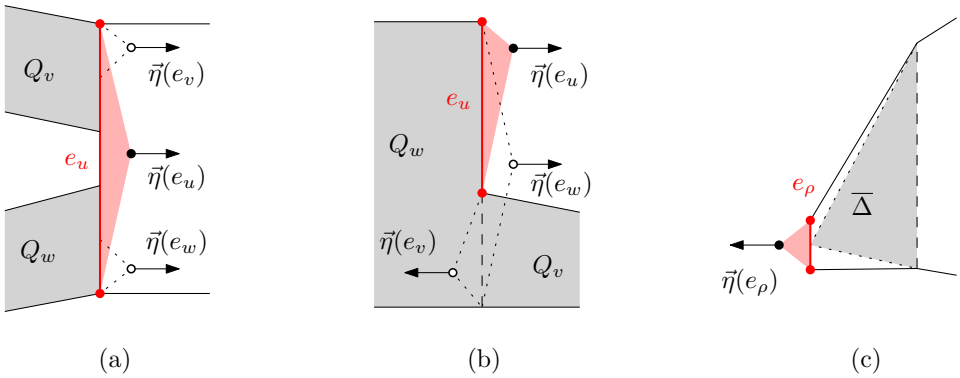


Figure 6 The subproblem structures $(Q_u, e_u, \vec{\eta}(e_u))$ in $\widetilde{G}$ when u is (a) a same-side bridge, (b) a cross-side bridge, and (c) the root; attaching the cap $T_\rho(\varepsilon)$ along e_ρ yields the simple polygon Q_ρ^ε.

any strip partition of P. We set $e_\rho := e$ and define $\vec{\eta}(e_\rho)$ as left if e lies on the left side of $\overline{\Delta}$, or right if e lies on the right side. This yields the subproblem structure $(Q_\rho, e_\rho, \vec{\eta}(e_\rho))$ for the root node ρ. See Figure 6(c).

▶ **Lemma 4.** *For every node u of $\widetilde{G}$, the subproblem structure $(Q_u, e_u, \vec{\eta}(e_u))$ is well-defined: Q_u is a weakly simple subpolygon of P, e_u is a vertical edge of Q_u with positive length, and $\vec{\eta}(e_u)$ is the outward normal direction of e_u with respect to Q_u.*

Reducing weakly simple subpolygons to simple ones. By Lemma 4, every node u of $\widetilde{G}$ has an associated subproblem structure $(Q_u, e_u, \vec{\eta}(e_u))$. If u is a trapezoid node or a cross-side bridge node, then Q_u is simple. If u is a same-side bridge node, however, Q_u is only weakly simple. To handle all subproblems in the simple polygon setting, we convert Q_u into a simple polygon by attaching a thin triangular cap along e_u on the outside of Q_u.

Let q_u be any point on the edge e_u, excluding its endpoints. For sufficiently small $\varepsilon > 0$, let $q_u(\varepsilon)$ be the point obtained by shifting q_u by ε units in the direction of $\vec{\eta}(e_u)$. Let $T_u(\varepsilon)$ be the triangle with base e_u and apex $q_u(\varepsilon)$.

▶ **Lemma 5.** *For each node u of $\widetilde{G}$, there exists $\varepsilon_0 > 0$ such that $Q_u \cup T_u(\varepsilon)$ is a simple polygon for all $0 < \varepsilon \le \varepsilon_0$.*

Let $(x_i)_{i=1}^n$ be the x-coordinates of the vertices of P, and let $\mathbb{Z}(P, 0) := \{\, x_i + m \mid i \in [n],\, m \in \mathbb{Z} \,\}$. Let $\delta > 0$ be the minimum positive distance between two distinct points of $\mathbb{Z}(P, 0)$. From Lemma 5, for each node u, there exists $\varepsilon_0(u) > 0$ such that $Q_u \cup T_u(\varepsilon)$ is simple for all $0 < \varepsilon \le \varepsilon_0(u)$. Choose a particular $\varepsilon > 0$ satisfying $\varepsilon \le \min_u \varepsilon_0(u)$ and $\varepsilon < \frac{\delta}{2}$. Note that δ and ε are not computed explicitly; we only fix such values conceptually.

Now we define the capped subpolygon $Q_u^\varepsilon := Q_u \cup T_u(\varepsilon)$, which is a simple polygon. Figure 6 illustrates how the cap $T_u(\varepsilon)$ attaches to e_u for each node type. Let $\mathsf{opt}^\varepsilon(Q_u)$ denote the minimum number of strips in an optimal strip partition of Q_u^ε. Among all optimal strip partitions of Q_u^ε, let $\mathcal{S}^\varepsilon(Q_u, e_u)$ be the set of strips incident to the apex $q_u(\varepsilon)$, and define $\mathsf{Ant}^\varepsilon(Q_u, e_u) := \min_\subseteq \{ I_x(S) \mid S \in \mathcal{S}^\varepsilon(Q_u, e_u) \}$, where $I_x(S)$ is the interval obtained by projecting S onto the x-axis. Then $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ is an antichain of intervals. By definition, every interval in $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ is realized as the x-projection of a strip incident to $q_u(\varepsilon)$ in some optimal partition of Q_u^ε. Although the family $\mathcal{S}^\varepsilon(Q_u, e_u)$ may be infinite, Algorithm 1 and Lemma 9 together imply that $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ is finite. We define the dynamic-programming

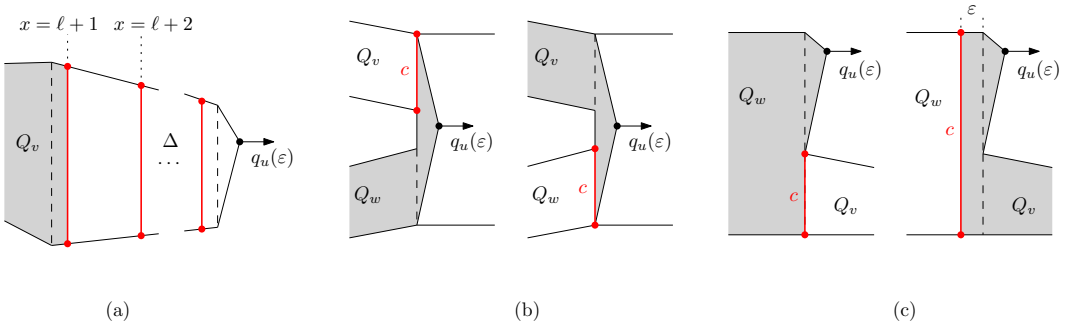


Figure 7 Additional vertical cuts are inserted when no strip can be merged or extended across the node: (a) trapezoid node, (b) same-side bridge node, and (c) cross-side bridge node.

subproblem $\mathrm{OSP}^{\varepsilon}(Q_u, e_u)$ to be the task of computing both $\mathsf{opt}^{\varepsilon}(Q_u)$ and $\mathsf{Ant}^{\varepsilon}(Q_u, e_u)$. The pair $\big(\mathsf{opt}^{\varepsilon}(Q_u), \mathsf{Ant}^{\varepsilon}(Q_u, e_u)\big)$ is called the *DP state* at node u; its first entry is the *value component*, and its second entry is the *antichain component*.

4 Dynamic programming algorithm and complexity

We process the nodes of the rooted binary tree $\widetilde{G}$ in bottom-up order. For each node u, after the subproblems for all of its descendants have been solved, we compute the DP state of $\mathrm{OSP}^{\varepsilon}(Q_u, e_u)$ using the update rule for either a trapezoid node or a bridge node.

During this process, we maintain the following invariant for each node u of $\widetilde{G}$. The value $\mathsf{opt}^{\varepsilon}(Q_u)$ is the minimum number of strips in a strip partition of Q_u^{ε}. Let $\mathcal{S}^{\varepsilon}(Q_u, e_u)$ be the set of strips incident to the apex $q_u(\varepsilon)$ among all optimal partitions of Q_u^{ε}. Then $\mathsf{Ant}^{\varepsilon}(Q_u, e_u)$ consists of the inclusion-minimal x-intervals of the strips in $\mathcal{S}^{\varepsilon}(Q_u, e_u)$.

4.1 Bottom-up evaluation on the refined tree

The update procedure is described in detail in Algorithm 1. Throughout the algorithm, we assume that $\vec{\eta}(e_u) = \mathsf{right}$; the case $\vec{\eta}(e_u) = \mathsf{left}$ is symmetric. We use the low-pass operator Filter_1 on the family of intervals. For $\mathcal{F} \subseteq \mathbf{I}_{\mathbb{R}}$, let $\mathrm{Filter}_1(\mathcal{F}) := \{\, I \in \mathcal{F} \mid |I| \le 1 \,\}$, which discards intervals of length greater than 1.

Trapezoid nodes. Let u be a trapezoid node corresponding to a trapezoid Δ. If u has a child v in $\widetilde{G}$, the DP state $(\mathsf{opt}^{\varepsilon}(Q_v), \mathsf{Ant}^{\varepsilon}(Q_v, e_v))$ has already been computed. If u is a leaf, we introduce a virtual child v by taking e_v to be the vertical side of Δ opposite to e_u, letting Q_v be the degenerate trapezoid consisting of e_v, and setting $\vec{\eta}(e_v) = \vec{\eta}(e_u)$; then $\mathsf{opt}^{\varepsilon}(Q_v) = 1$ and $\mathsf{Ant}^{\varepsilon}(Q_v, e_v)$ consists of a single interval of length ε adjacent to $x(e_v)$.

The update at a trapezoid node proceeds as follows. Let $[a, b]$ be the x-interval of Δ, where $a = x(e_v)$ and $b = x(e_u)$. For each strip $S \in \mathcal{S}^{\varepsilon}(Q_v, e_v)$, we remove the old cap $T_v(\varepsilon)$, insert Δ into the polygon, and then attach the new cap $T_u(\varepsilon)$ along e_u. If $[\ell, r]$ is the x-interval of S, then the extended strip has x-interval $[\min\{\ell, a\}, \max\{r, b + \varepsilon\}]$, which is exactly the interval obtained by taking the meet of $\{[\ell, r]\}$ with $\{[a, b + \varepsilon]\}$. Thus, the inclusion-minimal x-intervals of all feasible extensions are exactly $\mathrm{Filter}_1\big(\mathsf{Ant}^{\varepsilon}(Q_v, e_v) \wedge \{[a, b + \varepsilon]\}\big)$.

If at least one extended strip survives, then we do not introduce any new cut at u. If none survive, then no strip can be extended through Δ without violating the unit-width constraint. Let $[\ell, r]$ be the x-interval of the rightmost strip in $\mathcal{S}^{\varepsilon}(Q_v, e_v)$. If no extended

Algorithm 1 DP update at a node u (case $\vec{\eta}(e_u) = \mathsf{right}$).

Input: node u of $\widetilde{G}$ and the DP states at its children
Output: DP state $(\mathsf{opt}^\varepsilon(Q_u), \mathsf{Ant}^\varepsilon(Q_u, e_u))$ at u

if *u is a trapezoid node with child v* **then**
  TrapezoidUpdate(u, v)
else if *u is a bridge node with children v, w* **then**
  BridgeUpdate(u, v, w)

procedure *TrapezoidUpdate*(u, v)
  Let Δ be the trapezoid of u with x-interval $[a, b]$
  $\mathsf{Ant}^\varepsilon(Q_u, e_u) \leftarrow \mathrm{Filter}_1\big(\mathsf{Ant}^\varepsilon(Q_v, e_v) \wedge \{[a, b + \varepsilon]\}\big), \quad \mathsf{opt}^\varepsilon(Q_u) \leftarrow \mathsf{opt}^\varepsilon(Q_v)$
  if $\mathsf{Ant}^\varepsilon(Q_u, e_u) = \varnothing$ **then**
    let $[\ell, r]$ be the interval in $\mathsf{Ant}^\varepsilon(Q_v, e_v)$ with maximum left endpoint ℓ
    $k \leftarrow \lfloor b + \varepsilon - \ell \rfloor$
    $\mathsf{Ant}^\varepsilon(Q_u, e_u) \leftarrow \{[\ell + k,\ b + \varepsilon]\}, \quad \mathsf{opt}^\varepsilon(Q_u) \leftarrow \mathsf{opt}^\varepsilon(Q_v) + k$

procedure *BridgeUpdate*(u, v, w)
  $\mathsf{Ant}^\varepsilon(Q_u, e_u) \leftarrow \mathrm{Filter}_1\big(\mathsf{Ant}^\varepsilon(Q_v, e_v) \wedge \mathsf{Ant}^\varepsilon(Q_w, e_w)\big)$
  $\mathsf{opt}^\varepsilon(Q_u) \leftarrow \mathsf{opt}^\varepsilon(Q_v) + \mathsf{opt}^\varepsilon(Q_w) - 1$
  if $\mathsf{Ant}^\varepsilon(Q_u, e_u) = \varnothing$ **then**
    $\mathsf{Ant}^\varepsilon(Q_u, e_u) \leftarrow \mathsf{Ant}^\varepsilon(Q_v, e_v) \vee \mathsf{Ant}^\varepsilon(Q_w, e_w)$
    $\mathsf{opt}^\varepsilon(Q_u) \leftarrow \mathsf{opt}^\varepsilon(Q_v) + \mathsf{opt}^\varepsilon(Q_w)$

strip survives, we insert a series of vertical cuts in $\Delta \cup T_u(\varepsilon)$ at x-coordinates $\ell + 1, \ell + 2, \ldots$, decomposing the extended region into unit-width strips (with the last piece possibly shorter than 1). We then increase $\mathsf{opt}^\varepsilon(Q_u)$ by the number of new strips, and define $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ to consist of the single x-interval of the rightmost suffix strip incident to $q_u(\varepsilon)$.

Bridge nodes. Let u be a bridge node of $\widetilde{G}$ with children v and w. At a bridge node, we consider pairs of strips $S_v \in \mathcal{S}^\varepsilon(Q_v, e_v)$ and $S_w \in \mathcal{S}^\varepsilon(Q_w, e_w)$, and ask whether they can be combined into a single strip incident to $q_u(\varepsilon)$. By the choice of ε, the x-interval of each of S_v and S_w already contains $[x(e_u) - \varepsilon,\ x(e_u) + \varepsilon]$. Hence, replacing the child caps $T_v(\varepsilon), T_w(\varepsilon)$ with the cap $T_u(\varepsilon)$ does not change either x-interval.

For a same-side bridge, this combination is obtained by removing the caps $T_v(\varepsilon)$ and $T_w(\varepsilon)$, gluing the two strips along e_u, and attaching the cap $T_u(\varepsilon)$. For a cross-side bridge, we remove the two caps, take the union of the resulting strips, and attach $T_u(\varepsilon)$ along e_u. In either case, if $[\ell, r]$ and $[\ell', r']$ are the x-intervals of S_v and S_w, respectively, then the combined region has x-interval $[\min\{\ell, \ell'\}, \max\{r, r'\}]$. Thus, the inclusion-minimal x-intervals of all feasible combinations are exactly $\mathrm{Filter}_1\big(\mathsf{Ant}^\varepsilon(Q_v, e_v) \wedge \mathsf{Ant}^\varepsilon(Q_w, e_w)\big)$.

If at least one pair of strips yields a feasible strip, then no additional cut is introduced at u, and the feasible combined strips form $\mathcal{S}^\varepsilon(Q_u, e_u)$. Otherwise, we insert a vertical cut near e_u so that strips from the two child subproblems cannot be combined across u. The strips from Q_v and Q_w are then extended separately to $q_u(\varepsilon)$. Hence, $\mathsf{Ant}^\varepsilon(Q_u, e_u) = \mathsf{Ant}^\varepsilon(Q_v, e_v) \vee \mathsf{Ant}^\varepsilon(Q_w, e_w)$.

Cut placement at nodes. We now describe how new cuts are placed when strips cannot be merged or extended at a node u without violating the unit-width constraint.

If u is a trapezoid node, no strip extends through its trapezoid Δ, so we place vertical cuts in $\Delta \cup T_u(\varepsilon)$. They start at $x = \ell + 1$, where $[\ell, r]$ is the interval with the maximum left endpoint in $\mathsf{Ant}^\varepsilon(Q_v, e_v)$, and continue at unit distance. See Figure 7(a).

Now suppose that u is a bridge node with children v and w. If some strip in $\mathcal{S}^\varepsilon(Q_v, e_v)$ can merge with one in $\mathcal{S}^\varepsilon(Q_w, e_w)$ into a unit-width strip, no new cut is added at u. Otherwise, we place a single vertical cut near e_u to prevent any strip from Q_v from merging with a strip from Q_w, so that all strips are extended to $q_u(\varepsilon)$ separately. See Figure 7(b–c).

The position of this cut depends on the bridge type and on the child from which each x-interval in $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ originates. For a same-side bridge, the cut is placed along e_w if the interval comes from $\mathsf{Ant}^\varepsilon(Q_v, e_v)$ and along e_v if it comes from $\mathsf{Ant}^\varepsilon(Q_w, e_w)$. For a cross-side bridge, either $e_v \subsetneq e_w$ or $e_w \subsetneq e_v$. Assume $e_v \subsetneq e_w$. If the interval comes from $\mathsf{Ant}^\varepsilon(Q_w, e_w)$, the cut is placed along e_v. Otherwise it is placed inside Q_w at x-distance ε from e_u, blocking any strip from Q_w^ε from reaching $q_u(\varepsilon)$; see Figure 7(c). The case $e_w \subsetneq e_v$ is symmetric. Note that inserting these cuts at bridge nodes does not increase the x-interval of any strip from Q_v^ε or Q_w^ε; we prove this in the proof of Lemma 6.

Every interval appearing in an antichain $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ has endpoints of the form $x(p) + m$ or $x(p) + m \pm \varepsilon$, where p is a vertex of P and $m \in \mathbb{Z}$. Thus all endpoints lie in the locally finite, totally ordered set $\mathbb{Z}(P, \varepsilon)$, and each antichain maintained by our DP is an element of the Clarke–Cormack–Burkowski lattice $\mathcal{E}_{\mathbb{Z}(P,\varepsilon)}$. The update rules are written using meet ($\wedge$) and join ($\vee$) in this lattice, and the outputs of these operations remain in $\mathcal{E}_{\mathbb{Z}(P,\varepsilon)}$.

Since $\varepsilon > 0$ is chosen sufficiently small, the values $\mathsf{opt}(Q_u)$ and $\mathsf{opt}^\varepsilon(Q_u)$ differ by at most one. This difference occurs only when, in every optimal partition of Q_u^ε, the only strip incident to $q_u(\varepsilon)$ is the cap $T_u(\varepsilon)$; in this case we have $\mathsf{Ant}^\varepsilon(Q_u, e_u) = \{[x(e_u),\ x(e_u) + \varepsilon]\}$.

In the DP state, we store an endpoint of the form $x(p) + m$ or $x(p) + m \pm \varepsilon$ as the value $x(p) + m$ together with a single bit indicating whether the ε–offset is present. Since $\varepsilon > 0$ is chosen sufficiently small, all order and width tests used by the algorithm can be decided from the base values and the flags. For example, if $a, b \in \mathbb{Z}(P, 0)$ and $m \in \mathbb{Z}$, then

$$\begin{array}{ccccccc} b - a - 2\varepsilon > m & \iff & b - a > m, & \qquad & b - a + 2\varepsilon > m & \iff & b - a \geq m, \\ \lfloor (b - a) + 2\varepsilon \rfloor & = & \lfloor b - a \rfloor, & & \lfloor (b - a) - 2\varepsilon \rfloor & = & \lceil b - a \rceil - 1. \end{array}$$

▶ **Lemma 6.** *When Algorithm 1 is executed on the tree $\widetilde{G}$ in bottom-up order, it correctly computes, for every node u of $\widetilde{G}$, the pair $(\mathsf{opt}^\varepsilon(Q_u), \mathsf{Ant}^\varepsilon(Q_u, e_u))$, where $\mathsf{opt}^\varepsilon(Q_u)$ is the number of strips in an optimal partition of Q_u^ε, and $\mathsf{Ant}^\varepsilon(Q_u, e_u)$ is the antichain of x-intervals of the strips incident to $q_u(\varepsilon)$ across all optimal partitions of Q_u^ε.*

By Lemma 6, Algorithm 1 correctly computes $(\mathsf{opt}^\varepsilon(Q_\rho), \mathsf{Ant}^\varepsilon(Q_\rho, e_\rho))$ at the root ρ of $\widetilde{G}$, where $Q_\rho = P$. By the discussion above on the relation between $\mathsf{opt}^\varepsilon(Q_u)$ and $\mathsf{opt}(Q_u)$, we obtain $\mathsf{opt}(P)$ from this pair in constant time.

Encoding cut descriptors. We show that the additional cuts placed at trapezoid and bridge nodes admit a lossless encoding from which all cut descriptors in the optimal partition can be reconstructed in $O(n)$ time using $O(n)$ extra space. Importantly, we can generate this encoding during the DP without increasing the asymptotic running time.

▶ **Lemma 7.** *The cut descriptors produced by the dynamic program admit an $O(n)$-space encoding from which all cuts in an optimal partition can be reconstructed in $O(n)$ time.*

Failure of greedy approach. One might suspect that we can discard some intervals from an antichain component before passing information to its parent. This is impossible in general: if we discard even one interval before passing the DP state upward, the reduced state no longer determines the optimum.

Let Q be a simple polygon and let e be a vertical edge of Q. Let $\vec{\eta}(e) \in \{\mathsf{left}, \mathsf{right}\}$ denote the outward normal direction of e. Let $\mathcal{P}(Q, e)$ be the set of all simple polygons P for which there exists a simple polygon R such that $P = Q \cup R$ and $Q \cap R \subseteq e$. In other words, $\mathcal{P}(Q, e)$ consists of all simple polygons that contain $(Q, e, \vec{\eta}(e))$ as a subproblem structure in the orthogonal strip partition problem.

Let $\mathsf{Ant}(Q, e)$ be the antichain component computed by Algorithm 1 for the subproblem structure $(Q, e, \vec{\eta}(e))$. A *certificate* for $(Q, e, \vec{\eta}(e))$ is a subset $\mathcal{W} \subseteq \mathsf{Ant}(Q, e)$ with the following property: there exists an algorithm that, for every $P \in \mathcal{P}(Q, e)$, computes $\mathsf{opt}(P)$ from given $\mathcal{W}$, $\mathsf{opt}(Q)$, and the remaining region $P \setminus Q$. The *certificate complexity* of $(Q, e, \vec{\eta}(e))$ is $\min\{|\mathcal{W}| \mid \mathcal{W}$ is a certificate for $(Q, e, \vec{\eta}(e))\}$. This viewpoint is standard in lower bound arguments [4], and shows whether pruning $\mathsf{Ant}(Q, e)$ loses information necessary to determine $\mathsf{opt}(P)$.

▶ **Lemma 8.** *For every integer $n \geq 1$, there exists a subproblem instance $(Q, e, \vec{\eta}(e))$ with $O(n)$ vertices whose certificate complexity is at least n.*

Although keeping all intervals in the antichain component often appears redundant, Lemma 8 shows that one cannot safely prune intervals in general. There exist instances in which every interval is necessary; hence any greedy approach that discards intervals using only information available in the subproblem fails on some instance.

4.2 Algorithmic complexity and lower bounds

We analyze the running time and space of our dynamic program. The value component is updated only by testing whether the resulting antichain is empty, and the cut descriptor is computed during the meet or join operations without extra asymptotic cost. Thus, the overall running time is dominated by the cost of the meet and join operations on antichains.

In an antichain, intervals are strictly ordered: for two distinct intervals $[a, b]$ and $[a', b']$, we have $a < a'$ if and only if $b < b'$. Thus, we store each antichain as a list of intervals sorted by their left (or right) endpoints. Every antichain produced by our lattice operations is maintained in this sorted order.

▶ **Lemma 9.** *Let $\mathcal{A}$ and $\mathcal{B}$ be finite antichains of intervals in $\mathcal{E}_{\mathbb{Z}(P,\varepsilon)}$. Then, $|\mathcal{A} \wedge \mathcal{B}| \leq |\mathcal{A}| + |\mathcal{B}| - 1$ and $|\mathcal{A} \vee \mathcal{B}| \leq |\mathcal{A}| + |\mathcal{B}|$.*

If u is a leaf node of $\widetilde{G}$, $\mathsf{Ant}^{\varepsilon}(Q_u, e_u)$ consists of a single interval. At every trapezoid node or bridge node, Algorithm 1 performs exactly one meet or one join of two antichains. At every node u of $\widetilde{G}$, Lemma 9 implies that $|\mathsf{Ant}^{\varepsilon}(Q_u, e_u)|$ is bounded by the size of the subtree rooted at u. Boldi and Vigna [7] give one-pass, optimally lazy algorithms that compute $\mathcal{A} \wedge \mathcal{B}$ and $\mathcal{A} \vee \mathcal{B}$ in time linear in $|\mathcal{A}| + |\mathcal{B}|$. Moreover, the operation Filter_1 can clearly be implemented in time linear in the number of intervals in the antichain.

Let $T(t)$ be the total running time of Algorithm 1 on a subtree with t nodes, where $T(1) = O(1)$. If u is a trapezoid node, its unique child has a subtree of size $t-1$, the antichain at u has size $O(t)$, and the update takes $O(t)$ time; thus $T(t) = T(t-1)+O(t)$. If u is a bridge node with children whose subtrees have sizes a and $t-a-1$, then the two antichains together have size $O(t)$, so the update again takes $O(t)$ time and $T(t) = T(a) + T(t-a-1) + O(t)$. In the worst case, this recurrence yields $T(n) = O(n^2)$.

To improve this bound, we work in the real-RAM model, where random access and binary search allow these operations to be implemented more efficiently than the one-pass algorithms. Let $L(\mathcal{B})$ and $R(\mathcal{B})$ denote the sorted list of left and right endpoints of intervals in $\mathcal{B}$, respectively. For each interval $I = [\ell, r] \in \mathcal{A}$, we compute the insertion ranks of ℓ in $L(\mathcal{B})$ and of r in $R(\mathcal{B})$, that is, their positions in these sorted lists.

Given two sorted lists X and Y with $|X| = k$ and $|Y| = m$, we use a selection technique of Kaplan et al. [14] to compute all insertion ranks of X in Y in $O\left(k \log \frac{m}{k}\right)$ time for $m \geq 4k$. We partition Y into blocks of size $B := \lfloor m/(4k) \rfloor$; for each $x \in X$, we first identify the block containing x and then perform a binary search within that block. The block size B itself can be computed in $O\left(\log \frac{m}{k}\right)$ comparisons by testing powers of two, so this does not change the overall running time.

The following lemma shows that the join and filtered meet of two antichains can be computed efficiently, assuming precomputed insertion ranks.

▶ **Lemma 10.** *Let $\mathcal{A}, \mathcal{B}$ be two finite antichains of intervals in $\mathcal{E}_{\mathbb{Z}(P,\varepsilon)}$ such that every interval in $\mathcal{A} \cup \mathcal{B}$ has length at most 1. Assume that, for every interval $I \in \mathcal{A}$, the insertion ranks of its endpoints among those of $\mathcal{B}$ are available in constant time. Then, $\mathcal{A} \vee \mathcal{B}$ and $\mathrm{Filter}_1(\mathcal{A} \wedge \mathcal{B})$ can be computed in $O(|\mathcal{A}|)$ time.*

Let $|\mathcal{A}| = k$ and $|\mathcal{B}| = m$ with $k \leq m$. We first compute the insertion ranks of all endpoints of intervals in $\mathcal{A}$ among $L(\mathcal{B})$ and $R(\mathcal{B})$ in $O\left(k \log \frac{m}{k}\right)$ time. If $4k \leq m$, we then compute $\mathrm{Filter}_1(\mathcal{A} \wedge \mathcal{B})$ and $\mathcal{A} \vee \mathcal{B}$ in $O(k)$ time. If $k \leq m \leq 4k$, we instead use the linear-time algorithm of Boldi and Vigna, which runs in $O(m + k) = O(k)$ time.

Let $T(t)$ be the running time of Algorithm 1 on a subtree of size t. For a trapezoid node with a unique child, one antichain in the meet is a singleton, so the update takes $O(\log t)$ time and $T(t) = T(t-1) + O(\log t)$. For a node whose children have subtree sizes s and $t-s-1$ with $s \leq t-s-1$, the cost at that node is $O\left(s \log \frac{t-s}{s}\right)$, so $T(t) = T(s) + T(t-s-1) + O\left(s \log \frac{t-s}{s}\right)$. These recurrences give $T(n) = O(n \log n)$ in the worst case.

▶ **Theorem 11.** *For a simple polygon P with n vertices, the problems $OSP_V^{\mathrm{simp}}(P)$ and $OSP_R^{\mathrm{simp}}(P)$ can be solved in $O(n \log n)$ time using $O(n)$ space.*

Lower bounds. We now show that no sublinear-time algorithm exists for the value or reporting version, so our algorithm is optimal up to a logarithmic factor. If the input polygon is given as an array of vertices in boundary order and each query reveals one input vertex at unit cost, then any algorithm requires $\Omega(n)$ time in the worst case.

▶ **Theorem 12.** *Let P be a simple polygon with n vertices stored in an array in boundary order, and suppose that each query reveals one vertex of the input polygon and has unit cost. Then, any algorithm for $OSP_V^{\mathrm{simp}}(P)$ or $OSP_R^{\mathrm{simp}}(P)$ requires $\Omega(n)$ time in the worst case.*

5 Orthogonal strip partitioning of convex and self-overlapping polygons

We also consider the orthogonal strip partition problem for convex and self-overlapping polygons.

- **Convex polygons.** For a convex polygon P with n vertices given in boundary order, $\mathrm{OSP}_V^{\mathrm{conv}}(P)$ and $\mathrm{OSP}_R^{\mathrm{conv}}(P)$ can be solved in $O(\log n)$ and $O\left(h \log\left(1 + \frac{n}{h}\right)\right)$ time, respectively, where h is the horizontal width of P. Both bounds are optimal in the decision-tree model.

- **Self-overlapping polygons.** For a self-overlapping polygon P with n vertices given together with a triangulation, $\mathrm{OSP}_V^{\mathrm{self}}(P)$ and $\mathrm{OSP}_R^{\mathrm{self}}(P)$ can be solved in $O(n \log n)$ time using $O(n)$ space. This bound is optimal in the algebraic computation-tree model.

6 Concluding remarks

We studied orthogonal strip partitioning for convex, simple and self-overlapping polygons. For convex polygons, we presented optimal algorithms for both value and reporting versions. In particular, for the reporting version, our reporting algorithm is input-sensitive optimal with respect to the horizontal width of P.

For simple and self-overlapping polygons, we presented a dynamic programming approach that runs in $O(n \log n)$ time and uses $O(n)$ space for both the value and reporting versions. The DP recurrences are naturally formulated on the Clarke–Cormack–Burkowski lattice from order theory. For self-overlapping inputs, the running time is optimal in the algebraic computation-tree model; for simple polygons, it is optimal up to a logarithmic factor in the decision-tree model.

The main open problem is to close the logarithmic gap between the upper and lower bounds for simple polygons. Lemma 8 rules out any greedy pruning rule, so in the worst case the algorithm must compute a meet or join operation between antichains of size $\Theta(n)$. However, it seems difficult to construct a simple polygon in which such expensive lattice operations occur repeatedly, whereas this is easy for self-overlapping inputs. This implies that simplicity may limit the number of costly lattice operations. Even if this intuition is correct, a binary-search based implementation still incurs $O(n \log n)$ time. To obtain an $O(n)$-algorithm, one likely needs a linear-scan procedure, similar to that of Boldi and Vigna [7], together with an amortized analysis of the total cost of all lattice operations.

References

1. Mikkel Abrahamsen, Joakim Blikstad, André Nusser, and Hanwen Zhang. Minimum star partitions of simple polygons in polynomial time. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 904–910. Association for Computing Machinery, 2024. doi:10.1145/3618260.3649756.
2. Mikkel Abrahamsen and Nichlas Langhoff Rasmussen. Partitioning a polygon into small pieces. In *Proc. 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3562–3589. Society for Industrial and Applied Mathematics, 2025. doi:10.1137/1.9781611978322.118.
3. Mikkel Abrahamsen and Jack Stade. Hardness of packing, covering and partitioning simple polygons with unit squares. In *Proc. 65th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1355–1371. IEEE, 2024. doi:10.1109/FOCS61266.2024.00087.
4. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
5. Thøger Bang. A solution of the "plank problem". *Proc. American Mathematical Society*, 2(6):990–993, 1951.
6. Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice-Hall, 1990.
7. Paolo Boldi and Sebastiano Vigna. Efficient optimally lazy algorithms for minimal-interval semantics. *Theoretical Computer Science*, 648:8–25, 2016. doi:10.1016/J.TCS.2016.07.036.
8. Paolo Boldi and Sebastiano Vigna. On the lattice of antichains of finite intervals. *Order*, 35(1):57–81, 2018. doi:10.1007/S11083-016-9418-8.
9. Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991. doi:10.1007/BF02574703.

10 Bernard Chazelle and David Dobkin. Decomposing a polygon into its convex parts. In *Proc. 11th Annual ACM Symposium on Theory of Computing (STOC)*, pages 38–48. Association for Computing Machinery, 1979. doi:10.1145/800135.804396.

11 Jaehoon Chung, Kazuo Iwama, Chung-Shou Liao, and Hee-Kap Ahn. Minimum Partition of Polygons Under Width and Cut Constraints. In *36th International Symposium on Algorithms and Computation (ISAAC 2025)*, volume 359 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ISAAC.2025.22.

12 Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, 1995. doi:10.1093/COMJNL/38.1.43.

13 J. W. S. Hearle. The structural mechanics of fibers. *Journal of Polymer Science Part C: Polymer Symposia*, 20(1):215–251, 1967.

14 Haim Kaplan, László Kozma, Or Zamir, and Uri Zwick. Selection from heaps, row-sorted matrices, and x+y using soft heaps. In *Proc. 2nd Symposium on Simplicity in Algorithms (SOSA)*, pages 5:1–5:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.5.

15 J. Mark Keil. Decomposing a polygon into simpler components. *SIAM Journal on Computing*, 14(4):799–817, 1985. doi:10.1137/0214056.

16 Tianhui Lan, Bingze Wang, Junchao Zhang, Hao Wei, and Xu Liu. Utilization of Waste Wind Turbine Blades in Performance Improvement of Asphalt Mixture. *Frontiers in Materials*, 10:1164693, 2023.

17 Jaegun Lee, Hyojeong An, Hwi Kim, and Hee-Kap Ahn. Monotone partitions of simple polygons. In *Proc. 36th International Workshop on Combinatorial Algorithms (IWOCA)*, pages 72–85, 2025. doi:10.1007/978-3-031-98740-3_6.

18 A. Lingas and V. Soltan. Minimum convex partition of a polygon with holes by cuts in given directions. *Theory of Computing Systems*, 31(5):507–538, 1998. doi:10.1007/S002240000101.

19 Robin Ru-Sheng Liu. *Partitioning Rectilinear Polygons into Simpler Components*. PhD thesis, The University of Texas at Dallas, 1985.

20 Arnold Schönhage. On the power of random access machines. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, pages 520–529, 1979. doi:10.1007/3-540-09510-1_42.

21 Peter W. Shor and Christopher J. Van Wyk. Detecting and decomposing self-overlapping curves. *Computational Geometry*, 2(1):31–50, 1992. doi:10.1016/0925-7721(92)90019-O.

One Color Makes All the Difference in the Tractability of Partial Coloring in Semi-Streaming

Avinandan Das
Aalto University, Finland

Abstract

This paper investigates the semi-streaming complexity of *k-partial coloring*, a generalization of proper graph coloring. For $k \geq 1$, a k-partial coloring requires that each vertex v in an n-node graph is assigned a color such that at least $\min\{k, \deg(v)\}$ of its neighbors are assigned colors different from its own. This framework naturally extends classical coloring problems: specifically, k-partial $(k+1)$-coloring and k-partial k-coloring generalize $(\Delta+1)$-proper coloring and Δ-proper coloring, respectively.

Prior works of Assadi, Chen, and Khanna [SODA 2019] and Assadi, Kumar, and Mittal [TheoretiCS 2023] show that both $(\Delta+1)$-proper coloring and Δ-proper coloring admit one-pass randomized semi-streaming algorithms. We explore whether these efficiency gains extend to their partial coloring generalizations and reveal a sharp computational threshold: while k-partial $(k+1)$-coloring admits a one-pass randomized semi-streaming algorithm, the k-partial k-coloring remains semi-streaming intractable, effectively demonstrating a "dichotomy of one color" in the streaming model.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases Graph Coloring, Semi-streaming algorithms, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.15

Related Version *Full Version*: https://arxiv.org/abs/2602.18987

Funding This work was supported in part by the Research Council of Finland, Grants 363558 and 359104.

Acknowledgements Avinandan would like to thank Pierre Fraigniaud and Adi Rosén for proposing the problem of k-partial $(k+1)$-coloring and many discussions.

1 Introduction

Partial coloring. A proper vertex coloring requires that every edge is bichromatic, i.e., every vertex v has *all* its neighbors colored differently from v. Partial coloring relaxes this requirement by only demanding *enough* differently-colored neighbors. Formally, for integers $k \geq 0$ and $c \geq 1$, a *k-partial c-coloring* of a graph $G = (V, E)$ is a map $\chi : V \to [c]$ such that for every vertex v,

$$\left|\{u \in N_G(v) : \chi(u) \neq \chi(v)\}\right| \;\geq\; \min\{k, \deg_G(v)\}.$$

Equivalently, if $\deg_G(v) \geq k$ then v must have at least k neighbors of a different color, whereas if $\deg_G(v) < k$ then all neighbors of v must have a different color. Specifically, Δ-partial coloring implies a proper coloring of G where Δ is the max. degree of G.

The most prominent parameter choices are k-partial $(k+1)$-coloring and k-partial k-coloring. These notions generalize the classical degree-based benchmarks: when $k = \Delta$, a Δ-partial $(\Delta+1)$-coloring is exactly a proper $(\Delta+1)$-coloring, and a Δ-partial Δ-coloring is exactly a proper Δ-coloring.

A simple greedy argument in fact shows that a k-partial $(k+1)$-coloring always exists. Initialize all vertices to color 1 and consider each vertex sequentially. For each considered vertex v, let $C(v)$ be the set of colors present in the neighborhood of v. If $|C(v)| = k+1$, then

© Avinandan Das;
licensed under Creative Commons License CC-BY 4.0

the vertex retains it's color. Otherwise, it recolors itself with a color from the set $[k+1] \setminus C(v)$. The resulting coloring is k-partial because (1) node v has at least $\min\{k, \deg(v)\}$ neighbors with a color different from its own color when it adopts its final color, and (2) two neighboring vertices with different colors at some time t during the execution of the greedy algorithm will remain with different colors at any time $t' \geq t$.

A central question in this paper is whether the semi-streaming tractability known for $(\Delta + 1)$- and Δ-proper coloring [2, 3] extends to these partial-coloring generalizations as well.

A simple Streaming Algorithm. As a warm-up, a simple one-pass approach already yields a k-partial $(k + o(k))$-coloring.

A useful lens is to view partial coloring through a *witness* subgraph. Suppose we find a spanning subgraph $G' \subseteq G$ such that every vertex v retains at least $\min\{k, \deg_G(v)\}$ incident edges in G'. Then any *proper* coloring of G' with c colors is automatically a k-partial c-coloring of G: each vertex has enough neighbors inside G', and properness makes all witness edges bichromatic.

We run the *filter* from Section 5.2 to construct online a spanning subgraph $G_f = (V, E_f)$: when an edge $\{u, v\}$ arrives, we accept it into E_f iff the current degree of u in G_f is $< k$ or the current degree of v in G_f is $< k$. Lemma 10 in Section 5.2.2 proves that G_f is a valid witness, and by Lemma 14, that it is k-degenerate. Hence we may apply the degeneracy-based semi-streaming coloring algorithm [6] to properly color G_f with $k + o(k)$ many colors in semi-streaming.

Challenges in getting down to $k + 1$ colors. The witness perspective also clarifies why achieving $k + 1$ colors requires new ideas.

1. **Selecting the right streamed edges.** In semi-streaming we must decide online which edges to keep in the witness G' under $\widetilde{O}(n)$ space. The greedy filter above yields only a k-degenerate witness. However, properly coloring an *arbitrary* k-degenerate graph with $k + 1$ colors is not feasible in one-pass semi-streaming space (refer to [6]). Thus, the first challenge is to maintain a witness that is not merely sparse, but structured enough that $(k + 1)$-coloring becomes tractable.
2. **Even with a good witness, what coloring approach applies?** Assume we somehow manage to keep a "nice" witness G'. It is still not immediate what algorithmic approach should be deployed to color it with $k + 1$ colors in one pass. At first glance one might hope that the witness has Δ bounded by k and hence that one can adapt $(\Delta + 1)$-coloring ideas. However, this intuition can fail sharply: there exist graphs for which any k-partial $(k + 1)$-coloring essentially forces a proper coloring, while the graph still contains vertices of degree strictly larger than k; see Fig. 1 for such examples. Therefore, one cannot rely on a maximum-degree bound, and the coloring algorithm must exploit a different structural handle than naive $\Delta \leq k$ reasoning.

***k*-partial *k*-coloring.** A tight classical characterization of when a graph admits a proper Δ-coloring is given by Brooks' theorem (and its many variants and proofs); in particular, for a connected graph with maximum degree Δ, a proper Δ-coloring exists unless the graph is a clique (or an odd cycle when $\Delta = 2$); see, e.g., the survey [8]. More recently, it was proven in [7] that for every fixed constant $k \geq 3$, deciding whether a graph admits a k-partial k-coloring is NP-complete; their reduction replaces each edge of the input graph by a constant-size "k-edge gadget" (see Section 6 and Fig. 1). At the same time, the problem of k-partial k-coloring for any constant k is trivially semi-streaming tractable; implement the

simple streaming algorithm discussed in Section 1 and store the filtered graph G_f (which has degeneracy bounded by k which is a constant and therefore can be stored offline in $\widetilde{O}(n)$ space) and find a coloring offline. This phenomenon is not unique: there are NP-hard problems that are nonetheless trivial in semi-streaming space, e.g. deciding whether a cubic graph is 3-edge-colorable is NP-complete [12], cubic graphs have only $3n/2$ edges and hence can be stored exactly in semi-streaming space. Consequently, the question remains whether the one-pass semi-streaming tractability known for proper Δ-coloring [3] extends to k-partial k-coloring as well.

2 Related Work

Graph problems in streaming are commonly studied in the *semi-streaming* model introduced in [10], where the graph arrives as an edge stream and the algorithm uses $\tilde{O}(n)$ space. For background on techniques and representative results in graph streams, see survey [14].

A central tool behind modern sublinear graph-coloring algorithms is the *palette sparsification theorem* [2], which shows that sampling $O(\log n)$ colors per vertex from $\{1, \ldots, \Delta + 1\}$ typically preserves the existence of a proper $(\Delta + 1)$-coloring, enabling single-pass semi-streaming algorithms in dynamic streams. Subsequent work studied palette sparsification in special classes of graphs, like in triangle free graphs with a smaller palette of colors as well as generalizing it to $(\deg +1)$-coloring [1]. It was further generalized to $(\deg +1)$-list coloring in [11]. Recently a substantially simpler (albeit slightly weaker) proof of palette sparsification has appeared in [4].

Proper Δ-coloring was also studied in the streaming framework and was proven to be semi-streaming tractable in [3]. Coloring was also studied w.r.t *degeneracy* of a graph. In [6], a semi-streaming algorithm was designed which colors a graph with degeneracy κ in $\kappa + o(\kappa)$ many colors. They also proved that $\kappa + 1$ coloring of a n-vertex graph requires $\Omega(n^2)$ bits of space. We adapt their lower bound proof to the partial coloring setting in Section 6.

The notion of *partial* (or *k-partially proper*) coloring has appeared previously in distributed graph algorithms before [5,7,9,16]. Specifically, [7] studied the problem of k-partial k-coloring and proved that the problem is NP-complete and proved that the problem is"global" for every constant k.

Our work complements these results by establishing an analogous *one-color* threshold phenomenon in the semi-streaming model.

3 Our Results

The results of this paper demonstrate a sharp *one-color dichotomy* in the semi-streaming setting: allowing k+1 colors makes k-partial coloring tractable in semi-streaming space, whereas restricting to exactly k colors can force super-linear memory.

▶ **Theorem 1.** *There exists a one-pass randomized semi-streaming algorithm that, with high probability, produces a k-partial coloring of an input graph G using $k + 1$ colors, where $k \in \mathbb{N}$ is given as input prior to the stream and G is presented as an insertion-only stream.*

Section 5 is dedicated to proving the theorem. In section 5.1, we begin by identifying a sparse spanning subgraph $G' \subseteq G$ that serves as a *witness* for partial coloring: a proper $(k + 1)$-coloring of G' immediately induces a k-partial $(k + 1)$-coloring of G. We then prove a palette sparsification theorem for proper $(k + 1)$-coloring of the witness graph G' and finally in Section 5.2, we adapt the sparsification lemma to obtain a one-pass semi-streaming algorithm for the coloring.

In Section 6, we augment this upper bound by a fine-grained super linear lower bound for k-partial k-coloring. More specifically, we prove the following theorem.

▶ **Theorem 2.** *For every constant $\varepsilon \geq 0$, let G be an n-vertex graph presented as an adversarial insertion-only stream, and let the partial coloring parameter be $k = \Theta(n^{\frac{1}{3+\varepsilon}})$. Any one-pass randomized streaming algorithm $\mathcal{A}$ that solves k-partial k-coloring on G with probability at least $2/3$ requires $\Omega\left(n^{1+\frac{1}{3+\varepsilon}}\right)$ bits of memory. In particular, for $\varepsilon = 0$, the algorithm requires $\Omega(n^{4/3})$ bits.*

Without loss of generality, we assume that $k = \omega(\text{poly} \log n)$. Otherwise, the problem becomes trivially semi-streaming tractable; run the filter of the simple algorithm in Sec 1 and store the edges which pass the filter. The resultant graph will have degeneracy bounded by k and therefore, can be stored and colored offline in semi-streaming space.

Technical Overview

Semi-Streaming Algorithm. Our approach is based on the observation that partial coloring can be viewed as a proper coloring problem on a suitable spanning subgraph. By definition, any k-partial coloring of a graph G implicitly induces a proper coloring on a subgraph $G' \subseteq G$, which acts as a *witness* for partial coloring: if a coloring is proper on G', then every vertex has at least $\min\{k, \deg_G(v)\}$ neighbors colored differently in G.

We exploit this perspective by explicitly constructing such a witness subgraph. Starting from the input graph G, we repeatedly delete edges whose both endpoints currently have degree strictly larger than k. This process terminates with a subgraph G' in which every edge has at least one endpoint of degree at most k. A key property of this construction is that any proper coloring of G' is automatically a valid k-partial coloring of the original graph G.

Once the problem is reduced to properly coloring the witness graph G', we aim to apply *palette sparsification*, following the general strategy of [2]. However, unlike the standard $(\Delta + 1)$-coloring setting, palette sparsification does not immediately apply here: vertices sample colors from the fixed palette $[k + 1]$, rather than from degree-dependent palettes. Moreover, we might end up in situations where the graph G' has to be properly colored but $k < \Delta(G')$ (refer to Fig. 1). This mismatch prevents a direct application of existing sparsification results.

To overcome this difficulty, we adapt palette sparsification to the structure of the witness graph. The key idea is to exploit the degree asymmetry enforced by the witness construction. We partition the vertices of G' into *high-degree* and *low-degree* vertices. By construction, the set of high-degree vertices (those with degree greater than k) forms an independent set. This allows us to color all high-degree vertices first using a small subset of colors. In particular, we show that they can be colored with at most $k/4$ colors with high probability, using only their randomly sampled palettes.

After fixing the colors of the high-degree vertices, we proceed to color the low-degree vertices. Each low-degree vertex has at most k neighbors in total, and only a small number of colors are forbidden by already-colored neighbors. Although the palettes of these vertices were sampled from $[k + 1]$, we show that the *effective palettes* obtained after removing forbidden colors remain sufficiently large and retain the necessary randomness. This enables the application of a variant of palette sparsification theorem for $(\deg+1)$-list coloring [11, Theorem 4] to complete the coloring.

To translate this existential argument into a streaming algorithm, we introduce a three-phase procedure. In *Phase 1 (Filtering)*, edges are discarded once both endpoints have accumulated degree k, ensuring that the maintained graph remains a witness. In *Phase 2 (Palette Sparsification)*, edges whose endpoints have disjoint sampled palettes are discarded, since they cannot induce color conflicts; only potentially constraining edges are stored. Finally, in *Phase 3 (Offline Solving)*, the algorithm solves the resulting demand-partial list coloring instance offline. By the adapted palette sparsification argument, a solution exists with high probability, and any such solution corresponds to a proper coloring of G' and hence to a valid k-partial coloring of G.

Lower Bound. We base ourselves on the lower bound technique of [6] who prove that there exists a κ such that coloring an input n-vertex κ-degenerate graph G with $\kappa+1$ colors requires $\Omega(n^2)$ bits in insertion-only stream. Their proof follows a standard reduction from the communication problem INDEX where Alice is given a vector $X \in \{0,1\}^N$ and Bob, an index $i \in [N]$ and Bob outputs the value X_i. They follow a standard reduction procedure of shaping X as a square matrix $A[n \times n]$ and the index is mapped as (g,h). These are then encoded as a graph G' where two sets of vertices U and W represent the rows and columns of A such that for vertices $u_i \in U$ and $w_j \in W$, $\{u_i, w_j\} \in E(G')$ if and only if $A[i,j]=1$. The graph G' is encoded in such a way that *for any $\kappa+1$ coloring γ, for vertices $u_g \in U$ and $w_h \in W$, $\gamma(u_g) \neq \gamma(w_h)$ if and only if $A[g,h]=1$.*

We follow the same strategy and reshape the vector X into a rectengular matrix $A[(k-1) \times \ell]$ and the index i mapped as (t,q) and Alice and Bob construct a graph G encoding A and (t,q). Let R and C be the row and column vertices in G corresponding to A. Alice ensures that for each vertex $v \in C$: 1) $\deg_G(v) = k-1$ and 2) $\chi(N_G[v]) = [k]$ and for each vertex $u \in R$, $|\chi(N_G[u])| = k-1$. Bob, based on the index (t,q) adds edges to the graph ensuring that $\chi(N_G[r_t]) = [k-1]$. This is ensured via the special gadgets *edge-gadgets* and *color repeaters* which, when added between two vertices force the two vertices to take different and same colors respectively. These properties, along with the fact that any k-partial k-coloring χ will color all the incident edges of v properly w.r.t it ensures that G has to satisfy the property that for vertices $r_t \in R$ and $c_q \in C$, $\chi(r_t) \neq \chi(c_q)$ if and only if $A[t,q]=1$.

Any streaming algorithm acts as a one-way communication protocol and in the end, the coloring of the vertices reveal the bit $A[t,q]$. The space bound follows from the fact that the $|V(G)| = O(k+\ell)$, setting $k^{3+\varepsilon} = O(\ell)$ and that INDEX requires $\Omega(k\ell)$ bit one-way communication bandwidth.

4 Preliminaries

We write $[k] := \{1,2,\ldots,k\}$. All logarithms are base two unless stated otherwise. All graphs considered in this paper are simple, undirected, and have n vertices.

Throughout the paper, we sample colors *independently and uniformly at random with replacement* from a given palette. Formally, sampling with replacement from a finite set U means that each draw is an independent random variable distributed uniformly over U. Consequently, the same color may appear multiple times in a sampled list.

▶ **Lemma 3.** *Let X be a color sampled uniformly at random from $[k+1]$. Let $U_u \subseteq [k+1]$ be any fixed subset. Conditioned on the event that $X \in U_u$, the color X is uniformly distributed over U_u.*

Proof. For any $c \in U_u$, $\Pr[X=c \mid X \in U_u] = \frac{\Pr[X=c]}{\Pr[X \in U_u]} = \frac{1/(k+1)}{|U_u|/(k+1)} = \frac{1}{|U_u|}$. ◀

We use standard Chernoff bounds for sums of independent Bernoulli random variables [15].

Let $X = \sum_{i=1}^{m} X_i$, where the X_i are independent and take values in $\{0, 1\}$. Then, for any $0 < \delta < 1$,

$$\Pr\big[X \leq (1-\delta)\mathbb{E}[X]\big] \;\leq\; \exp\left(-\frac{\delta^2}{2}\mathbb{E}[X]\right).$$

Degeneracy. A graph $G = (V, E)$ is *d-degenerate* if every non-empty subgraph $H \subseteq G$ contains a vertex of degree at most d.

Equivalently, G is d-degenerate if there exists an ordering of the vertices $v_1, \ldots, v_n$ such that for every i, vertex v_i has at most d neighbors among $\{v_{i+1}, \ldots, v_n\}$. We orient each edge $\{v_i, v_j\}$ with $i < j$ from v_i to v_j. The *outdegree* of v_i is defined as

$$\text{outdeg}(v_i) = \big|\{\, v_j : j > i \text{ and } \{v_i, v_j\} \in E \,\}\big|.$$

With this orientation, G is d-degenerate if and only if there exists an ordering such that $\text{outdeg}(v_i) \leq d$ for all $i \in [n]$.

Semi-Streaming Algorithms. We work in the *semi-streaming model* for graph algorithms.

An input graph $G = (V, E)$ is presented as an insertion-only stream of edges. The algorithm knows the vertex set V apriori and processes the stream of edges in one pass while maintaining a memory state of size at most $\widetilde{O}(n)$ bits, where polylogarithmic factors in n are suppressed.

The algorithm may be randomized and succeeds with high probability (i.e. with probability at least $(1 - 1/n^c)$ for some constant c). At the end of the stream, the algorithm outputs a solution to the given graph problem.

Communication Complexity. We use communication complexity to prove lower bounds for streaming algorithms.

In the *one-way communication model*, there are two players, Alice and Bob. Alice receives an input x, Bob receives an input y, and Alice sends a single message to Bob. Based on this message and his input, Bob must compute a function $f(x, y)$.

The communication cost is the number of bits sent from Alice to Bob. Protocols may be randomized and are required to succeed with probability at least $2/3$.

A standard connection between streaming algorithms and communication complexity is that any one-pass streaming algorithm using S bits of space yields a one-way communication protocol with communication cost S, where Alice simulates the algorithm on the first part of the stream and sends the memory state to Bob.

Index. The INDEX problem is defined as follows: Alice holds a bit string $X \in \{0, 1\}^N$, Bob holds an index $j \in [N]$, and Bob must output the bit X_j.

It is well known that any one-way randomized protocol for Index that succeeds with probability at least $2/3$ requires $\Omega(N)$ bits of communication [13].

5 Semi-Streaming Algorithm for k-partial $(k+1)$-coloring

This section is dedicated to design, correctness and analysis of a single pass insertion-only semi-streaming algorithm for partial coloring. Specifically, we prove the following theorem.

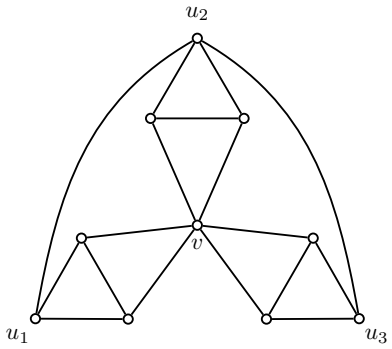


Figure 1 Example of a graph where 3-partial coloring implies a proper coloring. Observe that the vertices v and u_2 have degree greater than 3 but are surrounded by degree 3 vertices which enforce a proper coloring on them.

▶ **Theorem 4.** *There exists a one-pass randomized semi-streaming algorithm that, with high probability, produces a k-partial coloring of an input graph G using $k+1$ colors, where $k \in \mathbb{N}$ is given as input prior to the stream and G is presented as an insertion-only stream.*

5.1 Palette Sparsification for Partial Coloring

▶ **Definition 5** (Witness for Partial Coloring)**.** *Let $G = (V, E)$ be a graph and k a positive integer. A* witness for k-partial coloring *is a subgraph $G' \subseteq G$ obtained by the following procedure:*

1. *Initialize $G' = G$.*
2. *While there exists an edge $\{u, v\} \in E(G')$ such that $\deg_{G'}(u) > k$ and $\deg_{G'}(v) > k$, remove $\{u, v\}$ from $E(G')$.*

The resultant graph satisfies that for every edge $\{u, v\} \in E(G')$, $\min\{\deg_{G'}(u), \deg_{G'}(v)\} \leq k$.

▶ **Lemma 6.** *Let G' be a witness for k-partial coloring of G. If χ is a proper coloring of G', then χ is a k-partial coloring of G.*

Proof. We first establish a lower bound on the degrees in G'. We claim that for every vertex $v \in V(G)$, $\deg_{G'}(v) \geq \min\{k, \deg_G(v)\}$.

Consider the edge deletion procedure defining G'. An edge incident to v is removed only if $\deg_{current}(v) > k$. Consequently, the degree of v can never be reduced to a value less than k, as the removal condition would fail before such a reduction could occur.

- If $\deg_G(v) \leq k$, the condition $\deg(v) > k$ is never met, so no edges incident to v are ever removed. Thus, $\deg_{G'}(v) = \deg_G(v)$.
- If $\deg_G(v) > k$, edges incident to v may be removed, but the process must halt for v if its degree reaches k. Thus, $\deg_{G'}(v) \geq k$.

Combining these cases, we have $\deg_{G'}(v) \geq \min\{k, \deg_G(v)\}$.

Now, since χ is a proper coloring of G', v is assigned a color distinct from all its neighbors in G'. The number of such neighbors is $\deg_{G'}(v)$. Using our lower bound, the number of neighbors of v in G colored differently than v is at least $\min\{k, \deg_G(v)\}$. This satisfies the definition of a k-partial coloring. ◀

Our aim now is to prove palette sparsification for the witness graph G'. Before proceeding to prove the theorem, we rephrase the palette sparsification theorem for (deg +1)-list coloring [11, Theorem 4] which we will use as a blackbox in our proof.

▶ **Theorem 7** (Palette sparsification for $(\deg+1)$-list coloring)**.** *Let G be an n-vertex graph. Suppose each vertex $v \in V(G)$ is assigned a list $L(v)$ of $\deg(v)+1$ colors. For each vertex v, independently sample a subset $L'(v) \subseteq L(v)$ of size $\Theta(\log^2 n)$ uniformly at random. Then, with high probability, there exists a proper coloring*

$$\mathcal{C}\colon V(G) \to \bigcup_{v \in V(G)} L(v)$$

such that $\mathcal{C}(v) \in L'(v)$ for every vertex $v \in V(G)$.

Without loss of generality, for Theorem 7, we assume that for each vertex $v \in V$, $|L'(v)| = \alpha \log^2 n$ for some fixed constant α and each color in $L'(v)$ is independently sampled uniformly at random with replacement from the palette of $L(v)$.

▶ **Theorem 8.** *Given an n-vertex graph G and a positive integer k, let G' be the witness for k-partial coloring of G. Suppose that for every vertex $v \in V(G')$, we sample a list $L(v)$ of $\Theta(\log^2 n)$ colors uniformly at random from the palette $[k+1]$. Then, with high probability, there exists a proper $(k+1)$-coloring $\mathcal{C}'\colon V(G') \to [k+1]$ of G' such that $\mathcal{C}'(v) \in L(v)$ for all $v \in V(G')$.*

Proof. We prove the existence of such a coloring using a two-phase randomized procedure. First, for every vertex $v \in V(G')$, we split the sampled list $L(v)$ into two disjoint sub-lists $L_1(v)$ and $L_2(v)$ such that $|L_1(v)| = C_1 \log n$ and $|L_2(v)| = C_2 \log^2 n$ for sufficiently large constants C_1, C_2.

We partition the vertex set $V(G')$ into two sets: $S = \{v \in V(G') \mid \deg_{G'}(v) > k\}$ and $R = V(G') \setminus S$. By the definition of a witness (specifically the constraint that for every edge $\{u,v\}$, $\min\{\deg_{G'}(u), \deg_{G'}(v)\} \leq k$), no two vertices with degree strictly greater than k can be adjacent. Consequently, S is an independent set in G'.

The coloring proceeds in two phases:

- **Phase 1 (Coloring S):** We assign colors to vertices in S using only the palettes $L_1(v)$.
- **Phase 2 (Coloring R):** We extend the coloring to R using the palettes $L_2(v)$.

Phase 1: Coloring the Independent Set S. We claim that with high probability, S can be properly colored using a subset of colors $T \subseteq [k+1]$ of size at most $k/4$, such that for every $v \in S$, the assigned color belongs to $L_1(v) \cap T$.

To show this, we construct a bipartite *palette graph* $P = (S \cup [k+1], E_P)$, where an edge $\{v, c\}$ exists if and only if $c \in L_1(v)$. A subset of colors $T \subseteq [k+1]$ is said to *dominate* S in P if every vertex $v \in S$ has at least one neighbor in T. If such a set T exists, we can define a valid coloring χ_1 for S by setting $\chi_1(v) = \min\{c \in T \cap L_1(v)\}$. Since S is an independent set, no conflicts can arise between vertices in S regardless of the color choices.

We now show that there exists $T \subseteq [k+1]$ of size $k/4$ that dominates S with high probability. Let T be an arbitrarily chosen but fixed subset of $[k+1]$ with $|T| = k/4$. For any fixed vertex $v \in S$, the probability that $L_1(v) \cap T = \emptyset$ is

$$\mathbb{P}[L_1(v) \cap T = \emptyset] \leq \left(1 - \frac{|T|}{k+1}\right)^{|L_1(v)|} \approx \left(1 - \frac{1}{4}\right)^{C_1 \log n} \leq e^{-0.25 C_1 \log n} = n^{-0.25 C_1}.$$

By choosing C_1 sufficiently large (e.g., $C_1 \geq 12$), we ensure this probability is at most n^{-3}. Taking a union bound over all vertices in S (where $|S| \leq n$), the probability that there exists any vertex in S not dominated by T is at most $n \cdot n^{-3} = n^{-2}$. Thus, with high probability, there exists a dominating set T of size $k/4$, and consequently a valid coloring χ_1 for S using only colors from T.

Phase 2: Extending the coloring to $R = V \setminus S$. Let χ_1 be the coloring of S obtained in Phase 1. Recall that Phase 1 ensures the set of colors used in S, denoted $C(S)$, satisfies $|C(S)| \leq k/4$.

We extend this coloring to R. For each vertex $u \in R$, let $C_{\text{forb}}(u) = \{\chi_1(v) \mid v \in N_{G'}(u) \cap S\}$ be the set of colors forbidden by neighbors in S. We define the target palette for u as $U_u = [k+1] \setminus C_{\text{forb}}(u)$.

Since $u \in R$, we know $\deg_{G'}(u) \leq k$. The number of forbidden colors from S is at most the number of neighbors u has in S, which is $\deg_{G'}(u) - \deg_R(u)$. Therefore, it follows easily that $|U_u| \geq \deg_R(u) + 1$ meaning that a valid extension exists if we had access to U_u and coloring R reduces to solving (deg +1)-list coloring.

At this point, we cannot directly apply Theorem 7 to R as it requires that for vertex u, the random list of size at least $\alpha \log^2 n$ be sampled uniformly from the available palette U_u for some constant α while $L_2(u)$ is sampled from $[k+1]$. Let $L_{\text{eff}}(u) = L_2(u) \cap U_u$. We claim that $L_{\text{eff}}(u)$ satisfies the that

1. Each color in $L_{\text{eff}}(u)$ is uniformly distributed over U_u.
2. $|L_{\text{eff}}(u)| \geq \alpha \log^2 n$ with high probability .

The first item follows immediately from Lemma 3.

Now, we show the second item. Since $C_{\text{forb}}(u) \subseteq C(S)$ and $|C(S)| \leq k/4$, we have $|U_u| \geq \frac{3}{4}(k+1)$. The size $|L_{\text{eff}}(u)|$ follows a Binomial distribution $B(|L_2(u)|, p)$ with $p = \frac{|U_u|}{k+1} \geq 3/4$. Fix $C_2 = 8\alpha/3$. Using a Chernoff bound, with $|L_2(u)| = (8/3)\alpha \log^2 n$ and $\mathbb{E}[|L_{\text{eff}}(u)|] \geq 2\alpha \log^2 n$, the effective list size $|L_{\text{eff}}(u)|$ is at least $\alpha \log^2 n$ with high probability. Applying union bound, this holds to every vertex in R with high probability.

Since for every vertex $w \in R$, $|L_{\text{eff}}(w)|$ is sufficiently large and conditionally uniform over available palette U_w, we can invoke the Palette Sparsification Theorem on the induced subgraph $G'[R]$ with palettes U_w. This guarantees that a valid coloring χ_2 consistent with the lists exists with high probability.

We now combine the results of both phases to bound the total failure probability. Let E_1 denote the event that Phase 1 successfully colors S with a subset of colors T such that $|T| \leq k/4$. We established that $\mathbb{P}[E_1] \geq 1 - n^{-2}$.

Conditioned on the occurrence of E_1, the coloring χ_1 and the forbidden sets $C_{\text{forb}}(u)$ are well-defined. Let $E_{2|1}$ denote the event that Phase 2 successfully extends this specific coloring to R using the lists L_2. Our analysis in Phase 2 demonstrates that for any fixed successful outcome of Phase 1,

$$\mathbb{P}[E_{2|1}] \geq 1 - n^{-c} \text{ for some positive integer } c.$$

The event that a valid $(k+1)$-coloring exists for the entire graph G' corresponds to the joint occurrence of both phases succeeding. By the chain rule of probability:

$$\mathbb{P}[\text{Success}] = \mathbb{P}[E_1] \cdot \mathbb{P}[E_{2|1}] \geq (1 - n^{-2})(1 - n^{-c}) \geq 1 - n^{-O(1)}.$$

Thus, with high probability, there exists a proper $(k+1)$-coloring $\mathcal{C}'$ of G' such that $\mathcal{C}'(v) \in L(v)$ for all $v \in V(G')$. ◀

5.2 Algorithm

We now describe our one-pass semi-streaming algorithm for computing a k-partial $(k+1)$-coloring. Before describing the algorithm, we first define *demand-partial list coloring*.

Algorithm 1 One-pass semi-streaming algorithm for k-partial $(k+1)$-coloring.

Input: Graph $G = (V, E)$ given as an insertion-only stream, integer k
Output: A k-partial $(k+1)$-coloring of G (with high probability)
Let $s := C \log^2 n$ for a sufficiently large constant C. For each $v \in V$, sample a list $L(v)$ of s colors independently and uniformly at random from $[k+1]$ (with replacement)
Initialize counters $\deg(v) \leftarrow 0$ and $\mathsf{free}(v) \leftarrow 0$ for all $v \in V$
Initialize an empty graph H
for each streamed edge $\{u, v\} \in E$ **do**
    if $\deg(u) < k$ **or** $\deg(v) < k$ **then**
        $\deg(u) \leftarrow \deg(u) + 1$
        $\deg(v) \leftarrow \deg(v) + 1$
        if $L(u) \cap L(v) = \emptyset$ **then**
            $\mathsf{free}(u) \leftarrow \mathsf{free}(u) + 1$
            $\mathsf{free}(v) \leftarrow \mathsf{free}(v) + 1$
        else
            Store the edge $\{u, v\}$ in H
        end if
    end if
end for
for each $v \in V$ **do**
    $\mathsf{demand}(v) \leftarrow \max\{0, \min\{k, \deg(v)\} - \mathsf{free}(v)\}$
end for
Offline: find (by brute force) a coloring $\chi : V \to [k+1]$ which is a solution for the demand-partial list coloring instance (H, demand) such that $\chi(v) \in L(v)$ for all $v \in V$.
return χ

▶ **Definition 9** (Demand-partial list coloring.). *Let $H = (V, F)$ be a graph and let each vertex $v \in V$ have a list $L(v) \subseteq [k+1]$. Let $\mathsf{demand} : V \to [k+1]$ be a demand function. A coloring $\chi : V \to [k+1]$ is a* solution to the demand-partial list coloring instance (H, demand) *if*
1. $\chi(v) \in L(v)$ *for every $v \in V$, and*
2. *for every vertex $v \in V$, $\big|\{\, x \in N_H(v) \;:\; \chi(x) \neq \chi(v) \,\}\big| \;\geq\; \mathsf{demand}(v)$.*

In other words, $\mathsf{demand}(v)$ specifies how many neighbors of v in H must end up with a color different from $\chi(v)$; we do not require χ to be a proper coloring of H.

5.2.1 Description of the Algorithm

The algorithm knows the parameter k apriori and the input is an insertion-only stream of edges of a n-vertex graph $G = (V, E)$. Let $s := C \log^2 n$ for a sufficiently large constant C. Before the stream starts, each vertex $v \in V$ independently samples a list $L(v)$ of s colors from $[k+1]$ uniformly with replacement. For each vertex $v \in V$, the algorithm maintains two counters $\deg(v)$ and $\mathsf{free}(v)$ both initialized at 0.

The algorithm has three phases. As an edge $\{u, v\}$ streams in, the algorithm processes it in two phases. The third phase is post-processing where the final coloring of the graph is computed.

Phase 1: Filtering. if $\deg(u) < k$ or $\deg(v) < k$, the edge $\{u, v\}$ proceeds to the next phase and the degrees are updated as follows:

$$\deg(u) \leftarrow \deg(u) + 1, \qquad \deg(v) \leftarrow \deg(v) + 1.$$

If $\deg(u) \geq k$ and $\deg(v) \geq k$, the edge is ignored and no state changes.

Phase 2: Sparsification. Conditioned on passing the filter, the algorithm checks whether the sampled lists are disjoint. If $L(u) \cap L(v) = \emptyset$, then regardless of the eventual choices $\chi(u) \in L(u)$ and $\chi(v) \in L(v)$, we necessarily have $\chi(u) \neq \chi(v)$. Hence this edge does not need to be stored; instead we increment

$$\mathsf{free}(u) \leftarrow \mathsf{free}(u) + 1, \qquad \mathsf{free}(v) \leftarrow \mathsf{free}(v) + 1.$$

Otherwise, if $L(u) \cap L(v) \neq \emptyset$, the edge may constrain the final solution and the algorithm stores $\{u, v\}$.

Phase 3: Post-Processing. Let the graph stored after the end of the stream be $H = (V, F)$. For each vertex $v \in V$, the algorithm defines the demand as

$$\mathsf{demand}(v) \;:=\; \max\big\{0,\ \min\{k, \deg(v)\} - \mathsf{free}(v)\big\}.$$

The algorithm computes offline the coloring $\chi : V \to [k+1]$ for the demand-partial list coloring (H, demand) where for each vertex $v \in V$, $\chi(v) \in L(v)$.

▶ Remark. The algorithm works only in the insertion-only framework. The key challenge in adapting it to dynamic setup is the implementation of the filter in Phase 1.

5.2.2 Correctness

▶ **Lemma 10.** *Let G' be the graph consisting of the edges that pass the filter of Algorithm 1. If χ is a k-partial coloring of G', then χ is a k-partial coloring of G.*

Proof. Fix an arbitrary vertex $v \in V$. There are two cases.

Case 1: $\deg_G(v) < k$. Each edge that arrives and is incident on it passes through the filter. Hence, $N_G(v) = N_{G'}(v)$ and χ colors both neighborhoods properly w.r.t v and trivially satisfies the condition of valid k-partial coloring on v.

Case 2: $\deg_G(v) \geq k$. The first k incident edges in the stream pass through the filter. Hence, $\deg_{G'}(v) \geq k$. Given that $N_{G'}(v) \subseteq N_G(v)$ and χ is a k-partial coloring of G', at least k vertices in $N_G(v)$ are also colored differently than v in G satisfying the condition of k-partial coloring. ◀

▶ **Lemma 11.** *If χ is a k-partial coloring of G' such that $\chi(x) \in L(x)$ for all $x \in V(G')$, then for every vertex $v \in V(G')$,*

$$\big|\{u \in N_H(v) : \chi(u) \neq \chi(v)\}\big| \;\geq\; \mathsf{demand}(v).$$

Proof. By the description of the algorithm, at the end of the filtering phase, for each vertex $v \in V(G')$, $\deg_{G'}(v) = \deg(v)$.

Fix a vertex v. Split its neighborhood in G' into stored and discarded parts:

$$N_{G'}(v) = N_H(v) \;\cup\; \big(N_{G'}(v) \setminus N_H(v)\big).$$

For every $u \in N_{G'}(v) \setminus N_H(v)$ we have $L(u) \cap L(v) = \emptyset$ by definition of H, and hence $\chi(u) \neq \chi(v)$. Therefore,

$$\big|\{u \in N_{G'}(v) : \chi(u) \neq \chi(v)\}\big| = \mathsf{free}(v) \;+\; \big|\{u \in N_H(v) : \chi(u) \neq \chi(v)\}\big|.$$

Since χ is k-partial on G', the left-hand side is at least $\min\{k, \deg_{G'}(v)\}$. Rearranging yields

$$\begin{aligned}\big|\{u \in N_H(v) : \chi(u) \neq \chi(v)\}\big| &\geq \min\{k, \deg_{G'}(v)\} - \mathsf{free}(v) \\ &\geq \max\{0, \min\{k, \deg_{G'}(v)\} - \mathsf{free}(v)\} = \mathsf{demand}(v).\end{aligned}$$ ◀

▶ **Lemma 12.** *With high probability over the sampled lists $\{L(v)\}_{v\in V}$, there exists a coloring $\chi : V \to [k+1]$ such that*

- *$\chi(v) \in L(v)$ for all $v \in V$, and*
- *$\left|\{u \in N_H(v) : \chi(u) \neq \chi(v)\}\right| \geq \mathsf{demand}(v)$ for all $v \in V$.*

Proof. Let W be any witness for k-partial coloring of G'. By Theorem 8 applied to W, with high probability there exists a proper $(k+1)$-coloring χ of W such that $\chi(v) \in L(v)$ for all $v \in V$.

By Lemma 10, χ is a k-partial coloring of G' and is list respecting. Finally, Lemma 11 implies that χ satisfies the demand constraints on H. ◀

▶ **Theorem 13.** *Algorithm 1 outputs a k-partial $(k+1)$-coloring of G with high probability.*

Proof. By Lemma 12, with high probability there exists at least one list-respecting coloring χ that satisfies the demand constraints on H. The offline brute-force step finds such a coloring whenever it exists.

Assume the brute-force step outputs such a χ. Then, by construction, χ satisfies the demand constraints on H.

Fix an arbitrary vertex $v \in V$. Since every discarded edge of G' in the sparsification step of the algorithm has disjoint lists, every neighbor of v which contributes to $\mathsf{free}(v)$ is colored differently than it by χ.

$$\begin{aligned}|\{u \in N_{G'}(v) : \chi(u) \neq \chi(v)\}| &\geq \mathsf{free}(v) + \mathsf{demand}(v)\\ &\geq \mathsf{free}(v) + \max\{0, \min\{k, \deg_{G'}(v)\} - \mathsf{free}(v)\} \geq \min\{k, \deg_{G'}(v)\}\end{aligned}$$

Therefore, the constraints of k-partial coloring is satisfied by χ on G' and by Lemma 10, χ is a k-partial coloring of G. ◀

5.2.3 Space Guarantee

▶ **Lemma 14.** *Let $G' = (V, E')$ be the graph induced by the edges that pass the filter of Algorithm 1. Then G' is k-degenerate.*

Proof. We use the equivalent characterization: a graph is k-degenerate if and only if every non-empty subgraph has a vertex of degree at most k.

Let $S \subseteq V$ be any non-empty set of vertices and consider the induced subgraph $G'[S]$. Let t be the earliest time (in the stream) at which an edge with both endpoints in S is accepted *after* every vertex in S has already reached degree at least k in $G'[S]$. We claim that such a time t cannot exist.

Indeed, fix any time during the stream and suppose (for contradiction) that at this time every vertex $x \in S$ has current degree at least k *within* $G'[S]$. Then, in particular, each endpoint of any arriving edge $\{u, v\}$ with $u, v \in S$ has current degree at least k in the full graph G' as well (since $\deg_{G'}(x) \geq \deg_{G'[S]}(x)$ for all $x \in S$ at all times). Therefore the filter condition "$\deg_{G'}(u) < k$ or $\deg_{G'}(v) < k$" fails, and no further edge with both endpoints in S can ever be accepted into G'. This contradicts the definition of t.

Hence, at the end of the stream it cannot be the case that all vertices in S have degree at least $k+1$ in $G'[S]$. Equivalently, $G'[S]$ contains a vertex of degree at most k. Since S was arbitrary, every non-empty induced subgraph of G' has a vertex of degree at most k, and therefore G' is k-degenerate. ◀

▶ **Lemma 15.** *Let $s = C\log^2 n$. With high probability, $|E(H)| = O(n\log^4 n)$.*

Proof. Since the graph G' is k-degenerate (by Lemma 14), fix a k-degenerate ordering of G' and define acyclic orientation of the edges such that every vertex v satisfies $\text{outdeg}_{G'}(v) \leq k$.

Fix a vertex v. For each out-neighbor u of v in G', the edge $\{v, u\}$ is stored in H iff $L(v) \cap L(u) \neq \emptyset$. Conditioned on $L(v)$, these events are independent across different out-neighbors u, and $\Pr[L(v) \cap L(u) \neq \emptyset] \leq \frac{s^2}{k+1}$.

Hence $\text{outdeg}_H(v)$ is stochastically dominated by a binomial random variable with k trials and mean at most $s^2 = O(\log^4 n)$. A Chernoff bound implies that $\text{outdeg}_H(v) = O(\log^4 n)$ with high probability. A union bound over all $v \in V$ shows that with high probability, every vertex v has $\text{outdeg}_H(v) = O(\log^4 n)$ in this fixed orientation. Therefore H is $O(\log^4 n)$-degenerate and hence $|E(H)| \leq n \cdot O(\log^4 n)$ with high probability. ◀

6 Lower Bound: A Super-Linear Space Barrier

We establish that verifying a k-partial k-coloring requires super-linear space in the semi-streaming model. This stands in sharp contrast to $(k+1)$-coloring, which we have shown is tractable with $\tilde{O}(n)$ space. Specifically, we prove the following theorem.

▶ **Theorem 16.** *For every constant $\varepsilon \geq 0$, let G be an n-vertex graph presented as an adversarial insertion-only stream, and let the partial coloring parameter be $k = \Theta(n^{\frac{1}{3+\varepsilon}})$. Any one-pass randomized streaming algorithm $\mathcal{A}$ that solves k-partial k-coloring on G with probability at least $2/3$ requires $\Omega\left(n^{1+\frac{1}{3+\varepsilon}}\right)$ bits of memory. In particular, for $\varepsilon = 0$, the algorithm requires $\Omega(n^{4/3})$ bits.*

Our proof is a reduction from the one-way INDEX communication problem. Before proceeding with the proof, we construct two "gadgets" that force equality or inequality of colors between two vertices using the constraints of k-partial coloring and will be used in the construction of lower bound graphs for the reduction.

Gadgets

For a graph G and two distinct vertices $x, y \in V(G)$,

- **Edge-gadget $e_{x,y}$.** Introduce a k-clique $K = \{p_1, \ldots, p_k\}$; connect x to $p_1, \ldots, p_{k-1}$ and connect y to p_k.
- **Color-repeater $c_{x,y}$.** Introduce a $(k-1)$-clique $Q = \{q_1, \ldots, q_{k-1}\}$ and connect both x and y to all vertices of Q.

▶ **Lemma 17** (Edge-gadget forces inequality [7]). *In any k-partial k-coloring χ of $G \cup e_{x,y}$, we have $\chi(x) \neq \chi(y)$.*

Proof. Consider the clique $K = \{p_1, \ldots, p_k\}$ within the edge gadget. Every vertex $p \in K$ has degree exactly k in $G \cup e_{x,y}$. Therefore, by the definition of k-partial coloring, χ is a proper coloring on $e_{x,y}$. Since K induces a k-clique and the palette has size exactly k, the vertices of K must utilize all k distinct colors. Consequently, the set of colors assigned to the subset $\{p_1, \ldots, p_{k-1}\}$ consists of exactly $k-1$ distinct colors.

As the vertex x is adjacent to every vertex in $\{p_1, \ldots, p_{k-1}\}$, the coloring $\chi(x)$ must be the unique remaining color in the palette, which is precisely $\chi(p_k)$. Since y is adjacent to p_k, we must have $\chi(y) \neq \chi(p_k)$ and therefore, $\chi(x) \neq \chi(y)$ ◀

▶ **Lemma 18** (Color-repeater forces equality). *In any k-partial k-coloring χ of $G \cup c_{x,y}$, we have $\chi(x) = \chi(y)$.*

Proof. As the same arguments in lemma 17, the coloring χ is a proper on $c_{x,y}$ and the $(k-1)$-clique Q utilizes $k-1$ distinct colors and therefore, vertices x and y have only one possible color that can be assigned to them and hence, $\chi(x) = \chi(y)$. ◀

The Reduction from Index

Let Alice hold a boolean string $X \in \{0,1\}^N$ and Bob hold an index $j \in [N]$. We interpret X as a matrix A of dimensions $(k-1) \times \ell$, where $N = (k-1)\ell$. Bob's index corresponds to a specific entry (g,h) in this matrix.

Alice and Bob create graph $G = (V(G), E_A \sqcup E_B)$ based on A and (g,h) respectively. We now describe the graph G as follows.

Vertex Set. The vertex set V consists of four disjoint sets of vertices $U = \{u_1, \ldots, u_{k-1}\}$, $V = \{v_1, \ldots, v_{k-1}\}$, $W = \{w_1, \ldots, w_\ell\}$ and X which is a set of auxiliary vertices of cardinality $O(k^3)$ which would be used to create edge-gadgets or repeater gadgets.

Alice's stream.

1. For each distinct pair of vertices u_i and $u_{i'}$ with $i \neq i'$, insert the edge-gadget $e_{u_i,u_{i'}}$.
2. For each distinct pair of vertices v_i and $v_{i'}$ with $i \neq i'$, insert the edge-gadget $e_{v_i,v_{i'}}$.
3. For every $i \in [k-1]$ and $h \in [\ell]$: If $A[i,h] = 1$, insert the edge $\{u_i, w_h\}$, else insert the edge $\{v_i, w_h\}$.

Bob's stream. For every $t \in [k-1] \setminus \{g\}$, insert the color-repeater c_{u_t,v_t} and insert the edge-gadget e_{u_g,v_g}.

Alice uses $O(\binom{k}{2})$ many edge-gadgets (for connecting each pair of vertices in U and V respectively) and Bob uses $(k-2)$ many color-repeaters and 1 edge-gadget for his stream. They require $O(k^3)$ vertices to build these gadgets and utilize the vertices of X to do so (which has sufficiently many vertices).

▶ **Lemma 19.** *Let χ be any k-partial k-coloring of the graph G constructed above. Then the following properties hold:*

1. *The vertices $u_1, \ldots, u_{k-1}$ receive pairwise distinct colors, and the vertices $v_1, \ldots, v_{k-1}$ receive pairwise distinct colors.*
2. *For every $t \in [k-1] \setminus \{g\}$, we have $\chi(u_t) = \chi(v_t)$, while $\chi(u_g) \neq \chi(v_g)$.*
3. *Let α denote the unique color in the palette $[k]$ that does not appear among $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$. Then $\chi(v_g) = \alpha$.*

Proof. The edge-gadgets inserted among the vertices of $U = \{u_1, \ldots, u_{k-1}\}$ enforce $\chi(u_i) \neq \chi(u_{i'})$ for all $i \neq i'$ by Lemma 17. The same argument applies to the vertices of $V = \{v_1, \ldots, v_{k-1}\}$, proving (1).

For every $t \neq g$, the color-repeater c_{u_t,v_t} enforces $\chi(u_t) = \chi(v_t)$ by Lemma 18, while the edge-gadget e_{u_g,v_g} enforces $\chi(u_g) \neq \chi(v_g)$. This proves (2).

Since $u_1, \ldots, u_{k-1}$ receive pairwise distinct colors and the palette has size k, there is a unique color α not appearing among $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$. For every $t \neq g$, we have $\chi(v_t) = \chi(u_t)$ by part (2), and hence the vertices $\{v_t : t \neq g\}$ use exactly the colors $\{\chi(u_t) : t \neq g\}$. As the vertices of V must all receive distinct colors, the color $\chi(v_g)$ must differ from $\chi(v_t)$ for every $t \neq g$, and therefore must differ from all colors in $\{\chi(u_t) : t \neq g\}$. Moreover, since $\chi(u_g) \neq \chi(v_g)$ by part (2), the color $\chi(v_g)$ must also differ from $\chi(u_g)$. It follows that $\chi(v_g)$ differs from every color in $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$ and the only possible choice is the unique remaining color α, and hence $\chi(v_g) = \alpha$ proving (3). ◀

▶ **Lemma 20** (Forced color at w_h). *Let χ be any k-partial k-coloring of G, and let α be the unique color in $[k]$ not appearing among $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$. Then*

$$A[g,h] = 1 \iff \chi(w_h) = \chi(v_g) = \alpha.$$

Proof. By Lemma 19, the vertices $u_1, \ldots, u_{k-1}$ receive pairwise distinct colors, and $\chi(v_g) = \alpha$, where α is the unique color in $[k]$ not appearing among $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$. In particular, $\alpha \neq \chi(u_g)$.

By construction, $\deg(w_h) = k - 1 \leq k$, hence χ is proper on all edges incident to w_h. For each $i \neq g$, the vertex w_h is adjacent to exactly one of u_i or v_i, and Lemma 19(2) gives $\chi(u_i) = \chi(v_i)$. Therefore the neighbors of w_h contributed by rows $i \neq g$ realize exactly the $k-2$ distinct colors $\{\chi(u_i) : i \neq g\}$.

If $A[g,h] = 1$, then $\{u_g, w_h\} \in E$, so the neighbors of w_h realize all colors in $\{\chi(u_1), \ldots, \chi(u_{k-1})\}$. Hence the unique color in $[k]$ absent from the neighborhood of w_h is α, and therefore $\chi(w_h) = \alpha = \chi(v_g)$.

If $A[g,h] = 0$, then $\{v_g, w_h\} \in E$, so the neighbors of w_h realize the set $\{\chi(u_i) : i \neq g\} \cup \{\alpha\}$. The unique missing color is then $\chi(u_g)$, and hence $\chi(w_h) = \chi(u_g) \neq \chi(v_g)$.

Combining the two cases yields $A[g,h] = 1 \iff \chi(w_h) = \chi(v_g)$. ◀

Proof of Theorem 16. Assume there exists a one-pass randomized streaming algorithm $\mathcal{A}$ that, under the promise that the input graph is k-partial k-colorable, outputs such a coloring using S bits of memory with probability at least $2/3$.

Alice simulates $\mathcal{A}$ on her portion of the stream and sends the memory state to Bob. Bob continues the simulation on his portion and obtains a coloring χ.

By Lemma 20, Bob can recover the bit $A[g,h]$ by checking whether $\chi(w_h) = \chi(v_g)$ or $\chi(w_h) = \chi(u_g)$. Hence this yields a one-way randomized protocol for INDEX on $N = (k-1)\ell$ bits using S bits of communication and success probability at least $2/3$. Therefore $S = \Omega(N)$.

Let $\ell = \Theta(k^{3+\varepsilon})$. Then

$$n = |U| + |V| + |W| + O(k^3) = \Theta(k^{3+\varepsilon}), \qquad N = (k-1)\ell = \Theta(k^{4+\varepsilon}).$$

Thus $k = \Theta(n^{1/(3+\varepsilon)})$ and $S = \Omega(N) = \Omega\left(n^{1+\frac{1}{3+\varepsilon}}\right)$. For $\varepsilon = 0$, this gives the bound $\Omega(n^{4/3})$. ◀

References

1. Noga Alon and Sepehr Assadi. Palette sparsification beyond $(\delta + 1)$ vertex coloring. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, Virtual Conference, August 17-19, 2020*, volume 176 of *LIPIcs*, pages 6:1–6:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.6.
2. Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\delta + 1)$ vertex coloring. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019. doi:10.1137/1.9781611975482.48.
3. Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: A single-pass semi-streaming algorithm for δ-coloring. *TheoretiCS*, 2, 2023. doi:10.46298/THEORETICS.23.9.
4. Sepehr Assadi and Helia Yazdanyar. Simple sublinear algorithms for $(\delta + 1)$ vertex coloring via asymmetric palette sparsification. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms, SOSA 2025, New Orleans, LA, USA, January 13-15, 2025*, pages 1–8. SIAM, 2025. doi:10.1137/1.9781611978315.1.

5 Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. Locality of not-so-weak coloring. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2019. doi:10.1007/978-3-030-24922-9_3.

6 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, Saarbrücken, Germany (Virtual Conference), July 8-11, 2020*, volume 168 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.11.

7 Jan Bok, Avinandan Das, Anna Gujgiczer, and Nikola Jedlicková. Generalizing brooks' theorem via partial coloring is hard classically and locally. *CoRR*, abs/2508.16308, 2025. doi:10.48550/arXiv.2508.16308.

8 Daniel W. Cranston and Landon Rabern. Brooks' theorem and beyond. *J. Graph Theory*, 80(3):199–225, 2015. doi:10.1002/JGT.21847.

9 Avinandan Das, Pierre Fraigniaud, and Adi Rosén. Distributed partial coloring via gradual rounding. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, *27th International Conference on Principles of Distributed Systems, OPODIS 2023, Tokyo, Japan, December 6-8, 2023*, volume 286 of *LIPIcs*, pages 30:1–30:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.OPODIS.2023.30.

10 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

11 Magnús M. Halldórsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonoyan. Near-optimal distributed degree+1 coloring. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 450–463. ACM, 2022. doi:10.1145/3519935.3520023.

12 Ian Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.

13 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory Comput.*, 4(1):129–135, 2008. doi:10.4086/TOC.2008.V004A006.

14 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

15 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2 edition, 2017.

16 Moni Naor and Larry J. Stockmeyer. What can be computed locally? In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 184–193. ACM, 1993. doi:10.1145/167088.167149.

Online Hitting Set for Axis-Aligned Squares

Minati De
Department of Mathematics, Indian Institute of Technology Delhi, New Delhi, India

Satyam Singh
Department of Computer Science, Aalto University, Espoo, Finland

Csaba D. Tóth
Department of Mathematics, California State University Northridge, Los Angeles, CA, USA
Department of Computer Science, Tufts University, Medford, MA, USA

Abstract

Given a set P of n points in the plane and a sequence of axis-aligned squares that arrive in an online fashion, the online hitting set problem consists of maintaining, by adding new points from P if necessary, a hitting set $H \subseteq P$, which contains at least one point in every input square that has already arrived. We present an $O(\log n)$-competitive deterministic algorithm for this problem. The competitive ratio is the best possible, apart from constant factors. In fact, this is the first $O(\log n)$-competitive algorithm for the online hitting set problem that works for geometric objects of arbitrary sizes (i.e., unbounded scaling factors) in the plane. We further generalize this result to positive homothets of a polygon with $k \geq 3$ vertices in the plane and provide an $O(k^2 \log n)$-competitive algorithm.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Online algorithms

Keywords and phrases axis-aligned squares, hitting set, homothets of a polygon, online algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.16

Related Version *Previous Version*: https://arxiv.org/pdf/2510.23107

Funding *Satyam Singh*: Research on this paper was supported by the Research Council of Finland, Grant 363444.
Csaba D. Tóth: Research on this paper was supported, in part, by the NSF award DMS-2154347.

Acknowledgements We thank an anonymous reviewer for a suggestion that helped improve the competitive ratio in Theorem 1 from $O(\varrho \log n)$ to $O((1 + \log \varrho) \log n)$.

1 Introduction

The ***minimum hitting set*** problem is one of Karp's 21 classic NP-hard problems [24]. Given a set P of elements and a collection $\mathcal{C} = \{S_1, \ldots, S_m\}$ of subsets of P, referred to as ***ranges***, we need to find a set $H \subseteq P$ (***hitting set***) of minimal size such that every set $S_i \in \mathcal{C}$ contains at least one element of H. Motivated by numerous applications in VLSI design, resource allocation, and wireless networks, researchers have extensively studied the problem for geometric range spaces. In the ***geometric hitting set*** problem, we have $P \subseteq \mathbb{R}^d$ for some constant dimension d, and the sets in $\mathcal{C}$ are geometric objects of some type, such as balls, unit balls, simplices, hypercubes, or hyper-rectangles. Note that the minimum hitting set problem is dual to the minimum set cover problem in the abstract setting, but this duality in general does not extend to the geometric setting.

In this paper, we study the online hitting set problem for geometric objects. In the ***online geometric hitting set*** problem, the point set P is known in advance, while the objects in $\mathcal{C}$ arrive one at a time (without prior knowledge). We need to maintain a hitting set $H_i \subseteq P$ for the first i objects for all $i \geq 1$. Importantly, in the online model, points may be added to

© Minati De, Satyam Singh, and Csaba D. Tóth;
licensed under Creative Commons License CC-BY 4.0

the hitting set as new objects arrive, but they cannot be removed (i.e., $H_i \subseteq H_j$ for $i \leq j$). Upon the arrival of a new object $S_i \in \mathcal{C}$, any number of points can be added to the hitting set. Depending on whether P is finite [18, 20, 23, 26] or infinite [2, 14, 17, 19, 21, 22], there are different versions of the online geometric hitting set problem. In this paper, we consider P to be a finite set of points in $\mathbb{R}^2$.

The quality measure for online algorithms is the competitive ratio, which quantifies the loss of performance due to irrevocable decisions made based on incomplete information. Let ALG be an algorithm for the online hitting set problem on an instance $(P, \mathcal{C})$. The ***competitive ratio*** of ALG, denoted by $\rho(\mathsf{ALG})$, is the supremum, over all possible input sequences σ, of the ratio between the size $\mathsf{ALG}(\sigma)$ of the hitting set obtained by the online algorithm and the minimum size $\mathsf{OPT}(\sigma)$ of a hitting set for the same input:

$$\rho(\mathsf{ALG}) = \sup_{\sigma} \left[\frac{\mathsf{ALG}(\sigma)}{\mathsf{OPT}(\sigma)} \right].$$

1.1 Related Previous Work

Alon et al. [3] initiated the study of the online hitting set problem and presented a deterministic algorithm with a competitive ratio of $O(\log |P| \log |\mathcal{C}|)$ and obtained an almost matching lower bound of $\Omega\left(\frac{\log |P| \log |\mathcal{C}|}{\log \log |P| + \log \log |\mathcal{C}|}\right)$. While their work addresses the general setting, Even and Smorodinsky [23] initiated the study of the online geometric hitting set problem for various geometric objects. They established an optimal competitive ratio of $\Theta(\log |P|)$ when P is a finite subset of $\mathbb{R}$, and the objects are intervals in $\mathbb{R}$. They also established an optimal competitive ratio of $\Theta(\log |P|)$ when P is a finite subset of $\mathbb{R}^2$, and the objects are half-planes or congruent disks in the plane.

Later, Khan et al. [26] examined the problem for the special case of a finite set of ***integer*** points $P \subseteq [0, N)^2 \cap \mathbb{Z}^2$ and a collection $\mathcal{C}$ of axis-aligned squares $S \subseteq [0, N)^2$ with integer coordinates, for $N > 0$. They developed an $O(\log N)$-competitive deterministic algorithm for this variant. They also established a randomized lower bound of $\Omega(\log |P|)$, where $P \subset \mathbb{R}^2$ is finite and $\mathcal{C}$ consists of congruent axis-aligned squares. De et al. [18, 20] further investigated the problem for an arbitrary finite set $P \subset \mathbb{R}^2$, where $\mathcal{C}$ consists of homothetic geometric objects with scaling factors (e.g., diameters) in the interval $[1, M]$ for some parameter $M > 0$. In [18], they considered homothetic copies of a regular k-gon (for $k \geq 4$) and developed a randomized algorithm with expected competitive ratio $O(k^2 \log M \log |P|)$. Although regular k-gons can approximate disks as $k \to \infty$, this result does not imply a competitive algorithm for disks with radii in $[1, M]$. In [20], they addressed this gap by presenting an $O(\log M \log |P|)$-competitive deterministic algorithm for homothetic disks, and further generalized their result to positive homothets of any convex body in the plane with scaling factors in $[1, M]$.

However, it remained an open problem whether the dependence on the parameter M is necessary, or whether there is an $O(\log |P|)$-competitive online hitting set algorithm for arbitrary homothets of a polygon or for arbitrary disks in the plane. In fact, the full version of Khan et al. [26] posed this as an open problem specifically for axis-aligned squares.

1.2 Our Contribution

We present the first $O(\log n)$-competitive algorithm for the online hitting set problem for a set P of $n = |P|$ points and geometric objects of arbitrary sizes in the plane; Table 1 summarizes both previous and new results. Our algorithm works for axis-aligned squares of

arbitrary sizes, and it generalizes to axis-aligned rectangles of bounded aspect ratio. The ***aspect ratio*** of a rectangle is the ratio of the length of the longer to that of the shorter side (e.g., the aspect ratio of a square is 1, and the aspect ratio of a 1×2 or a 2×1 rectangle is 2).

Table 1 Summary of known and new results for the geometric online hitting set problem where $|P| = n$ for some $n \in \mathbb{N}$. (#) indicates lower bounds for randomized algorithms. The results presented in this paper are listed in the last three rows.

Finite Point Set	Objects	Lower Bound	Upper Bound
$P \subset \mathbb{R}$	Intervals in $\mathbb{R}$	$\Omega(\log \lvert P\rvert)$ [23]	$O(\log \lvert P\rvert)$ [23]
$P \subset \mathbb{R}^2$	Half-planes in $\mathbb{R}^2$	$\Omega(\log \lvert P\rvert)$ [23]	$O(\log \lvert P\rvert)$ [23]
$P \subset \mathbb{R}^2$	Congruent disks in $\mathbb{R}^2$	$\Omega(\log \lvert P\rvert)$ [23]	$O(\log \lvert P\rvert)$ [23]
$P \subseteq [0, N)^2 \cap \mathbb{Z}^2$	Axis-aligned squares with integral vertices	$\Omega(\log \lvert P\rvert)$ [26] (#)	$O(\log N)$ [26]
$P \subseteq [0, N)^2 \cap \mathbb{Z}^2$	Bottomless rectangles (of the form $[a, b] \times (-\infty, c]$)	$\Omega(\log \lvert P\rvert)$ [23]	$O(\log N)$ [20]
$P \subset \mathbb{R}^2$	Positive homothets of an arbitrary convex body in $\mathbb{R}^2$ with scaling factors in $[1, M]$	$\Omega(\log \lvert P\rvert)$ [26] (#)	$O(\log M \log \lvert P\rvert)$ [20]
$P \subset \mathbb{R}^2$	Axis-aligned squares	$\Omega(\log \lvert P\rvert)$ [26] (#)	$O(\log \lvert P\rvert)$ [Theorem 1]
$P \subset \mathbb{R}^2$	Axis-aligned rectangles of aspect ratio at most $\varrho \geq 1$	$\Omega(\log \lvert P\rvert)$ [26] (#)	$O((1 + \log \varrho) \log \lvert P\rvert)$ [Theorem 1]
$P \subset \mathbb{R}^2$	Positive homothets of a polygon with $k \geq 3$ vertices	$\Omega(\log \lvert P\rvert)$ [26] (#)	$O(k^2 \log \lvert P\rvert)$ [Theorem 2]

▶ **Theorem 1.** *For every $\varrho \geq 1$, there is an $O((1 + \log \varrho) \log n)$-competitive deterministic algorithm for the online hitting set problem for any set of n points in the plane and a sequence of axis-aligned rectangles of aspect ratio at most ϱ.*

We further generalize Theorem 1 to positive homothets of a polygon.

▶ **Theorem 2.** *Let M be a polygon with $k \geq 3$ vertices. Then there is an $O(k^2 \log n)$-competitive deterministic algorithm for the online hitting set problem for any set of n points in the plane, and a sequence of positive homothets of M.*

The previous best competitive ratio for these problems was $O(\log^2 n)$, by Alon et al. [3], which holds more generally for any collection of sets of polynomial size, including any collection of geometric objects of bounded VC-dimension. As noted above, De et al. [18] designed an $O(k^2 \log M \log n)$-competitive randomized algorithm for homothets of regular k-gons with scaling factors in the range $[1, M]$. Theorem 2 provides a deterministic algorithm for homothets of arbitrary convex k-gons of arbitrary sizes. However, it is unclear whether the dependence on k is necessary. Even and Smorodinsky [23] asked whether there is an online hitting set algorithm with an $O(\log n)$-competitive ratio for any set system on n points with bounded VC-dimension. Khan et al. [26] asked whether there is an $O(\log n)$-competitive algorithm for the online hitting set problem with squares, as their algorithm is restricted to integer points in $[0, N)^2 \cap \mathbb{Z}^2$ and it is $O(\log N)$-competitive even if $|P| \ll N$. Our

result (Theorem 1) gives an affirmative answer to their question. It is unclear whether $O(\log n)$-competitive online algorithms are possible for any other families of geometric set systems. In Section 6, we briefly discuss roadblocks to possible generalizations to disks (of arbitrary radii) in $\mathbb{R}^2$ or to cubes in $\mathbb{R}^3$, as well as possible extensions to the dual problem of online geometric set cover.

Technical highlights. We present the key ideas for axis-aligned squares (of arbitrary size), since all technical tools readily generalize to axis-aligned rectangles of bounded aspect ratio. Recall that the point set $P \subset \mathbb{R}^2$ is given in advance, that axis-aligned squares $\mathcal{C} = \{S_1, S_2 \ldots\}$ arrive one-by-one, and that we need to construct a hitting set $H \subset P$ of size $|H| \leq O(|\mathsf{OPT}| \log n)$ where $\mathsf{OPT} \subset P$ is an optimal hitting set for $\mathcal{C}$. Each square $S_i \in \mathcal{C}$ is hit by an unknown point $p \in \mathsf{OPT}$ in the offline optimum; we would like to add points to H that also hit many other potential squares that contain (the unknown) point p. Our general strategy is to preprocess P into a hierarchy of depth $O(\log n)$, and then use this hierarchy to narrow down the possible location of the points in OPT. In one dimension, where $P \subset \mathbb{R}$ and $\mathcal{C}$ is a sequence of intervals, a simple binary hierarchy is sufficient [23].

In the plane, we preprocess $P \subset \mathbb{R}^2$ with the classical BBD tree data structure by Arya et al. [5], which has found a wide range of applications in computational geometry, including binary space partitions, range searching, and many other problems [1, 4, 5, 11, 25]; this data structure was previously used for the online geometric set cover problem by Khan et al. [26]. For a set $P \subset \mathbb{R}^2$ of n points, the BBD tree is a hierarchical space partition of depth $O(\log n)$, where all cells of the partition are "fat" (in the sense defined below). When a square S_i arrives online, we choose hitting points via a top-down traversal of the BBD tree. Specifically, we "activate" the $O(1)$ lowest inactive cells of the BBD tree that intersect S_i (see Section 4 for details), and add $O(1)$ extremal points in each of these cells to H. The extremal points of a rectangular cell C are simply the leftmost, rightmost, topmost, and bottommost points in $P \cap C$ (i.e., extremal with respect to axis-parallel directions). Extremal points have previously been used in computational geometry for many different problems, including guarding set or shortest path problems [8, 10, 16, 30], as well as restricted versions of the online hitting set problem in combination with quadtrees [26].

Unfortunately, the cells in the BBD tree are not necessarily convex – they include rectangles with a rectangular hole – so we need to define extremal points much more carefully. The intersection $C_v \cap S_i$ of a cell C_v of the BBD tree and a square S_i can be reduced to axis-aligned half-spaces, and we show that $C_v \cap S_i \cap P$ is either empty or contains at least one extremal point of C_v (see Section 3.2 for details). Note that these techniques work for arbitrary axis-aligned fat rectangles (the fatness of a rectangle is quantified by the aspect ratio ϱ): extremal points in the four axis-parallel directions are sufficient if S_i is an axis-aligned rectangle, and the intersection $C_v \cap S_i$ reduces to halfplanes if both C_v and S_i are fat.

Each (unknown) point $p \in \mathsf{OPT}$ is contained in $O(\log n)$ cells of the BBD tree. After $O(\log n)$ steps, in which a square S_i arrives that contains the point $p \in \mathsf{OPT}$ but our algorithm adds $O(1)$ new points to H, the BBD tree is "saturated", in the sense that all cells containing p are activated – at this point, the extremal points in H already hit any subsequent square S_i that contains p. We prove that our algorithm uses only $O(\log n)$ points for any sequence of squares that can be optimally hit by a single point; see Section 4.2 for further details.

Relation to online set cover for squares. Consider the ***online geometric set cover*** problem: given a set $\mathcal{S}$ of n geometric objects in $\mathbb{R}^d$, and a sequence of points arriving one by one, we need to maintain a set cover $C_i \subset \mathcal{S}$ for the first i points in an online fashion.

Note, however, that the hitting set and the set cover problems are not equivalent in the geometric setting: points in the plane form a 2-parameter family (parameterized by x- and y-coordinates), while axis-aligned squares form a 3-parameter family (parameterized by, e.g., a lower-left corner and a side length). In the online hitting set problem, the 2-parameter family is known in advance, and the 3-parameter family arrives without prior knowledge – the roles are reversed in online set cover. That is, an online hitting set algorithm makes decisions based on less information (i.e., more uncertainty) than an analogous online set cover algorithm. Importantly, these problems are not equivalent, and techniques developed for one problem do not easily carry over to the other. Khan et al. [26] studied a restricted version of the online set cover problem for axis-aligned squares. In their model, both a collection $\mathcal{S}$ of m axis-aligned squares and a set $\tilde{P}$ of n points are given in advance. Points from $\tilde{P}$ arrive online, and we need to maintain a set of squares $C_i \subset \mathcal{S}$ that cover the first i points. Using a BBD tree decomposition over the point set $\tilde{P}$, they obtain a deterministic $O(\log n)$-competitive algorithm.

We remark that their result yields an $O(\log m)$-competitive algorithm for the ***unrestricted*** version of the online set cover problem for axis-aligned squares (where $\tilde{P}$ is not given in advance). An arrangement of m squares in the plane determines $O(m^2)$ cells: points in the same cell are contained in the same subset of squares (they are combinatorially indistinguishable). We can define $\tilde{P}$ as a set of $O(m^2)$ representative points, one from each cell. When a sequence of arbitrary points $p_1, p_2, \ldots$ arrive online, we can replace each point p_i with the representative of its cell, and invoke the algorithm by Khan et al. [26], with competitive ratio $O(\log |\tilde{P}|) = O(\log m^2) = O(\log m)$.

Organization. Section 2 begins by introducing the necessary definitions and then reviews a classical space partition data structure, the Balanced Box Decomposition Tree (BBD tree) [6]. Section 3 presents several key properties of BBD trees. Section 4 describes our online algorithm for hitting axis-aligned rectangles, and analyzes its competitive ratio. Next, Section 5 generalizes the main result from axis-aligned squares to positive homothets of an arbitrary polygon. Finally, Section 6 concludes with a discussion of future research directions.

2 Notation and Preliminaries

Unless stated otherwise, the term ***object*** refers to a compact set in $\mathbb{R}^d$ with a nonempty interior. Let σ denote such an object. For a scaling parameter $\lambda \in \mathbb{R}$ and a translation vector $b \in \mathbb{R}^d$, the set $\lambda\sigma + b = \{\lambda x + b : x \in \sigma\}$ is called a ***homothet*** or ***homothetic copy*** of σ; and it is a ***positive homothet*** if $\lambda > 0$.

BBD Trees. Arya et al. [6] introduced the ***Balanced Box Decomposition Tree*** (***BBD tree***, for short), which is a binary space partition tree for a set of n points in $\mathbb{R}^d$. Since its introduction in the 1990s, BBD trees have become a widely used data structure for processing and classifying spatial data in computational geometry and related fields. In contrast to the quadtree (or compressed quadtree), the depth of the BBD tree is $O(\log n)$, and the nodes correspond to "fat" regions; the precise definition is given below.

For a set P of n points in an axis-aligned square (the bounding box), the BBD tree is a binary tree T, where the nodes correspond to regions, called ***cells*** (with the root corresponding to the bounding box). The parent-child relation corresponds to containment between the corresponding cells with the following properties:

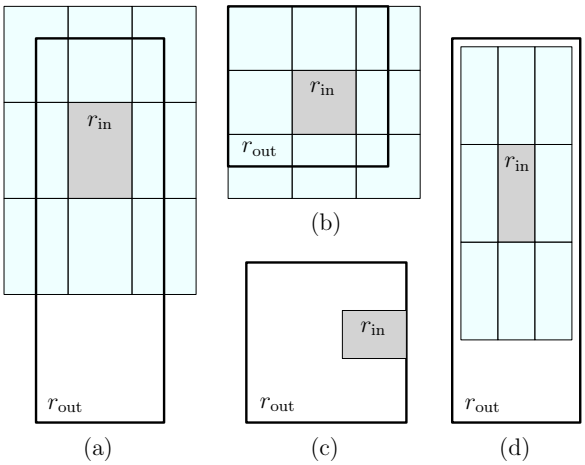


Figure 1 The rectangle r_{in} is not sticky for the rectangle r_{out} in (a) and (b), and is sticky in (c) and (d).

- Each node $v \in V(T)$ corresponds to a ***cell*** $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$, where $r_{\text{in}}(v)$ and $r_{\text{out}}(v)$ are axis-aligned rectangles such that $r_{\text{in}}(v) \subset r_{\text{out}}(v)$, and where possibly $r_{\text{in}}(v) = \emptyset$.
- the aspect ratio of $r_{\text{out}}(v)$ and $r_{\text{in}}(v)$ (if $r_{\text{in}} \neq \emptyset$) is at most 3;
- if $r_{\text{in}}(v) \neq \emptyset$, then it is ***sticky***, which means that the vertical (respectively, horizontal) distance between $r_{\text{in}}(v)$ and the boundary of $r_{\text{out}}(v)$ is either 0 or at least the side length of $r_{\text{in}}(v)$. An equivalent condition for stickiness can be obtained by considering the regular grid consisting of 3^2 translated copies of $r_{\text{in}}(v)$, centered around $r_{\text{in}}(v)$. The rectangle $r_{\text{in}}(v)$ is sticky for $r_{\text{out}}(v)$ if and only if every copy of $r_{\text{in}}(v)$ in this grid either lies entirely within $r_{\text{out}}(v)$ or is disjoint from the interior of $r_{\text{out}}(v)$, see Figure 1;
- the cells $\{C_v : v \in V(T)\}$ form a laminar set system, that is, if u is a descendant of v, then $C_u \subset C_v$, otherwise $\text{int}(C_u) \cap \text{int}(C_v) = \emptyset$;
- for each leaf node $v \in V(T)$, the region C_v contains at most one point of P.

Each internal node has exactly two children, generated by one of two operations: a ***fair split***, which decomposes a cell C_v along an axis-parallel line into two cells, or a ***shrink***, which introduces a new box R such that $r_{\text{in}}(v) \subseteq R \subseteq r_{\text{out}}(v)$ and decomposes $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ into $r_{\text{out}}(v) \setminus R$ and $R \setminus r_{\text{in}}(v)$. Furthermore, the number of nodes in T is $O(n)$, the depth of T is $O(\log n)$, and the entire structure can be constructed in $O(n \log n)$ time [6].

3 A Rectangle amid Cells of the BBD Tree

In this section, we present some important properties of BBD trees and discuss the relative position of an arbitrary axis-aligned rectangle of bounded aspect ratio with respect to the cells of the BBD tree. These properties play a crucial role in the design and analysis of our algorithms in Sections 4 and 5.

3.1 Crossing Between Rectangles and Cells of the BBD Tree

Let T be a BBD tree for a finite point set P. We say that an axis-aligned rectangle R ***crosses*** a cell C_v, $v \in V(T)$, if $C_v \cap R \neq \emptyset$ but C_v does not contain any vertex of R and R does not contain any vertex of C_v (i.e., vertices of $r_{\text{out}}(v)$ and vertices of $r_{\text{in}}(v)$ if any). See Figure 2 for examples.

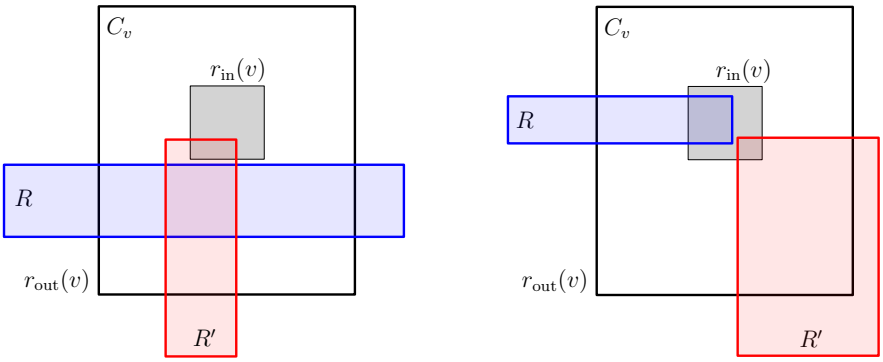


Figure 2 In both (a) and (b), the rectangle R crosses the cell C_v, but R' does not.

▶ **Lemma 3.** *If R is an axis-aligned rectangle of aspect ratio at most ϱ, for some $\varrho \geq 1$, then it crosses $O(\varrho)$ interior-disjoint cells of a BBD tree.*

Proof. Let T be a BBD tree, and let $U \subset V(T)$ be a set of nodes that corresponds to a family of interior-disjoint cells crossed by R. Let $v \in U$. Recall that a cell of the BBD tree is defined as $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$, where $r_{\text{in}}(v)$ may be empty. Depending on how R intersects with C_v, we distinguish between two cases.

Case 1: R crosses $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ and R intersects two opposite edges of $r_{\text{out}}(v)$. Assume w.l.o.g. that R intersects two vertical edges of $r_{\text{out}}(v)$ (i.e., R crosses $r_{\text{out}}(v)$ horizontally); see Figure 2(a). Since the aspect ratios of R and r_{out} are ϱ and at most 3, respectively, then we have

$$\text{width}(R) \leq \varrho \cdot \text{height}(R) < \varrho \cdot \text{height}(r_{\text{out}}(v)) \leq 3\varrho \cdot \text{width}(r_{\text{out}}(v)).$$

In particular, this implies $\frac{1}{3\varrho}\text{width}(R) < \text{width}(r_{\text{out}}(v))$. Note also that $\text{width}(R \cap r_{\text{out}}(v)) = \text{width}(r_{\text{out}}(v))$. If R horizontally crosses k interior-disjoint rectangles $r_{\text{out}}(v_1), \ldots, r_{\text{out}}(v_k)$, then

$$\frac{k}{3\varrho}\text{width}(R) < \sum_{i=1}^{k} \text{width}(r_{\text{out}}(v_i)) \leq \text{width}(R),$$

hence $k \leq 3\varrho$. That is, R horizontally crosses at most 3ϱ interior-disjoint cells. Similarly, R may cross at most 3ϱ cells vertically. However, if R crosses a cell C_v horizontally and another cell C_w vertically, then $\text{int}(C_v) \cap \text{int}(C_w) \neq \emptyset$. In particular, U cannot contain both v and w. Overall, R crosses at most 3ϱ interior-disjoint cells in this case.

Case 2: R crosses $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ and R intersects a pair of parallel edges in $r_{\text{out}}(v)$ and $r_{\text{in}}(v)$, respectively. Assume w.l.o.g. that R intersects the left side of both $r_{\text{in}}(v)$ and $r_{\text{out}}(v)$; see Figure 2(b). Let $\text{dist}_{\text{left}}(v)$ denote the distance between the left sides of r_{in} and r_{out}. Then we have $\text{width}(R \cap C_v) = \text{dist}_{\text{left}}(v)$, and in particular $\text{dist}_{\text{left}}(v) < \text{width}(R)$. Due to the stickiness, we also have $\text{width}(r_{\text{in}}) \leq \text{dist}_{\text{left}}(v)$. Overall, we obtain

$$\text{width}(R) \leq \varrho \cdot \text{height}(R) < \varrho \cdot \text{height}(r_{\text{in}}(v)) \leq 3\varrho \cdot \text{width}(r_{\text{in}}(v)) \leq 3\varrho \cdot \text{dist}_{\text{left}}(v).$$

By the pigeonhole principle, R crosses at most 3ϱ interior-disjoint cells between the left sides of r_{in} and r_{out}. Similarly, R crosses at most 3ϱ interior-disjoint cells between the right (respectively, top, bottom) sides of r_{in} and r_{out}. As a result, R crosses $O(\varrho)$ interior-disjoint cells in this case. ◀

3.2 Extremal Points and their Properties

For each node v of the BBD tree, we define a set Ext_v consisting of a constant number of ***extremal points*** in P.

- For an axis-aligned rectangle r, let $\text{ext}(r)$ be a subset of $P \cap r$ that consists of a point with the minimum x-coordinate, maximum x-coordinate, minimum y-coordinate, and maximum y-coordinate (ties are broken arbitrarily).
- If $r_{\text{in}}(v) = \emptyset$ (that is, $C_v = r_{\text{out}}(v)$), then let $\text{Ext}_v = \text{ext}(r_{\text{out}}(v))$; see Figure 3(a).
- If $r_{\text{in}}(v) \neq \emptyset$, then we subdivide $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ along the lines spanned by the four sides of $r_{\text{in}}(v)$ into k, $2 \leq k \leq 9$, rectangular regions $C_v = \bigcup_{i=1}^{k} r_i$ and let $\text{Ext}_v = \bigcup_{i=1}^{k} \text{ext}(r_i)$; see Figure 3(b).

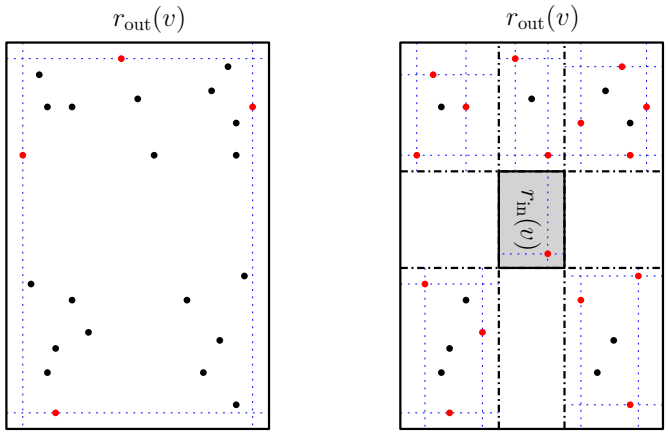


Figure 3 The extremal points Ext_v of the cell C_v are red, when (a) $r_{\text{in}}(v) = \emptyset$; (b) $r_{\text{in}}(v) \neq \emptyset$.

Properties of extremal points. We state a few properties of extremal points that will be used in the competitive analysis of our online algorithm in Section 4. Let P be a finite set of points in a bounding box (a square), and T be a BBD tree for P.

▶ **Lemma 4.** *Let r be an axis-aligned rectangle, and L^- be a half-plane bounded by an axis-parallel line L. If $P \cap r \cap L^- \neq \emptyset$, then $\text{ext}(r) \cap L^- \neq \emptyset$.*

Proof. Assume w.l.o.g. that L is the vertical line $x = a$, for some $a \in \mathbb{R}$, and let $L^- = \{(x, y) \in \mathbb{R}^2 : x \leq a\}$ be the left half-plane; see Figure 4(a). We prove the contrapositive. If $\text{ext}(r) \cap L^- = \emptyset$, then every point $p = (p_x, p_y) \in \text{ext}(P)$ satisfies $p_x > a$. In particular, we have $p_x > a$ for the leftmost point in $P \cap r$, hence we have $p_x > a$ for all points in $P \cap r$. This implies that $P \cap r \cap L^- = \emptyset$, as required. ◀

▶ **Lemma 5.** *Let $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ be a cell (where $r_{\text{in}}(v)$ may be empty), and L^- be a half-plane bounded by an axis-parallel line L. If $P \cap C_v \cap L^- \neq \emptyset$, then $\text{Ext}_v \cap L^- \neq \emptyset$.*

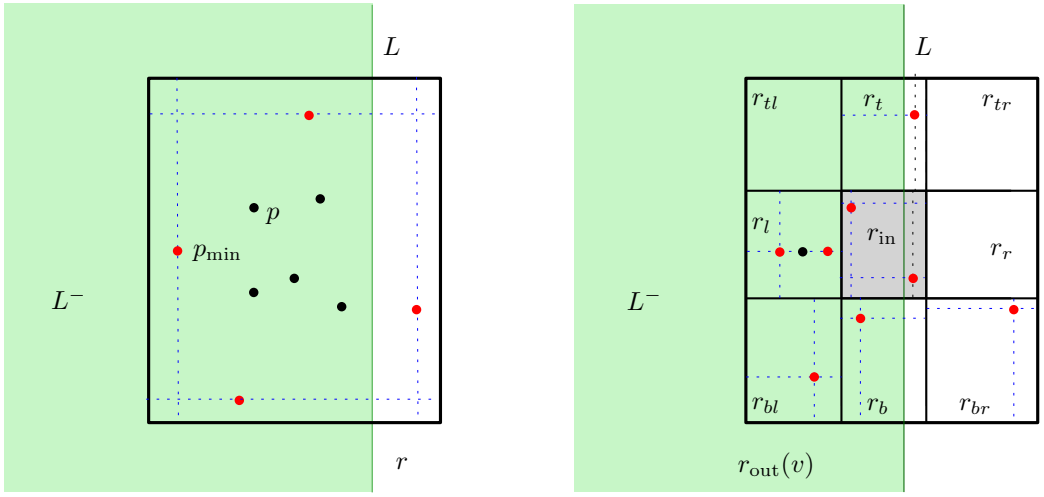


Figure 4 Illustration for (a) Lemma 4, and (b) Lemma 5.

Proof. Assume w.l.o.g. that L is the vertical line $x = a$ for some $a \in \mathbb{R}$, and let $L^- = \{(x, y) \in \mathbb{R}^2 : x \leq a\}$; see Figure 4(b) for an illustration. We prove the contrapositive. Suppose that $\text{Ext}_v \cap L^- = \emptyset$. Then every point of Ext_v has x-coordinate larger than a. Recall that $C_v = \bigcup_{i=1}^{k} r_i$. The leftmost point of $P \cap C_v$ is the leftmost point of $P \cap r_i$ for some $1 \leq i \leq k$. This point is included in $\text{ext}(r_i)$, and hence in $\text{Ext}_v = \bigcup_{i=1}^{k} \text{ext}(r_i)$. Consequently, Ext_v contains the leftmost point of $P \cap C_v$. Then, all points of $P \cap C_v$ have x-coordinate larger than a. Hence, we have $P \cap C_v \cap L^- = \emptyset$, as required. ◀

▶ **Lemma 6.** *Let $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ be a cell, and $S = L_1^- \cap L_2^+$ denote the intersection of two half-planes bounded by two horizontal lines or two vertical lines L_1 and L_2. If $P \cap C_v \cap S \neq \emptyset$ and S contains a corner of $r_{\text{in}}(v)$, then $\text{Ext}_v \cap S \neq \emptyset$.*

Proof. Assume w.l.o.g. that L_1 and L_2 are the vertical lines $x = a$ and $x = b$, respectively, for some $a, b \in \mathbb{R}$; see Figure 5 for an illustration. Let $L_1^- = \{(x, y) \in \mathbb{R}^2 : x \leq a\}$ be the left half-plane. If L_2^+ is also a left half-plane, then $L_1^- \cap L_2^+$ is a half-plane and Lemma 5 completes the proof. So we may assume that $L_2^+ = \{(x, y) \in \mathbb{R}^2 : x \geq b\}$ is the right half-plane, and $b < a$ (since $S = L_1^- \cap L_2^+ \neq \emptyset$).

Assume that S contains a corner $c = (c_1, c_2)$ of $r_{\text{in}}(v)$. Recall that C_v is subdivided along the lines spanned by the sides of $r_{\text{in}}(v)$. One of the subdivision lines is $x = c_1$, where $b < c_1 < a$ (since $c \in S$). Consequently, every subrectangle of C_v intersects at most one of L_1 and L_2.

Since $P \cap C_v \cap S \neq \emptyset$, there exists a point $p \in P \cap C_v \cap S$. Assume that $p \in r_x$, where r_x is one of the subrectangles of C_v. Since $\text{ext}(r_x) \cap S \subseteq \text{Ext}_v \cap S$ and by Lemma 4, we have $\text{ext}(r_x) \cap S \neq \emptyset$, thus we have $\text{Ext}_v \cap S \neq \emptyset$. Hence, the lemma follows. ◀

▶ **Lemma 7.** *Let $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$ be a cell, and let $S = L_1^- \cap L_2^+$ be the intersection of two half-planes bounded by two perpendicular axis-parallel lines L_1 and L_2. If $P \cap C_v \cap S \neq \emptyset$ and $L_1 \cap L_2 \in r_{\text{in}}(v)$, then $\text{Ext}_v \cap S \neq \emptyset$.*

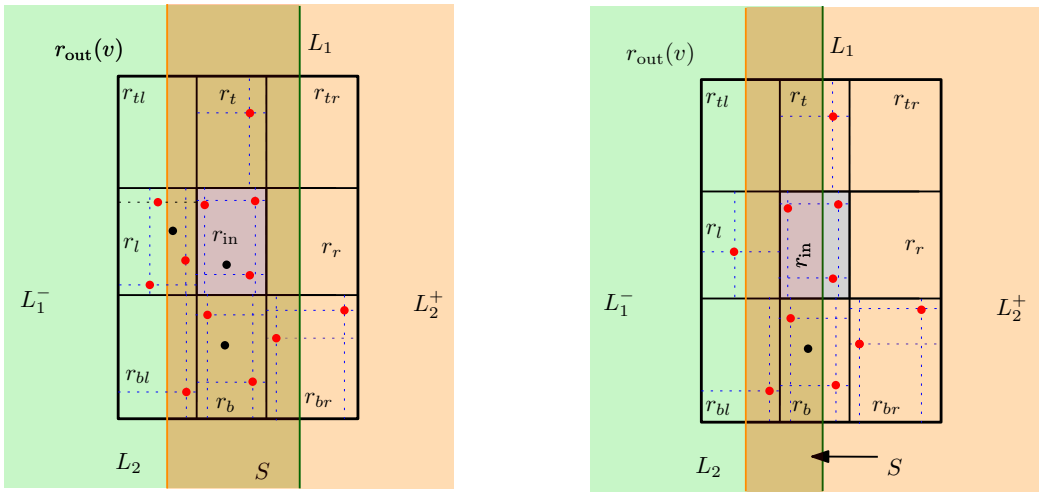


Figure 5 Illustration for Lemma 6.

Proof. Assume w.l.o.g. that L_1 is the vertical line $x = a$, and L_2 is the horizontal line $y = b$ for some $a, b \in \mathbb{R}$; see Figure 6 for an illustration. Let $L_1^- = \{(x, y) \in \mathbb{R}^2 : x \leq a\}$ be the left half-plane and $L_2^+ = \{(x, y) \in \mathbb{R}^2 : y \geq b\}$ be the top half-plane; see Figure 6. Since $L_1 \cap L_2 = (a, b) \in r_{\text{in}}(v)$, then L_1 intersects the subrectangles of C_v above and below r_{in}(v); and L_2 intersects the subrectangles of C_v to the left and right of $r_{\text{in}}(v)$. In particular, none of the subrectangles in C_v intersects both L_1 and L_2.

Since $P \cap C_v \cap S \neq \emptyset$, there exists a point $p \in P \cap C_v \cap S$. Assume that $p \in r_x$, where r_x is one of the subrectangles of C_v. Since r_x intersects at most one of L_1 and L_2, then $r_x \cap S$ equals $r_x \cap L_1^-$ or $r_x \cap L_2^+$. In both cases, by Lemma 4, we have $\text{ext}(r_x) \cap S \neq \emptyset$. Now $\text{ext}(r_x) \subseteq \text{Ext}_v$ implies that $\text{Ext}_v \cap S \neq \emptyset$, as required. ◀

4 Online Algorithm and Competitive Analysis

In this section, we first prove Theorem 8. In Section 4.1, we present our online hitting set algorithm for axis-aligned rectangles of aspect ratio ϱ, for constant $\varrho \geq 1$. In Section 4.2, we analyze its competitive ratio, and then use this algorithm as a subroutine to prove Theorem 1.

▶ **Theorem 8.** *For every $\varrho \geq 1$, there is an $O(\varrho \log n)$-competitive deterministic algorithm for the online hitting set problem for any set of n points in the plane and a sequence of axis-aligned rectangles of aspect ratio at most ϱ.*

4.1 Online Algorithm

We now describe our online algorithm. Given a set P of n points in the plane, we compute a BBD tree T for P, and the set H_v of extremal points $\text{Ext}_v \subset P$ for all nodes $v \in V(T)$. As the adversary presents a sequence of axis-aligned rectangles $\{S_1, \ldots, S_m\}$ of aspect ratio at most ϱ, we maintain the following data structures:

A hitting set $H_i \subset P$ for the first i rectangles $\{S_j : j \leq i\}$, which is initialized to $H_0 = \emptyset$. A set $A_i \subset V(T)$ of ***active*** nodes of T with the following property: If a node $v \in V(T)$ is active, then all ancestors of v are also active. Initially, all nodes are ***inactive*** (i.e., $A_0 = \emptyset$);

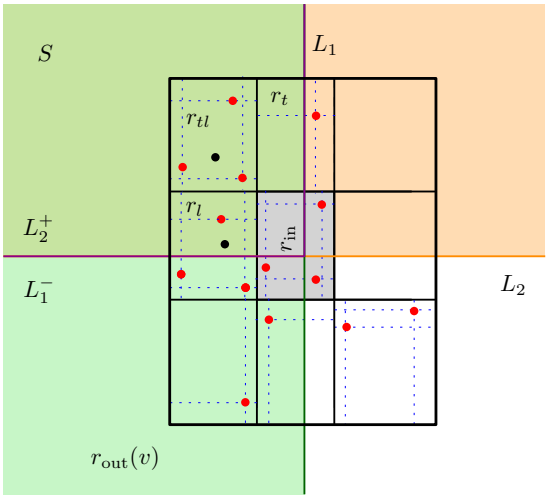


Figure 6 Illustration for Lemma 7.

inactive nodes can become active, and any active node remains active for the remainder of the algorithm. Furthermore, we maintain the property that $\bigcup_{v \in A_i} \text{Ext}_v \subset H_i$, i.e., H_i contains the extremal points of all active nodes (however, H_i may contain additional points as well).

When a rectangle S_i arrives, initialize $H_i := H_{i-1}$ and $A_i := A_{i-1}$. If $S_i \cap H_i \neq \emptyset$, then no further changes are needed in our data structures. Suppose that $S_i \cap H_i = \emptyset$.

1. Consider the four corners of S_i.
   - **a.** For each corner $a \in C_{\text{root}}$, find the highest inactive node $v \in V(T)$ such that $a \in C_v$: activate v and its sibling v' (set $A_i := A_i \cup \{v, v'\}$) and add their extremal points to H_i (set $H_i := H_i \cup \text{Ext}_v \cup \text{Ext}_{v'}$).
   - **b.** For each corner $a \notin C_{\text{root}}$, if the root of T is not already active, then activate the root and add its extremal points to H_i.
2. For every node $u \in V(T)$ where S_i crosses C_u,
   - **a.** If u is already active but its children are not, then activate both children of u and add their extremal points to H_i.
   - **b.** If u is inactive, then find the highest inactive ancestor v of u (possibly $v = u$): activate v and its sibling v' and add their extremal points to H_i.
3. If $S_i \cap H_i = \emptyset$ still holds, then add an arbitrary point in $S_i \cap P$ to H_i.

4.2 Competitive Analysis

The algorithm guarantees that H_i is a hitting set for the first i objects $\{S_1, \ldots, S_i\}$. It is also clear that for each new object S_i, we add $O(\varrho)$ new points to H_i: Since S_i has four corners and crosses $O(\varrho)$ cells of the BBD tree by Lemma 3.

Let $\mathsf{OPT} \subset P$ be an offline optimum, i.e., a minimum hitting set for $\mathcal{C} = \{S_1, \ldots, S_m\}$. For each point $p \in \mathsf{OPT}$, let $\mathcal{C}_p = \{S_j \in \mathcal{C} : p \in S_j\}$ be the set of objects hit by p. It is sufficient to show that for every $p \in \mathsf{OPT}$, the algorithm adds new points to the hitting set in $O(\log n)$ steps to hit objects in $\mathcal{C}_p$ (cf. Corollary 10). Since $O(\varrho)$ points are added to the hitting set in each such step, then the algorithm adds $O(\varrho \log n)$ points in response to the objects in $\mathcal{C}_p$, and hence $O(|\mathsf{OPT}| \cdot \varrho \log n)$ points in response to all objects in $\mathcal{C}$.

▶ **Lemma 9.** *If $S_i \in \mathcal{C}_p$ and $S_i \cap H_{i-1} = \emptyset$, then in step i, the algorithm activates a cell of the BBD tree containing p.*

Proof. The algorithm activates the root in step $i = 1$. In the remainder of the proof, we may assume that $i > 1$ and the root is already active. Before the arrival of S_i, we have $p \notin H_{i-1}$, so the leaf node of the BBD tree that contains p is inactive. Let v be the lowest active node in the BBD tree such that C_v contains p. Let u and w be the two children of v such that $p \in C_u$ (hence $p \notin C_w$).

We need to show that the algorithm activates u in step i. Suppose, for the sake of contradiction, that u is inactive at the beginning of step i. Then its sibling w is also inactive at that time (since our algorithm always activates two siblings).

The algorithm activates the highest inactive nodes that contain any of the four corners of S_i (and their siblings), as well as the highest inactive nodes corresponding to every cell crossed by S_i (and their siblings). Therefore, we may assume that neither C_u nor C_w contains any corner of S_i, and neither of them is crossed by S_i. Since $C_v = C_u \cup C_w$, then C_v does not contain any corner of S_i, either. If S_i crosses C_v, then at the beginning of step i, node v is active but its children u and w are inactive, and so the algorithm would activate both u and w in step i. For this reason, we may also assume that S_i does not cross C_v.

We examine all possible positions of C_v and C_u relative to S_i. Recall that $C_v = r_{\text{out}}(v) \setminus r_{\text{in}}(v)$, where $r_{\text{in}}(v)$ may be empty.

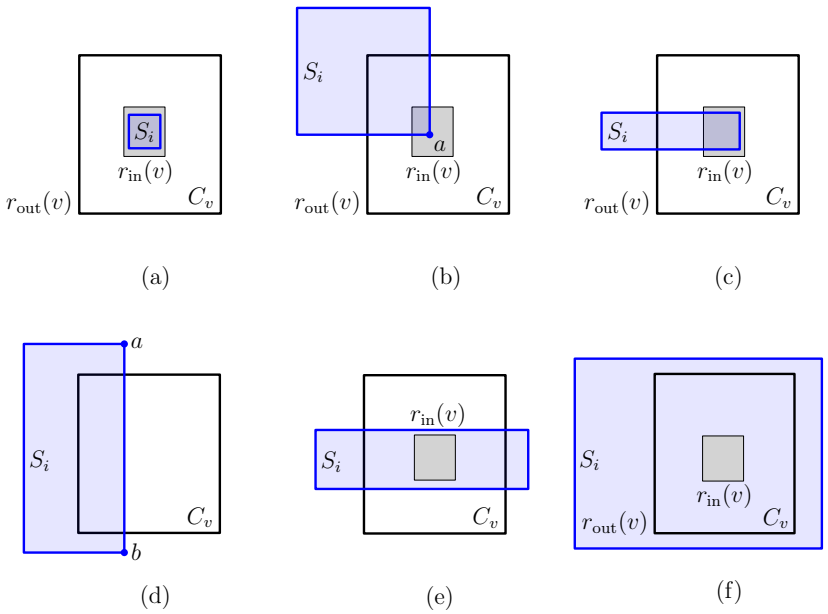


Figure 7 Illustration of (a) Case 1; (b) Case 2a; (c) Case 2b; (d) Case 3a; (e) Case 3b; and (f) Case 4.

Case 1: $S_i \subset r_{\text{out}}(v)$; see Figure 7(a). In this case, all four corners of S_i are in $r_{\text{out}}(v)$. If all four corners of S_i are in $r_{\text{in}}(v)$, then $p \in S_i \subset r_{\text{in}}(v)$, which contradicts the assumption that $p \in C_v$. Therefore, a corner of S_i lies in $r_{\text{out}}(v) \setminus r_{\text{in}}(v) = C_v$. For this corner of S_i, the highest inactive node is u or w. Consequently, the algorithm activates both siblings u and w. In particular, u is activated.

Case 2: $S_i \not\subset r_{\text{out}}(v)$ but $r_{\text{out}}(v)$ contains some corner of S_i; see Figure 7(b-c). Since C_v does not contain any corners of S_i, then all corners of S_i in $r_{\text{out}}(v)$ are in $r_{\text{in}}(v)$. Since $S_i \not\subset r_{\text{out}}(v)$, then $r_{\text{in}}(v)$ contains either one or two corners of S_i. We examine each case separately:

Case 2a: $r_{\text{in}}(v)$ contains precisely one corner of S_i. Assume w.l.o.g. that $r_{\text{in}}(v)$ contains the lower-right corner of S_i (as in Figure 7(b)). Denote by a the lower-right corner of S_i. Then S_i also contains precisely one corner of $r_{\text{in}}(v)$, namely its upper-left corner. Since the remaining three corners of S_i are outside of both $r_{\text{in}}(v)$ and C_v, they are outside of $r_{\text{out}}(v)$. Consequently, S_i also contains precisely one corner of $r_{\text{out}}(v)$, namely the upper-left corner of $r_{\text{out}}(v)$. Since $p \in S_i \cap C_v$, then Lemma 7 yields $\text{Ext}_v \cap S_i \neq \emptyset$. However, C_v was activated in a previous step. This implies that $\text{Ext}_v \subset H_{i-1}$, hence $S_i \cap H_{i-1} \neq \emptyset$: a contradiction.

Case 2b: $r_{\text{in}}(v)$ contains exactly two corners of S_i. Recall that an axis-aligned rectangle S_i crosses a cell C_v, for $v \in V(T)$ if $C_v \cap S_i \neq \emptyset$ but C_v does not contain any vertex of S_i and S_i does not contain any vertex of C_v. Consequently, S_i crosses C_v: a contradiction.

Case 3: $r_{\text{out}}(v)$ does not contain any corner of S_i, but it intersects some edges of S_i; see Figure 7(d-e). Let ab be an edge of S_i that intersects C_v, where a and b are corners of S_i. Since $r_{\text{out}}(v)$ contains neither a nor b, then both a and b are outside of $r_{\text{out}}(v)$, and so the two edges of S_i orthogonal to ab are also outside of $r_{\text{out}}(v)$. Consequently, $r_{\text{out}}(v)$ intersects one edge of S_i or two parallel edges of S_i.

Case 3a: $r_{\text{out}}(v)$ intersects precisely one edge of S_i. Assume w.l.o.g. that $r_{\text{out}}(v)$ intersects the right edge ab of S_i; see Figure 7(d). Let L be the vertical line spanned by ab, and let L^- be the left half-plane determined by L. Note that $p \in S_i \cap C_v$ implies $p \in L^-$, so $P \cap C_v \cap L^- \neq \emptyset$. Lemma 5 yields $\text{Ext}_v \cap S_i \neq \emptyset$. However, C_v was activated in a previous step. This implies that $\text{Ext}_v \subset H_{i-1}$, hence $S_i \cap H_{i-1} \neq \emptyset$: a contradiction.

Case 3b: $r_{\text{out}}(v)$ intersects two parallel edges of S_i. Assume w.l.o.g. that $r_{\text{out}}(v)$ intersects the left and right edges of S_i. If S_i does not contain any corners of $r_{\text{in}}(v)$, then S_i does not contain any vertex of C_v, and so S_i crosses C_v: a contradiction.

So we may assume that S_i contains some corners of $r_{\text{in}}(v)$. Lemma 6 yields $\text{Ext}_v \cap S_i \neq \emptyset$. However, C_v was activated in a previous step. This implies that $\text{Ext}_v \subset H_{i-1}$, hence $S_i \cap H_{i-1} \neq \emptyset$: a contradiction.

Case 4: $r_{\text{out}}(v)$ lies in the interior of S_i; see Figure 7(f). Since $p \in P \cap C_v$, then $P \cap C_v \neq \emptyset$, and so $\text{Ext}_v \neq \emptyset$. Since C_v is active, then $\text{Ext}_v \subset H_{i-1}$, and $C_v \subset r_{\text{out}}(v) \subset S_i$ implies that $\text{Ext}_v \subset S_i$. Consequently, $H_{i-1} \cap S_i \neq \emptyset$: a contradiction. ◀

▶ **Corollary 10.** *For every $p \in OPT$, the algorithm adds new points to the hitting set in $O(\log n)$ steps in which rectangles in $\mathcal{C}_p$ arrive.*

Proof. Let $\widehat{\mathcal{C}}_p \subseteq \mathcal{C}_p$ be the set of rectangles $S_i \in \mathcal{C}_p$ for which the algorithm adds new hitting points to H_i (that is, any rectangle $S_i \in \mathcal{C}_p \setminus \widehat{\mathcal{C}}_p$ contains p, but does not require new hitting points as $S_i \cap H_{i-1} \neq \emptyset$). By Lemma 9, the algorithm activates a cell containing p to hit each rectangle in $\widehat{\mathcal{C}}_p$. In the BBD tree, the cells containing p correspond to a descending path from root to leaf. Since the depth of the BBD tree for n points is $O(\log n)$, there are $O(\log n)$ such cells, which implies that $|\widehat{\mathcal{C}}_p| \leq O(\log n)$, as required. ◀

This completes the proof of Theorem 8. Now we have all the tools to prove Theorem 1.

Proof of Theorem 1. Theorem 8 implies the claim if $1 \leq \varrho \leq 2$. Assume that $\varrho > 2$. Partition the incoming rectangles into $O(1 + \log \varrho)$ classes: let $\mathcal{R}_0$ be the set of rectangles of aspect ratio in $[1, 2]$, and for $j = 1, \ldots, \lceil \log_2 \varrho \rceil$, let $\mathcal{R}_j$ be the set of rectangles of aspect ratio in $(2^j, 2^{j+1}]$. Within each class, we further separate rectangles whose width is at least their height from those whose height exceeds their width, yielding $2(1 + \lceil \log_2 \varrho \rceil)$ classes in total. Applying the linear transformation $(x, y) \mapsto (x/2^j, y)$ (respectively, $(x, y) \mapsto (x, y/2^j)$) to both P and the rectangles in the j-th class maps each rectangle to one with aspect ratio in $[1, 2]$. Since the scaling is applied to both P and the rectangles, any hitting set for the original instance corresponds to a hitting set of the same size in the scaled instance. Furthermore, the size of an offline optimum for each class is at most $|\mathrm{OPT}|$, the size of the offline hitting set of all rectangles. Running a separate instance of our online algorithm with $\varrho = 2$ on each of the $O(\log \varrho)$ classes yields a hitting set of size $O(|\mathrm{OPT}| \cdot \log n)$ per class. Taking the union of the resulting hitting sets over all classes is a hitting set of size $O(|\mathrm{OPT}| \cdot (1 + \log \varrho) \log n)$. ◀

4.3 Runtime Analysis

It is not difficult to implement our online algorithm (Section 4.1) for n points in the plane with $O(n \log^2 n)$ preprocessing time and $O(\log n)$ update time. It is known that the BBD tree can be computed in $O(n \log n)$ time [6]. Recall that the BBD tree has depth $O(\log n)$, and the cells on each level are interior-disjoint. Consequently, the extremal points on each level can be computed using four sweep-line algorithms (one for each axis-aligned direction) in $O(n \log n)$ time; and the extremal points of all cells can be computed in $O(n \log^2 n)$ time.

We need several data structures to support our online algorithm. First, we need an orthogonal range searching data structure for P that reports a point in $P \cap S_i$ for a query rectangle S_i. The classical data structure by Chazelle [15] (see also [7, Chap. 5]) already achieves $O(n \log n)$ preprocessing and $O(\log n)$ query time. Second, we need a semi-dynamic orthogonal range emptiness data structure for H_i that supports insertions and emptiness queries $H_i \cap S_i = \emptyset$. The semi-dynamic data structure by Mehlhorn and Näher [27, Theorem 8] supports $O(\log n)$ insertion and query time. (We note that $o(\log n)$ amortized update and query time is possible in the word RAM model [12, 13, 28], i.e., in the rank space of the point set P, which distorts the aspect ratios of orthogonal ranges). Finally, we also augment the BBD tree with binary variables to indicate which cells are active.

When a rectangle S_i arrives, we can test whether $H_i \cap S_i = \emptyset$ in $O(\log n)$ time using the semi-dynamic range emptiness data structure for H_i. If S_i requires new hitting points, we use the BBD tree to find the highest inactive cells containing the four corners of S_i, as well as the highest inactive cells that cross S_i, in $O(\log n)$ time. We can then activate these cells and their siblings, and add $O(1)$ extremal points to the hitting set in $O(1)$ time. If S_i does not contain any of the new extremal points added H_i, we can find a suitable hitting point in $P \cap S_i$ in $O(\log n)$ time using the orthogonal range reporting data structure for P. Finally, updating the semi-dynamic range emptiness data structure for H_i takes $O(\log n)$ time.

In the proof of Theorem 1, we run $O(1 + \log \varrho)$ parallel instances of this online algorithm, each acting on distinct classes of rectangles. So the preprocessing time increases to $O(n \log^2 n(1 + \log \varrho))$ but the update time remains $O(\log n)$.

5 Generalizations to Positive Homothets of Polygons

In Section 4, we presented an $O(\log n)$-competitive algorithm for the online hitting set problem with n points in the plane and axis-aligned squares (of aspect ratio $\varrho = 1$). Axis-aligned squares are homothets of a unit square. Since a suitable linear transformation maps any parallelogram to a unit square, our result extends to positive homothets of any parallelogram.

▶ **Corollary 11.** *For every parallelogram M in the plane, there is an $O(\log n)$-competitive algorithm for the online hitting set problem for any set of n points in the plane and a sequence of positive homothets of M in the plane.*

We further generalize Theorem 1 to positive homothets of polygons with k vertices.

▶ **Lemma 12.** *Every polygon with $k \geq 3$ vertices is the union of at most $5k-12$ parallelograms.*

Proof. Every simple polygon M with $k \geq 3$ vertices admits a triangulation with $k-2$ triangles. In general, a polygon M with $k \geq 3$ vertices and $h \geq 0$ holes admits a triangulation with $k+2h-2$ triangles [29, Lemma 5.2]. We have $h \leq k/3-1$, as every hole as well as the outer boundary has at least 3 vertices. Therefore, M is the union of at most $5k/3-4$ triangles. Every triangle T is decomposed by its three medians into four congruent subtriangles: one containing the center of T, and three incident to each of the corners of T. The union of the central subtriangle and a corner subtriangle is a parallelogram; see Figure 8. Thus, T is the union of three parallelograms, and consequently M is the union of at most $3(5k/3-4) = 5k-12$ parallelograms. ◀

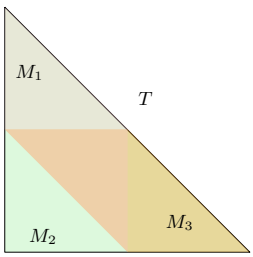


Figure 8 A triangle is the union of three parallelograms.

▶ **Lemma 13.** *Let T be a polygon that can be written as a union of k parallelograms. Then there is an $O(k^2 \log n)$-competitive deterministic algorithm for the online hitting set problem for any set P of n points and a sequence of positive homothets of T in the plane.*

Proof. Assume that T is the union of k parallelograms, i.e., $T = \bigcup_{j=1}^{k} M_j$, where each M_j is a parallelogram. By Corollary 11, there is an $O(\log n)$-competitive deterministic algorithm ALG_j for the online hitting set problem for the same point set P and a sequence of positive homothets of the parallelogram M_j, for every $j \in \{1, \ldots, k\}$.

Online algorithm. We now describe a deterministic online algorithm for the point set P and a sequence $T_1, T_2, \ldots$ of positive homothets of T. For every $i \in \mathbb{N}$, we have $T_i = a_i T + b_i$ for some scaling factor $a_i > 0$ and translation vector $b_i \in \mathbb{R}^2$. Since $T = \bigcup_{j=1}^{k} M_j$, then $T_i = \bigcup_{j=1}^{k} M_{i,j}$, where $M_{i,j} = a_i M_j + b_i$ is a positive homothet of the parallelogram M_j.

We maintain a hitting set $H_i \subset P$ for the first i homothets $\{T_\ell : \ell \leq i\}$. We initialize the algorithm ALG_j for $j = 1, \ldots, k$, that each maintain a hitting set $H_{i,j} \subset P$ for some ***subset*** of the first i parallelograms $\{M_{\ell,j} : \ell \leq i\}$. We also maintain the set $H_i = \bigcup_{j=1}^{k} H_{i,j}$.

When a homothet $T_i = a_i T + b_i$ arrives, we initialize $H_i := H_{i-1}$, and $H_{i,j} := H_{i-1,j}$ for all $j = 1, \ldots, k$. If $T_i \cap H_i \neq \emptyset$, then no further changes are needed. Otherwise, we compute the parallelograms $M_{i,j} = a_i M_j + b_i$ for $j = 1, \ldots, k$. For each $j = 1, \ldots, k$, if $P \cap M_{i,j} \neq \emptyset$, then we feed $M_{i,j}$ to the algorithm ALG_j, which in turn adds new points to $H_{i,j}$. Finally, we update H_i by setting $H_i := \bigcup_{j=1}^{k} H_{i,j}$. This completes the description of the algorithm.

Competitive analysis. The algorithm guarantees that H_i is a hitting set for the first i objects $\{T_1, \ldots, T_i\}$. It is also clear that for each new object T_i, we add $O(k)$ new points to H_i: since each algorithm ALG_j adds $O(1)$ points to $H_{i,j}$.

Let $\mathsf{OPT} \subset P$ be an offline optimum, i.e., a minimum hitting set for $\mathcal{C} = \{T_1, \ldots, T_m\}$. For each point $p \in \mathsf{OPT}$, let $\mathcal{C}_p = \{T_i \in \mathcal{C} : p \in T_i\}$ be the set of objects hit by p. It is sufficient to show that for every $p \in \mathsf{OPT}$, the algorithm adds $O(k^2 \log n)$ points to H_i in response to the arrival of the objects in $\mathcal{C}_p$.

Let $\mathcal{C}'_p$ be a set of objects $T_i \in \mathcal{C}_p$ such that our algorithm adds new points to the hitting set in step i. Since our algorithm adds $O(k)$ points to the hitting set in each step, it is enough to show that $|\mathcal{C}'_p| \leq O(k \log n)$. We further partition $\mathcal{C}'_p$ based on which parallelograms are hit by point p. For $j = 1, \ldots, k$, let $\mathcal{C}'_{p,j} = \{T_i \in \mathcal{C}'_p : p \in M_{i,j}$, and $p \notin M_{i,j'}$ for $j' < j\}$; and $\mathcal{M}'_{p,j} = \{M_{i,j} : T_i \in \mathcal{C}'_{p,j}\}$. By definition, $\mathcal{C}'_p = \bigcup_{j=1}^{k} \mathcal{C}'_{p,j}$. Note that $\mathcal{M}'_{p,i}$ is a set of homothets of the parallelogram M_j that contain the point $p \in P$. By assumption, algorithm ALG_j is $O(\log n)$-competitive, and therefore adds $O(\log n)$ points to the hitting set in response to parallelograms in $\mathcal{M}'_{p,j}$. Recall that we feed a parallelogram $M_{i,j} \in \mathcal{M}'_{p,j}$ to ALG_j only if object T_i has not been hit by H_{i-1}, and hence ALG_j adds at least one new point to the hitting set in step i. Consequently, we have $|\mathcal{M}'_{p,j}| \leq O(\log n)$. Overall, we obtain $|\mathcal{C}'_p| = \sum_{j=1}^{k} |\mathcal{C}'_{p,j}| = \sum_{j=1}^{k} |\mathcal{M}'_{p,j}| \leq O(k \log n)$, as required. Hence, the theorem follows. ◀

The combination of Lemmas 12 and 13 together immediately implies Theorem 2.

6 Conclusions

Our main result (Theorem 1) is an $O(\log n)$-competitive algorithm for the online hitting set problem for n points in the plane, and a sequence of axis-aligned squares (or axis-aligned rectangles of bounded aspect ratio). This is the first online hitting set algorithm that is $O(\log n)$-competitive for geometric objects of arbitrary sizes in the plane. Our result further generalizes to positive homothets of any simple k-gon for $k \geq 3$, and achieves an $O(k^2 \log n)$-competitive algorithm. Even though a disk can be approximated by a regular k-gon as $k \to \infty$, our generalized result does not imply any competitive algorithm for disks. Very recently, Bhore, Gupta and Kumar [9] presented a randomized online hitting set algorithm with $O(\log n)$ expected competitive ratio for n points and a sequence of pseudo-disks in the plane, including disks of arbitrary radii, or positive homothets of a convex polygon (with arbitrarily many vertices). It remains an open problem whether deterministic $O(\log n)$-competitive algorithms are possible for these classes of geometric objects.

Does our result generalize to higher dimensions? Is there an $O(\log n)$-competitive online hitting set algorithm for axis-aligned cubes of arbitrary sizes in $\mathbb{R}^d$ for a constant dimension d? Is there one for $d = 3$? While BBD trees generalize to $\mathbb{R}^d$, for any constant dimension $d \in \mathbb{N}$, our online algorithm does not generalize to (hyper-)cubes in d-space. The reason is that the extremal points (cf. Section 3.2) do not necessarily capture the intersection $C_v \cap R$ of a cell C_v of the BBD tree and an axis-aligned box. In the plane, we have shown that C_v contains a vertex of R, or $C_v \cap R$ is a crossing intersection, or $C_v \cap R$ behaves as an axis-aligned halfplane with respect to the sub-rectangles of C_v (Section 3.2). However, in $\mathbb{R}^d$, $d \geq 3$, it is possible that R contains exactly one edge of the cell $C_v = r_{\text{out}}(v)$ and yet $C_v \cap R$ does not contain any of the extremal points of C_v. It remains open whether $O(\log n)$-competitive algorithms are possible for geometric objects of arbitrary sizes in dimensions $d \geq 3$.

References

1 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, and Abhinandan Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proc. 35th ACM Symposium on Principles of Database Systems, (PODS)*, pages 429–440. ACM, 2016. doi:10.1145/2902251.2902303.

2 Shanli Alefkhani, Nima Khodaveisi, and Mathieu Mari. Online hitting set of d-dimensional fat objects. In *Proc. 21st International Workshop on Approximation and Online Algorithms, (WAOA)*, volume 14297 of *LNCS*, pages 134–144. Springer, 2023. doi:10.1007/978-3-031-49815-2_10.

3 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.

4 Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1:1–1:54, 2009. doi:10.1145/1613676.1613677.

5 Sunil Arya and David M. Mount. Approximate range searching. *Comput. Geom.*, 17(3-4):135–152, 2000. doi:10.1016/S0925-7721(00)00022-5.

6 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.

7 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

8 Mark de Berg, Haggai David, Matthew J. Katz, Mark H. Overmars, A. Frank van der Stappen, and Jules Vleugels. Guarding scenes against invasive hypercubes. *Comput. Geom.*, 26(2):99–117, 2003. doi:10.1016/S0925-7721(03)00016-6.

9 Sujoy Bhore, Anupam Gupta, and Amit Kumar. Improved Online Hitting Set Algorithms for Structured and Geometric Set Systems. In *42nd International Symposium on Computational Geometry (SoCG 2026)*, volume 367 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.SoCG.2026.14.

10 Srecko Brlek, Hugo Tremblay, Jérôme Tremblay, and Romaine Weber. Efficient computation of the outer hull of a discrete path. In *Proc. 18th IAPR International Conference on Discrete Geometry for Computer Imagery (DGCI)*, volume 8668 of *LNCS*, pages 122–133. Springer, 2014. doi:10.1007/978-3-319-09955-2_11.

11 Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998. doi:10.1007/PL00009390.

12 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *Proc. 27th Symposium on Computational Geometry*, pages 1–10. ACM Press, 2011. doi:10.1145/1998196.1998198.

13 Timothy M. Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the RAM, revisited. *J. Comput. Geom.*, 9(2):45–66, 2018. doi:10.20382/JOCG.V9I2A5.

14 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi:10.1137/S0097539702418498.

15 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

16 Danny Z. Chen and Haitao Wang. Computing L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 81(6):2430–2483, 2019. doi:10.1007/S00453-018-00540-X.

17 Minati De, Saksham Jain, Sarat Varma Kallepalli, and Satyam Singh. Online geometric covering and piercing. *Algorithmica*, 86(9):2739–2765, 2024. doi:10.1007/S00453-024-01244-1.

18 Minati De, Ratnadip Mandal, and Satyam Singh. New lower bound and algorithms for online geometric hitting set problem. *arXiv*, 2024. doi:10.48550/arXiv.2409.11166.

19 Minati De and Satyam Singh. Online hitting of unit balls and hypercubes in $\mathbb{R}^d$ using points from $\mathbb{Z}^d$. *Theor. Comput. Sci.*, 992:114452, 2024. doi:10.1016/J.TCS.2024.114452.

20 Minati De, Satyam Singh, and Csaba D. Tóth. Online hitting sets for disks of bounded radii. In *Proc. 33rd European Symposium on Algorithms, (ESA)*, volume 351 of *LIPIcs*, pages 50:1–50:16. Schloss Dagstuhl, 2025. doi:10.4230/LIPIcs.ESA.2025.50.

21 Adrian Dumitrescu, Anirban Ghosh, and Csaba D. Tóth. Online unit covering in Euclidean space. *Theor. Comput. Sci.*, 809:218–230, 2020. doi:10.1016/J.TCS.2019.12.010.

22 Adrian Dumitrescu and Csaba D. Tóth. Online unit clustering and unit covering in higher dimensions. *Algorithmica*, 84(5):1213–1231, 2022. doi:10.1007/S00453-021-00916-6.

23 Guy Even and Shakhar Smorodinsky. Hitting sets online and unique-max coloring. *Discret. Appl. Math.*, 178:71–82, 2014. doi:10.1016/J.DAM.2014.06.019.

24 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

25 Ziyue Huang and Ke Yi. Approximate range counting under differential privacy. In *Proc. 37th International Symposium on Computational Geometry (SoCG)*, volume 189 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.SoCG.2021.45.

26 Arindam Khan, Aditya Lonkar, Saladi Rahul, Aditya Subramanian, and Andreas Wiese. Online and dynamic algorithms for geometric set cover and hitting set. In *Proc. 39th Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl, 2023. See also https://arxiv.org/abs/2303.09524. doi:10.4230/LIPIcs.SOCG.2023.46.

27 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990. doi:10.1007/BF01840386.

28 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.

29 Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987. URL: https://www.science.smith.edu/~jorourke/books/ArtGalleryTheorems/.

30 Haitao Wang. A new algorithm for Euclidean shortest paths in the plane. In *Proc. 53rd ACM SIGACT Symposium on Theory of Computing, (STOC)*, pages 975–988. ACM, 2021. doi:10.1145/3406325.3451037.

Incremental Strongly Connected Components with Predictions

Ronald Deng
Williams College, Williamstown, MA, USA
Samuel McCauley
Williams College, Williamstown, MA, USA
Aidin Niaparast
Carnegie Mellon University, Pittsburgh, PA, USA
Helia Niaparast
Carnegie Mellon University, Pittsburgh, PA, USA
Bennett Ptak
Williams College, Williamstown, MA, USA
Shirel Quintanilla
Williams College, Williamstown, MA, USA
Shikha Singh
Williams College, Williamstown, MA, USA
Nathan Vosburg
Williams College, Williamstown, MA, USA

Abstract

Algorithms with predictions is a growing area that aims to leverage machine-learned predictions to design faster beyond-worst-case algorithms. In this paper, we use this framework to design a learned data structure for the incremental strongly connected components (SCC) problem. In this problem, the n vertices of a graph are known a priori and the m directed edges arrive over time. The goal is to efficiently maintain the strongly connected components of the graph after each insert. Our algorithm receives a possibly erroneous prediction of the edge sequence and uses it to precompute partial solutions to support fast inserts. We show that our algorithm achieves nearly optimal bounds with good predictions and its performance smoothly degrades with the prediction error. We also implement our data structure and perform experiments on real datasets. Our empirical results show that the theory is predictive of practical runtime improvements.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases algorithms with predictions, learning augmented algorithms, incremental graph algorithms, strongly connected components, data structures

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.17

Related Version *Full Version*: https://arxiv.org/abs/2604.26062

Funding *Aidin Niaparast*: This material is based upon work supported in part by the Air Force Office of Scientific Research under award number FA9550-23-1-0031.

1 Introduction

Incremental computation on graphs is a fundamental primitive in many database and optimization problems. In an incremental graph problem, the vertices of the graph G are known ahead of time and the edges are revealed over time. The goal is to design an efficient data structure for answering queries at time t about some property (such as connectivity or shortest path) of the current graph G_t. Designing fast *worst-case* algorithms for incremental graph problems is an area of extensive research [7, 3, 5, 12, 13].

Algorithms designed to be fast in the worst case provide strong guarantees against adversarial inputs. However, these algorithms fail to exploit the correlations in the structure of typical instances. As most computations are performed repeatedly on similar datasets, heuristics optimized for domain specific inputs tend to be faster in practice.

A growing body of work, known as algorithms with predictions [25], seeks to bridge the gap between simple heuristics that lack strong theoretical guarantees and more complex worst-case algorithms that are provably efficient but often impractical. The main idea is to use machine-learned predictions to capture correlations in the input instances and leverage them to guide algorithmic decisions.

© Ronald Deng, Samuel McCauley, Aidin Niaparast, Helia Niaparast, Bennett Ptak, Shirel Quintanilla, Shikha Singh, and Nathan Vosburg;
licensed under Creative Commons License CC-BY 4.0

Predictions are particularly effective for online or dynamic problems, where the input is not known ahead of time. Predictions trained on historical data can then serve as a proxy for future inputs – turning a hard *online* problem into an easier-to-solve *offline* one. Using such predictions, an algorithm can then precompute partial solutions ahead of time and consequently perform faster updates and queries at runtime. The algorithms designed in this framework are referred to as *learning-augmented* or *learned* algorithms.

The running time of a learned algorithm is measured in terms of the quality of the prediction – the bound is stated in terms of both the size of the input and the prediction error. Ideally, a learned algorithm for an online problem (1) is nearly as fast as the best-offline solution when the prediction is perfect, (2) has bounded worst-case performance cost when the prediction is arbitrarily wrong, and (3) has performance that degrades gracefully with error. These properties in order are referred to as *consistency, robustness*, and *smoothness* respectively in the literature. Thus, learned algorithms leverage good predictions to be fast and have provable safeguards against bad predictions.

Incremental SCC. In this paper, we study the *incremental strongly connected components (SCC)* problem in the algorithms with predictions model. In this problem, the n vertices V of a directed graph G are known a priori and the m directed edges E are inserted one by one. Let $e_1, \ldots, e_m$ denote the sequence of edges and let $G_t = (V, \{e_1, \ldots, e_t\})$ denote the graph at time t. The goal is to maintain the **strongly connected components** of G_t at all times t. A strongly connected component $S \subseteq V$ is a maximal set of vertices of G such that every pair of vertices u, v are *mutually reachable*, that is, there is a directed path from v to u and vice versa.

Maintaining SCCs is a fundamental building block in many applications, e.g. social networks [9], graph processing [34], program analysis [14] and fault-tolerant networks [2].

DFS-based approaches such as Tarjan's algorithm [31] can compute the SCCs of a static offline graph G in $O(n + m)$ time. As a step towards solving the incremental SCC problem, consider the *offline incremental problem*. In this variant, the edge insert sequence $\sigma = e_1, e_2, \ldots, e_m$ is known ahead of time and the goal is to design a data structure to answer queries about the SCCs of each intermediate graph G_t. Naively running Tarjan's repeatedly takes $O(m^2)$ time.[1] A folklore divide-and-conquer approach [28] improves upon this naive approach and solves the offline incremental SCC problem in $O(m \log m)$ total time.

Offline to Learned Incremental SCC. To design a learned algorithm for the *online incremental problem*, we assume that a prediction $\hat{\sigma}$ of the edge sequence σ is given to the algorithm. We define the **prediction error** η as the maximum error between when an edge was predicted to arrive in $\hat{\sigma}$ and its actual arrival time in σ. To ensure this metric is well-defined, we assume that $\hat{\sigma}$ is a permutation of σ (we discuss this assumption further in Section 2.1). If the prediction is perfect ($\eta = 0$), the problem reduces to the offline incremental variant. If the prediction is arbitrarily erroneous ($\eta = m$), the algorithm effectively gets no information about the input and the problem reduces to the worst-case model. The main challenge is to design an algorithm that smoothly interpolates between the two regimes as a function of the prediction error η.

1.1 Our Contributions

One of our main contributions is designing the first learned data structure for the incremental SCC problem which is consistent, robust and smooth with respect to the prediction error.

[1] We assume $m \geq n$ to simplify the runtimes.

▶ **Theorem 1.** *We design a learned data structure that, given prediction $\hat{\sigma}$ with error η, maintains the strongly connected components of the graph in total time $O(m \log m \cdot \max\{\eta, 1\})$ for m inserts. Queries about the SCC of a given vertex take $O(1)$ time.*

To put this result in context, notice that if the edge arrivals are reasonably predictable (e.g. $\eta = O(\text{polylog}(m))$), our algorithm for the online incremental problem is nearly as fast as the offline variant.

The algorithmic idea is to use a divide-and-conquer approach on the predicted edge sequence $\hat{\sigma}$ to recursively precompute solutions to partial subproblems using Tarjan's algorithm. As edges arrive that are different from $\hat{\sigma}$, our algorithm updates the prediction and lazily recomputes the affected subproblems. At a high level, we show that we can bound the number of times each subproblem is recomputed by the prediction error η and thus lose an η factor over the offline incremental algorithm.

Robustness to Error. The running time of our algorithm grows with η – in fact, for sufficiently high η, our algorithm may be slower than known worst-case techniques. However, the algorithm can be made robust to arbitrarily erroneous predictions with respect to any worst-case algorithm by switching to the worst-case algorithm if the learned algorithm's runtime grows larger than the worst-case guarantee.

Similarly, our algorithm's running time is given in terms of the maximum error – as stated, even a single outlier can significantly impact performance guarantees. One potential strategy to handle a small number of outliers is to reset the algorithm (restarting from time zero with the outlier predicted correctly) each time an outlier arrives. Each reset is as expensive as an entire algorithm run, but between resets, performance is proportional to the maximum error among non-outliers. While this strategy is expensive on a per-outlier basis, it can be used to handle a small number of outliers without degrading the performance too much. In our experiments, our algorithm (without this intervention) seems to be somewhat resistant to outliers – its performance on our datasets is better than the maximum error would indicate.

Comparison with Worst-Case Approaches. Most prior algorithms for maintaining SCCs in incremental graphs do so by maintaining *topological ordering* of the *condensed graph* (a directed acyclic graph in which all vertices within an SCC are contracted into a new single node). A topological ordering is a labeling $L : V \to \mathbb{Z}$ of the vertices V such that $L(v) < L(u)$ if there is a directed path from v to u. A topological ordering exists if and only if the directed graph is acyclic. An algorithm that maintains the topological ordering can detect new cycles whenever a new edge points from a node with a higher label to a lower label. Such a cycle leads to the merging of previous (condensed) SCCs.

The best-known worst case total running times of algorithms using the topological-ordering-based combinatorial approach are $\tilde{O}(n^2)$ [3] for dense graphs and $\tilde{O}(m^{4/3})$ [5] for sparse graphs. Note that the $\tilde{O}$-notation hides logarithmic factors. Thus, even with reasonably large error as long as $\eta < \min\{n^2/m, m^{1/3}\}$, our algorithm improves upon these approaches.

In a recent theoretical advancement, Chen et al. [7] use the interior-point method to maintain SCCs in incremental graphs (without a topological ordering) in total time $m \cdot e^{O(\log^{167/168} m \log\log m)}$. While the improvement is theoretically appealing, the algorithm and techniques employed are not practical and the running time itself is galactic.

Faster algorithms are known for maintaining SCCs in the *decremental* setting (where all edges are available at the beginning and deleted one by one). This can be done in total expected time $O(m \log^4 m)$ [6]. Predictions of the edge sequence essentially help turn an incremental SCC problem (which seems to be harder) into a decremental one.

In contrast to all prior work, our algorithm maintains SCCs without maintaining a topological ordering or using complex techniques. Our algorithm exploits predictions to precompute partial solutions to recursive subproblems. Our recursive technique that uses predictions to lift the offline problem to an online one follows the approach used by McCauley et al. [21] for maintaining approximate shortest paths in incremental graphs using predictions.

One way our paper differs from the previous works on dynamic graph problems with predictions [32, 20, 21, 15] is that we co-designed the algorithm with its implementation. Concretely, our analysis explicitly tracks logarithmic factors in running time, and avoids constant-factor overheads when possible (e.g. rebuilding unnecessary subproblems as in [21]).

Experimental Results. We compare the runtime of our algorithm to the state-of-the-art heuristic IncSCC$^+$ [12] on real datasets[2]. IncSCC$^+$ maintains a topological ordering of the condensed graph and optimizes over running Tarjan's from scratch repeatedly. Despite the optimizations, their worst-case total cost is $\Theta(m^2 \log m)$ for m inserts.

Our experiments show that (1) our algorithm is scalable and can handle large datasets, (2) it outperforms the known recursive algorithm for the offline problem under perfect predictions, (3) it significantly outperforms IncSCC$^+$ on reasonably accurate predictions, and finally (4) its runtime degrades gradually with error. These results complement our theoretical analysis: they give evidence that the algorithm is reasonably simple, and its running time does not involve large constants.

Summary. In summary, our theoretical and experimental results demonstrate that predictions can help bridge the gap between theoretically-good algorithms and practical heuristics. Our results provide a proof-of-concept for designing algorithms with strong provable guarantees that can match or even outperform state-of-the-art heuristics on predictable inputs.

1.2 Additional Related Work

Algorithms with Predictions For Runtime. Kraska et al. [16] jump-started the area by *empirically* demonstrating how machine-learned predictions can speed up data structures. Dinitz et al. [11] present a *theoretical* framework for analyzing predictions for speeding up *offline* algorithms. This has since been used for problems such as maximum flow [8, 27], shortest path [17], minimum cut [26], stable matching [24], and convex optimization [30].

Predictions have been used to design provably faster data structures for problems such as online list labeling [22], incremental topological ordering [23], binary search [10], sorting [1], dictionaries [33, 19], and priority queries [4]. Similar to this paper, predictions have been used to "lift" offline solutions to dynamic graph problems to improve the runtime of incremental graph problems, such as incremental shortest paths [32, 15, 21] and divide-and-conquer algorithms [20]. We note in particular that our divide-and-conquer framework is similar to that of McCauley et al. [21], however, we keep only a single path rather than maintaining the entire tree. This improves runtime performance by several log factors and also makes the data structure more practical. Liu et al. [20] also use a divide-and-conquer framework, but their analysis requires balanced subproblem sizes. In contrast, we can handle unbalanced subproblems with the trade-off that our error bounds are on max error (rather than average). Overall, these results demonstrate significant potential of leveraging machine-learned predictions to speed up algorithms and data structures.

[2] We ended up optimizing IncSCC$^+$ beyond what was intended in [12]. We include runtime comparison against both variants.

2 Preliminaries

This section presents the formal setup and a warm-up algorithm for the offline incremental SCC problem.

2.1 Model and Notation

Let V be the set of vertices of the directed graph G. Let $\sigma = e_1, \ldots, e_m$ denote the sequence of online edge inserts. We assume that one edge arrives per time step, so we often say that e_t arrives at "time t." We use "graph at time t", denoted by G_t, to refer to the graph with vertex set V and the first t edges $e_1, \ldots, e_t$. We define G_0 as the graph on vertex set V with no edges. We assume that G_m has no isolated vertices; if it is not then all isolated vertices in G_m can be removed before running the algorithm, increasing the cost of the algorithm by $|V|$. Before any edges arrive, the learned algorithm receives a prediction $\hat{\sigma}$ of σ. We define $m = |\sigma| = |\hat{\sigma}|$. Let $i(e)$ be the index of e in σ and $\widehat{i}(e)$ be the index of e in $\hat{\sigma}$. Let $\eta_e = |i(e) - \widehat{i}(e)|$ for each edge e in σ. The error $\eta = \max_{e \in \sigma} \eta_e$ is the maximum error of any edge; for simplicity of exposition, we assume $\eta \geq 1$ (the analysis is asymptotically the same if $\eta = 0$ or $\eta = 1$). As we go through the edges, we update the predictions, and we denote the predicted arrival sequence at time t by $\hat{\sigma}_t$. This is the same prediction model as past work on incremental graph algorithms [15, 32, 21].

We assume that the set of edges in $\hat{\sigma}$ and σ are the same – that is, $\hat{\sigma}$ is a permutation of σ. It is possible to generalize this assumption, albeit with some extra cost. If an edge e arrives that is not in $\hat{\sigma}$, the algorithm can simply start over from the beginning, at a cost of $O(m \log m)$ – this does not increase the overall cost if the number of such edges is at most η. Edges in $\hat{\sigma}$ that never arrive are more expensive, since effectively $\eta_e = m$. One strategy to handle such edges is to use a large threshold τ: at each time step t, the algorithm can remove any edge that was predicted to arrive before $t - \tau$; if the edge later arrives it can be treated as an edge that is not in $\hat{\sigma}$.

Our goal is to maintain a data structure that, after edge e_t arrives, can answer queries to determine if u and v are in the same SCC of G_t, for any $u, v \in V$.

2.2 Warm Up: Offline Incremental SCC

Our algorithm is based closely on the offline recursive algorithm for the problem. For the offline incremental problem, the entire sequence of edge insertions σ is known ahead of time. The goal is to quickly process σ to allow us to answer SCC queries for G_t for any t. We describe a folklore recursive method to perform this preprocessing efficiently. This algorithm has been described in programming contest literature, see e.g., [28], and a similar structure has been used for designing a worst-case algorithm for decremental SCC [29].

The algorithm recursively divides the interval $[0, m]$ into halves. Each interval generated in this process corresponds to a **subproblem**. In particular, the resulting intervals form a hierarchical set of dyadic subintervals (for ease of exposition, we assume that m is a power of two, although this assumption is not necessary for the algorithm or its correctness). Each subproblem has two endpoints ℓ and r. We refer to a subproblem either by its endpoints, $[\ell, r]$, or by its midpoint, $x := (r + \ell)/2$. The subproblem $[\ell, r]$ has a **left child** $[\ell, x]$ and a **right child** $[x, r]$, and is referred to as their **parent** subproblem. See Figure 1 for a schematic view of the recursion tree.

For each interval $[\ell, r]$, the algorithm maintains a graph $\tilde{G}_x = (V_x, E_x)$. $\tilde{G}_x$ is defined recursively below, but to help motivate the definition let us briefly summarize its contents. Consider contracting all SCCs of G_ℓ into single vertices, retaining only edges between two

distinct SCCs. These vertices and edges are enough to determine what SCCs are created during $[\ell, r]$. Now, consider trimming the graph further: we remove all edges that do not contribute to forming a new SCC during $[\ell, r]$, and remove all isolated vertices from the graph; trimming the graph in this way still allows us to determine the correct SCCs during $[\ell, r]$. This is the idea behind $\tilde{G}_x$.

At the root of the recursion tree, $\ell = 0$, $r = m$ and $\tilde{G}_x = G_{m/2}$. The algorithm runs Tarjan's algorithm [31] on G_x to compute its SCCs. It uses the SCCs to compute the graphs for the left and right subproblems $\tilde{G}_x^{\text{left}}$ and $\tilde{G}_x^{\text{right}}$ respectively. We define $\tilde{G}_x^{\text{left}}$ using its vertex set V_x^{left} and edge set E_x^{left}; $\tilde{G}_x^{\text{right}}$ is defined likewise.

Right Subproblem. Consider a pair of vertices u and v in the same strongly connected component S of $\tilde{G}_x$. Since edges are only ever inserted, u and v will remain in the same SCC at all time steps after x. Thus, to construct the right subproblem, the algorithm contracts all nodes within the same SCC in $\tilde{G}_x$ into a single node and deletes all edges (u, v) such that u and v are within the same SCC. In particular, for the right subproblem corresponding to the interval $[x, r]$, it sets V_x^{right} to contain a contracted node for each strongly connected component in $\tilde{G}_x$. The edge set E_x^{right} includes all edges $(u, v) \in E_x$ such that (u, v) is an *inter SCC* edge with endpoints u and v in different SCCs in $\tilde{G}_x$.

Left Subproblem. Now consider an edge (u, v) in $\tilde{G}_x$ with end points in different SCCs of $\tilde{G}_x$. Then, the vertices u and v will be in different SCCs at all time steps before x. The algorithm can safely delete such edges when recursing on the left subproblems. That is, V_x^{left} contains all vertices in V_x with at least one incident edge, and E_x^{left} only contains the *intra-SCC* edges in E_x with both end points in the same SCC.

Recursion Tree. The algorithm recurses on the left and right subproblems until the interval $[\ell, r]$ has length two. This way, each time $t = 1, \ldots, m-1$ is the midpoint of exactly one subproblem in the recursion tree. Notice that the nodes of this recursion tree in an in-order traversal correspond to the sequence of graphs $G_0, \ldots, G_m$; see Figure 1.

Queries. The folklore solution [28] builds a graph to implicitly track SCC merges. Here we describe one simple way to handle the queries recursively instead.

To see if u and v are in the same SCC at time t, we begin at the root and find the descendant of the root containing t. If it is a right child, u and v may each be merged into a new vertex. If so, we replace u (resp. v) with the merged vertex, and recurse. When we reach a subproblem with midpoint t, we check if u and v are in the same SCC in $\tilde{G}_t$, and return the result. This takes $O(\log m)$ time.

In Section 3.2, we present a data structure for the online problem that explicitly maintains node labels over time and answers queries in $O(1)$ time.

Analysis. Each time the algorithm processes a subproblem, it recurses on two subproblems obtained by splitting the time interval into two halves, implying a recursion depth of $O(\log m)$. Running Tarjan's algorithm on a subproblem with s edges takes $O(s)$ time. At each recursive call, an edge is either an inter- or intra-SCC edge, and thus is only passed down to one of the left or right subproblems. Thus, each edge contributes to one subproblem per level and the total runtime is $O(m)$ per level of the recursion tree. The total insertion cost is thus $O(m \log m)$.

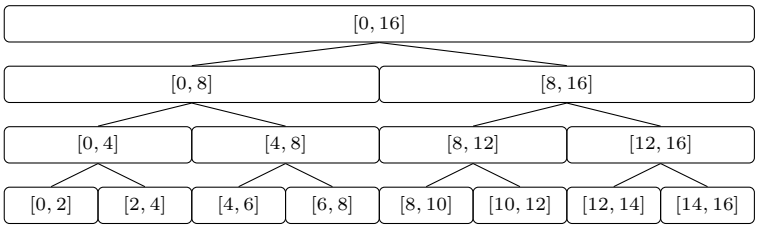


Figure 1 Recursive tree of subproblems for $m = 16$.

3 Incremental SCC with Predictions

In this section, we describe how we adapt the offline recursive algorithm for the incremental SCC problem with predictions. That is, given a prediction $\hat{\sigma}$ of σ at the beginning, we describe how to create a data structure that at each time step t, efficiently: (1) inserts e_t (the t-th edge in σ), and (2) allows answers to SCC queries on G_t.

First, notice that if the prediction sequence $\hat{\sigma}$ is *perfect* ($\hat{\sigma} = \sigma$), then the online problem reduces to the offline incremental problem. However, in general, the prediction $\hat{\sigma}$ could differ from σ significantly, and naively redoing the offline recursive approach each time is too expensive. Instead, our algorithm *only maintains a single path of the recursion tree* from Section 2.2; when each edge arrives, the algorithm updates the path, and updates the SCC of each vertex. In particular, initially, the algorithm only computes the root-to-left path from subproblem $[0, m]$ to subproblem $[0, 0]$ in the offline recursion tree built using the prediction $\hat{\sigma}$. As edges arrive, it updates the predicted sequence $\hat{\sigma}$ and recomputes the current root-to-leaf path for time step t. In our analysis, we show that we only incur recomputations that we can directly charge to the error in the prediction, that is, the algorithm is only $O(\eta)$-factor worse than the offline incremental algorithm.

3.1 Learned Incremental SCC Algorithm

First, we describe how our algorithm, **Learned IncSCC**, maintains and uses the prediction.

Our algorithm continuously updates $\hat{\sigma}$ as edges arrive. Let $\hat{\sigma}_t$ denote the updated prediction after t edges have arrived and $\hat{\sigma}_0 = \hat{\sigma}$. Thus $\hat{\sigma}_t$ agrees with σ on the first t edges. Let t and $\hat{t}$ denote the arrival time step of an edge e_t in σ and $\hat{\sigma}_{t-1}$ respectively. If $\hat{\sigma}_{t-1}$ was correct, then $\hat{t} = t$. Otherwise, we update $\hat{\sigma}_{t-1}$ to obtain $\hat{\sigma}_t$: we delete e from time $\hat{t}$, and insert it to time t. All edges in $\hat{\sigma}_{t-1}$ between t and $\hat{t}$ are shifted one time step later (note that $t' > t$); this gives $\hat{\sigma}_t$.

Now, we discuss how subproblems are maintained. At a high level, instead of building the entire recursive tree of the offline algorithm from Section 2.2 with respect to $\hat{\sigma}_0$, Learned IncSCC is more conservative. It lazily maintains the only thing needed to answer queries about the SCCs at the current timestep t: the path from the root to t in the recursive tree that corresponds to the updated prediction $\hat{\sigma}_t$.

Before any edges arrive, the algorithm uses the prediction $\hat{\sigma}_0$ and the offline recursive approach to **build** all subproblems along the path from the root $[0, m]$ to the leaf subproblem $[0, 0]$. We describe the implementation of this build in more detail momentarily.

Now consider step t when edge e_t is about to arrive. Let $\hat{t}$ be the time e_t was predicted to arrive in $\hat{\sigma}_{t-1}$. Before e_t arrives, the algorithm maintains the path from the root to time step $t-1$ of the recursion tree corresponding to $\hat{\sigma}_{t-1}$. After e_t arrives, the algorithm updates $\hat{\sigma}_t$ and the root-to-leaf path it maintains as follows. The algorithm begins with the last

subproblem in the current path, and walks backwards through the path to find a subproblem with interval $[\ell, r]$ such that $\hat{t} \in [\ell, r]$. Thus, $[\ell, r]$ is the lowest common ancestor (LCA) of the nodes $t-1$ and $\hat{t}$ in the recursion tree with respect to $\hat{\sigma}_{t-1}$. The algorithm calls the build procedure on this subproblem (and thus recursively on its children) to compute the path from the root to t with respect to $\hat{\sigma}_t$.

What We Store. Our algorithm maintains a path from the root to the current time t, implemented as an array of at most $\lceil \log_2 m \rceil$ pointers to subproblems.

Our algorithm maintains the following for each subproblem $[\ell, r]$ with midpoint x: (1) a graph $\tilde{G}_x = (V_x, E_x)$ whose nodes are contracted nodes of V, (2) the SCCs of $\tilde{G}_x$, (3) a mapping M_x from each vertex in $\tilde{G}_x$ to its **representative vertex** in V, and (4) a set of **left edges** and **right edges** that correspond to the edges of the left and right subproblems E_x^{left} and E_x^{right} as defined in Section 2.2. Each subproblem stores the edges of its children so that the edges do not need to be recomputed when the children are rebuilt.

Finally, our algorithm maintains the Node Label Data Structure which stores at all times t, a **label** $L(v)$ for each vertex $v \in V$, such that $L(u) = L(v)$ at time t if and only if u and v are in the same SCC in G_t. We use this data structure as a blackbox below and defer its implementation to Section 3.2.

Building a Subproblem. Below we describe how the algorithm performs a **build** operation at time t on an interval $[\ell, r]$ with midpoint x with respect to the predicted sequence $\hat{\sigma}_t$.

The algorithm begins by calculating the graph $\tilde{G}_x$ and the mapping M_x. We split into two cases depending on if $[\ell, r]$ is the root; in the case it is not, we split into further cases depending on if it is a left or right child.

If $[\ell, r]$ is the root subproblem, then $\tilde{G}_x = (V_x, E_x)$ where $V_x = V$ and E_x contains all edges in $\hat{\sigma}_t$. The mapping M_x has $M_x(v_i) = v_i$ for all vertices v_i.

If $[\ell, r]$ is not the root, let $[\ell^p, r^p]$ denote its current parent node. Let $\tilde{G}_x^p = (V_x^p, E_x^p)$ and M_x^p denote the graph and mapping stored by the parent subproblem. There are two cases depending on whether the interval $[\ell, r]$ is the left or right child of $[\ell^p, r^p]$.

If $[\ell, r]$ is the left child of $[\ell^p, r^p]$, then V_x is the set of vertices in V_x^p with at least one incident edge, i.e., incoming or outgoing, and the edge set E_x is the left edges stored in the parent subproblem $\tilde{G}_x^p$. The mapping is inherited from the parent, that is, $M_x = M_x^p$.

If $[\ell, r]$ is the right child of $[\ell^p, r^p]$, the algorithm contracts the vertices in the same SCC of $\tilde{G}_x^p$ to a single node, and these contracted nodes form V_x. The edge set E_x is the right edges stored in the parent subproblem $\tilde{G}_x^p$. For each node $c \in V_x$, which corresponds to an SCC in $\tilde{G}_x^p$, we compute $M_x(c)$ as follows. Let v be an arbitrary node in the SCC corresponding to c. We set $M_x(c) := M_x^p(v)$.

Now that $\tilde{G}_x$ and M_x are computed, the algorithm uses them to finish processing the subproblem. Similar to the offline case, the algorithm computes the strongly connected components of $\tilde{G}_x$ using Tarjan's algorithm, considering only the edges in E_x that appear at timesteps at most x according to $\hat{\sigma}_t$.

First, consider the case when the midpoint $x \neq t$. The algorithm computes the left edges of $\tilde{G}_x$ as those whose endpoints lie in the same SCC in $\tilde{G}_x$, and the right edges as those whose endpoints lie in different SCCs. These left and right edges are stored in the parent subproblem and inherited by child subproblems (rather than being recomputed when a child is rebuilt). After computing the left and right edges, the algorithm recursively builds the child containing t.

The base case occurs when the midpoint $x = t$. In the base case, the algorithm performs a final merge of the labels of nodes in the same SCC of $\tilde{G}_x$ in the Node Label Data Structure; we define this merge below.

3.2 Node Label Data Structure

The *Node Label Data Structure* stores, at all times t, the SCC of each vertex in G_t. It consists of: (1) an array L of size n, storing a *label* for each vertex such that $L[i] = L[j]$ if and only if vertices v_i and v_j are in the same SCC in G_t; and (2) for each label ℓ, a doubly linked list of all vertices with label ℓ (in other words, a list of all vertices v_i such that $L[i] = \ell$).

At initialization, $L[i] = i$ for all $i \in [n]$.

On a query to see if vertices v_i and v_j are in the same SCC, we simply check if $L[i] = L[j]$. This leads to $O(1)$ query time.

Given a collection of vertices V_c, the **merge** operation merges the SCCs of all vertices in V_c into a single SCC. Let V_c consist of k vertices $v_{c_1}, v_{c_2}, \ldots, v_{c_k}$. For $i = 1, \ldots, k-1$, we find labels $L[c_i]$ and $L[c_{i+1}]$, and determine which has a smaller list. Let ℓ_1 be the label of the smaller list (breaking ties arbitrarily) and ℓ_2 be the other. We iterate through the list of all vertices with label ℓ_1; for each vertex v_j, we set $L[j] = \ell_2$. Then, we add all vertices with label ℓ_1 to the list of vertices with label ℓ_2. Since the lists are implemented as linked lists, each merge operation takes time proportional to the number of vertices with label ℓ_1.

4 Running Time Analysis

In this section, we analyze the running time of our learned algorithm. We defer proofs related to correctness to the full version of the paper. To start, we bound what edges in the predicted edge sequence $\hat{\sigma}$ can cause a given subproblem to be rebuilt.

▶ **Lemma 2.** *If a subproblem $[\ell, r]$ with midpoint x using prediction $\hat{\sigma}$ is rebuilt at time t, then the edge e that arrives at time t was predicted to arrive in $\hat{\sigma}$ at a time either in $[x, x+\eta]$ or in $[r, r+\eta]$.*

Proof. The subproblem $[\ell, r]$ is rebuilt only when it lies on the path from $\mathrm{LCA}(t, \hat{t})$ to t for some edge. There are two ways this can happen:

1. $[\ell, r]$ is the $\mathrm{LCA}(t, \hat{t})$ for some pair $(t, \hat{t})$. This occurs only if $t \le x$ and $\hat{t} \ge x$. Since $|t - \hat{t}| \le \eta$, this requires that $\hat{t} \in [x, x+\eta]$.
2. $[\ell, r]$ lies strictly below the LCA. This only happens if t lies inside $[\ell, r]$, but $\hat{t}$ lies outside the subtree rooted at $[\ell, r]$; therefore, $\hat{t} \ge r$. Since $[\ell, r]$ is on the path to t, we must have $t \le r$; since $\hat{t} - t \le \eta$, we have that $\hat{t} \in [r, r+\eta]$. ◀

Before the next lemma, we motivate the structure of our analysis. Lemma 2 states that each subproblem is rebuilt only $O(\eta)$ times; this means that if we can bound the size of all subproblems (in other words, the total number of edges in all subproblems) we are done. Indeed, the classic analysis of the offline problem with $\eta = 0$ states that each edge is in at most one subproblem at each level. This would mean that the total size of all subproblems is $O(m \log m)$; combining with Lemma 2 would give the desired $O(\eta m \log m)$ bound.

Unfortunately, this analysis is incorrect. The reason is that as $\hat{\sigma}$ changes, what edge belongs to each subproblem changes as well. For example, according to $\hat{\sigma}_0$, the endpoints of an edge $e = (u, v)$ might not be in the same SCC in $G_{m/2}$, which puts e in the right child of the root. But at some later time $t < m/2$, an edge originally predicted to arrive at $t' > m/2$ might arrive that puts u and v in the same SCC in $G_{m/2}$. This would place e in the left child of the root. Lemmas 3 through 5 bound how often this can happen – i.e., an edge may be a part of $O(\eta \log m)$ builds in total.

For any edge (u, v) and prediction $\hat{\sigma}$, let the **combining time** $C_{u,v}(\hat{\sigma})$ be the first time in $\hat{\sigma}$ when u and v are in the same SCC. If u and v are never in the same SCC in $\hat{\sigma}$, then $C_{u,v}(\hat{\sigma})$ is undefined. Lemma 3 shows that at any time t, an edge (u, v) is only in a subproblem that contains its combining time. This lemma is useful both in analyzing the running time and proving correctness.

▶ **Lemma 3.** *Let $[\ell, r]$ be a subproblem in the offline recursive tree built on $\hat{\sigma}_t$ that contains edge $e = (u, v)$. If $C_{u,v}(\hat{\sigma}_t)$ is defined, then $C_{u,v}(\hat{\sigma}_t) \in [\ell, r]$; if $C_{u,v}(\hat{\sigma}_t)$ is undefined, then $r = m$.*

Proof. We argue by contradiction.

First, consider the case where $C_{u,v}(\hat{\sigma}_t)$ is undefined; assume by contradiction that e is in a subproblem $[\ell, r]$ with $r \neq m$. Let $[\ell, r]$ be the subproblem closest to the root containing e where $r \neq m$. Then $[\ell, r]$ was a left child – but by definition, this means that u and v were in the same component in the graph of the parent of $[\ell, r]$; a contradiction.

If $C_{u,v}(\hat{\sigma}_t)$ is defined, let $[\ell, r]$ be the subproblem in the current path closest to the root such that e is an edge in $[\ell, r]$, but $C_{u,v}(\hat{\sigma}_t) \notin [\ell, r]$.

If $C_{u,v}(\hat{\sigma}_t) < \ell$, let $[\ell_a, r_a]$ be the ancestor of $[\ell, r]$ with midpoint ℓ. $[\ell_a, r_a]$ must exist since if $C_{u,v}(\hat{\sigma}_t)$ exists, $\ell > 0$. Furthermore, $[\ell, r]$ must be a descendant of the right child of $[\ell_a, r_a]$. But since $C_{u,v}(\hat{\sigma}_t) < \ell$, u and v are in the same SCC in the midpoint of $[\ell_a, r_a]$, so e cannot be in the right edges of $[\ell_a, r_a]$, contradicting that it is an edge in $[\ell, r]$.

If $C_{u,v}(\hat{\sigma}_t) > r$, let $[\ell_a, r_a]$ be the ancestor of $[\ell, r]$ with midpoint r. $[\ell_a, r_a]$ must exist since if $C_{u,v}(\hat{\sigma}_t)$ exists, $r < m$. Furthermore, $[\ell, r]$ is a descendant of the left child of $[\ell_a, r_a]$. But since $C_{u,v}(\hat{\sigma}_t) > r$, u and v are in different SCCs in the midpoint of $[\ell_a, r_a]$, so e cannot be in the left edges of $[\ell_a, r_a]$, contradicting that it is an edge in $[\ell, r]$. ◀

Now, we show that the error bound η implies that the combining time of any pair of vertices cannot differ significantly as $\hat{\sigma}$ is updated. First, we need a lemma showing that the way we update predictions does not increase the error of any edge beyond η.

▶ **Lemma 4.** *For any edge e and any $\hat{\sigma}_t$, let $i_t(e)$ be the position of e in $\hat{\sigma}_t$, and $i(e)$ be the position of e in σ. Then if $\hat{\sigma}_0$ has maximum error η, we have $|i(e) - i_t(e)| \leq \eta$ – thus, $\hat{\sigma}_t$ also has maximum error η. This implies that for each t_1 and t_2 we have $|i_{t_1}(e) - i_{t_2}(e)| \leq 2\eta$.*

Proof. We prove this by induction on t; it is trivially true for $t = 0$.

Note that when we move from $\hat{\sigma}_{t-1}$ to $\hat{\sigma}_t$, we move an edge from $\hat{t} \geq t$ to its final location t; all edges in positions from t to $\hat{t} - 1$ move forward one, and all others remain unchanged. By definition, $\hat{t} - t \leq \eta$. The edge e being moved to t has $i_t(e) = i(e)$.

Any edge e being moved forward one (i.e. an edge between t and $\hat{t} - 1$ in $\hat{\sigma}_{t-1}$) that has $i(e) \geq \hat{t}$ is being moved closer to $i(e)$; by the inductive hypothesis the lemma holds. Now suppose an edge e is being moved forward one, and $i(e) \leq \hat{t}$. For all the edges that move forward we have $i_t(e) \leq \hat{t}$. Furthermore, since e has not arrived, $i(e) > t$. Combining with $\hat{t} - t \leq \eta$ implies $|i_t(e) - i(e)| \leq \eta$. ◀

Finally, Lemma 5 states that the combining time does not change significantly as we update the predictions.

▶ **Lemma 5.** *For any pair of vertices u, v, and any $\hat{\sigma}_t$, $C_{u,v}(\hat{\sigma}_t) \in [C_{u,v}(\hat{\sigma}_0) - 2\eta, C_{u,v}(\hat{\sigma}_0) + 2\eta]$.*

Proof. Consider the graph G' constructed using all vertices in V, and the first $C_{u,v}(\hat{\sigma}_0)$ edges in $\hat{\sigma}_0$; u and v are in the same strongly connected component in G'. Let G'' be the graph constructed using vertices in V, and the first $C_{u,v}(\hat{\sigma}_0) + 2\eta$ edges in $\hat{\sigma}_t$. By Lemma 4,

all edges in G' must have already arrived by time $C_{u,v}(\hat{\sigma}_0) + 2\eta$ in $\hat{\sigma}_t$, so the edges of G'' are a superset of the edges of G'; that is, u and v must be in the same SCC in G''. Therefore, $C_{u,v}(\hat{\sigma}_t) \leq C_{u,v}(\hat{\sigma}_0) + 2\eta$.

Let G' be the graph at time $C_{u,v}(\hat{\sigma}_0) - 1$ for $\hat{\sigma}_0$; u and v are not in the same strongly connected component in G'. Let G'' be the graph at time $C_{u,v}(\hat{\sigma}_0) - 2\eta - 1$ for $\hat{\sigma}_t$. By Lemma 4, all edges in G'' must have already arrived by $C_{u,v}(\hat{\sigma}_0) - 1$ in $\hat{\sigma}_0$, so G' has every edge G'' has. This means that u and v are not in the same strongly connected component in G'', which implies that $C_{u,v}(\hat{\sigma}_t) \geq C_{u,v}(\hat{\sigma}_0) - 2\eta$. ◀

We are ready to prove Theorem 1.

Proof of Theorem 1. Each time a node is relabeled in the Node Label Data Structure, the size of its SCC doubles; thus the total cost of all merges is $O(n \log n)$.

The cost to build a subproblem is linear in the number of vertices of $\tilde{G}_x$ and the number of edges $\tilde{E}_x$. Let $\mathcal{S}(t)$ be the set of subproblems built at time t. The total cost to build all subproblems is

$$\sum_t \sum_{t' \in \mathcal{S}(t)} (\#\text{ edges in } t') = \sum_{e \in \sigma} (\#\text{ builds of subproblems containing } e).$$

We bound this final sum. First, any $e = (u, v)$ where $C_{u,v}(\hat{\sigma}_t)$ is undefined can only be in subproblems with $r = m$ by Lemma 3; there are at most $\log m$ such subproblems, and each is rebuilt $O(\eta)$ times by Lemma 2, so the total cost of these edges is $O(m\eta \log m)$.

Now suppose $C_{u,v}(\hat{\sigma}_t)$ is defined and $e = (u, v)$ is in $[\ell, r]$ built at time t. We show that in each level, over time, there are $O(\eta)$ such subproblems. Summing over all $\log m$ levels gives a cost of $O(\eta m \log m)$.

If e is in $[\ell, r]$ at time t, by Lemma 3 we have $C_{u,v}(\hat{\sigma}_t) \in [\ell, r]$, and Lemma 5 implies that $C_{u,v}(\hat{\sigma}_0) \in [\ell - 2\eta, r + 2\eta]$.

Consider a depth d, such that $r - \ell \geq \eta$ for subproblems at depth d. There are at most five subproblems at depth d that have $C_{u,v}(\hat{\sigma}_0) \in [\ell - 2\eta, r + 2\eta]$. Each is rebuilt $O(\eta)$ times by Lemma 2; therefore, there are $O(\eta)$ subproblem builds at depth d that contain e.

Now we consider a depth d such that all subproblems have $r - \ell < \eta$. If a subproblem $[\ell, r]$ in this level contains e, then $C_{u,v}(\hat{\sigma}_0) \leq r + 2\eta < \ell + 3\eta$ and $C_{u,v}(\hat{\sigma}_0) \geq \ell - 2\eta > r - 3\eta$, which imply $\ell > C_{u,v}(\hat{\sigma}_0) - 3\eta$ and $r < C_{u,v}(\hat{\sigma}_0) + 3\eta$, respectively. Then if e' causes the subproblem $[\ell, r]$ to be rebuilt, by Lemma 2, e' was predicted to arrive at time steps $[\ell, r + \eta]$ in $\hat{\sigma}$. The length of this interval is $O(\eta)$, which means that e has been involved in $O(\eta)$ rebuilds at depth d over time. ◀

5 Experimental Evaluation

In this section, we describe our experimental results.[3]

Incremental SCC$^+$. We compare against the state-of-the-art heuristic for the problem: IncSCC$^+$ [12]. In IncSCC$^+$, the algorithm maintains a topological ordering of the nodes at all times, that is, each node v in the current (possibly contracted) graph is assigned a *topological rank* $r(v)$ such that for every arc (u, u') between distinct SCCs, we have $r(u) > r(u')$. When

[3] The Python implementations of our experiments are available at `https://github.com/helia-niaparast/incremental-strongly-connected-components-with-predictions`.

a new arc (v, w) is added in the "wrong" direction, i.e., $r(v) < r(w)$, the algorithm identifies potentially affected nodes by running forward and backward depth-first searches from w and v, respectively. It then applies Tarjan's algorithm on the affected nodes to determine whether a new cycle has been formed. We were not able to obtain the authors' original code, so we implemented our own version of their algorithm. In this implementation, when a cycle is formed, we merge the SCCs in the cycle into a new SCC, and run DFS on the whole graph to find the new topological ordering.

We also made an optimized version (IncSCC+ optimized) with two modifications. First, we skip the final run of Tarjan's algorithm, as cycle formation can be detected immediately from the affected nodes. Second, after a node is contracted, in order to find a new valid topological ordering, we only rearrange the affected nodes. These factors do not change the main ideas of the algorithm, but lead to a noticeable performance improvement in practice. In our tests, we compare to both IncSCC$^+$ implementations.

We also include the performance of the **offline incremental algorithm** from Section 2.2, which has access to the entire edge arrival sequence ahead of time.

Datasets. Our datasets are from SNAP[4] [18]. The `sx-superuser-c2a` and `sx-askubuntu` datasets are graphs based on two stackexchange forums; a link represents one user posting an answer or comment on another user's question or answer. These two are temporal, i.e., links are ordered based on the time the comments occurred. The other dataset used, `Slashdot0811`, is not temporal; edges arrive in random order. Each edge represents a user who tagged another as "friend" or "foe." The number of vertices and number of unique edges for each of these datasets is given in Table 1.

Experimental Setup. Our implementations are in Python. Our experiments were run on an Ubuntu Linux machine with a 2.8GHz i7-1165G7 Intel CPU with 32GB RAM.

Predictions. We parameterize how to generate predictions with a variable S, which is roughly the standard deviation of the error. For each value of S, we construct a perturbed edge sequence from the original ordering as follows. We traverse the edges in their original order, and for each edge, we sample an offset from a normal distribution with mean 0 and standard deviation S. The target index is set to the current index plus this offset. If both the current and target indices have not yet been modified, we swap the edges at these positions; otherwise, the edge remains fixed.

After processing all edges, we measure the prediction quality using two metrics relative to the original sequence: the average prediction error (η_{avg}) and the maximum prediction error ($\eta_{\max}$). For each $S \in \{0, 10, 20, \ldots\}$, we generate 10 independent perturbed sequences using the procedure described above. Each sequence yields its own average and maximum error values. We then run the algorithm on each sequence and report the average runtime and error across the 10 runs. In the plots, η_{avg} is the mean of the 10 average-error values and $\eta_{\max}$ is the mean of the 10 maximum-error values.

Discussion. We include the plot for `sx-askubuntu` in Figure 2, and include the results for the remaining datasets in the full version of the paper. With sufficiently accurate predictions, our algorithm gives significantly improved performance over both versions of IncSCC$^+$. The

[4] `https://snap.stanford.edu/data/`

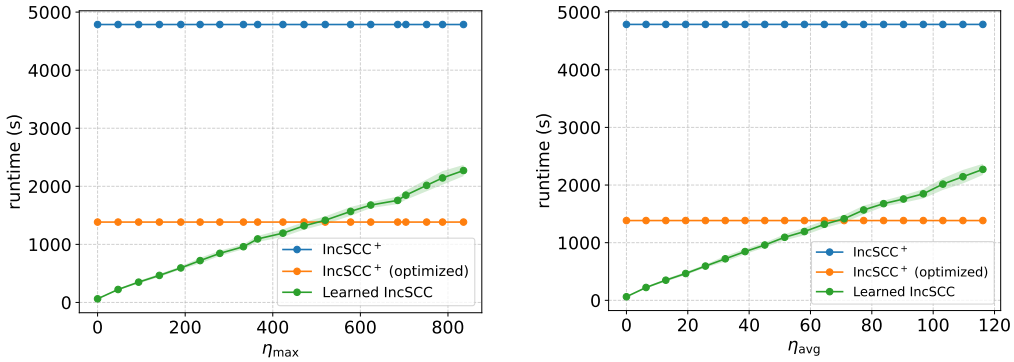


Figure 2 Runtime comparison on the `sx-askubuntu` dataset. In the left and right plots, the x-axis denotes the maximum and average prediction error with respect to the actual edge sequence, respectively. The green line shows the mean runtime, and the surrounding shaded region indicates the standard deviation.

Table 1 Comparison of the performance of Learned IncSCC against the baselines. The first two columns show the number of unique edges and nodes. The next four columns report the runtimes of the purely offline algorithm, Learned IncSCC with perfect predictions, and the two versions of IncSCC$^+$. Crossover η_{avg} and crossover $\eta_{\max}$ are the largest average and max errors for which Learned IncSCC outperforms IncSCC$^+$ (optimized).

Dataset	m	n	Offline	Learned IncSCC $\eta = 0$	IncSCC$^+$	IncSCC$^+$ (optimized)	Crossover η_{avg}	Crossover $\eta_{\max}$
`sx-superuser-c2a`	289487	101052	60.70	34.21	932.10	331.33	≈ 32	≈ 219
`sx-askubuntu`	596933	159316	85.37	62.24	4786.33	1384.01	≈ 70	≈ 517
`Slashdot0811`	905468	77360	71.33	47.92	2221.04	362.68	≈ 20	≈ 150

experiments show our performance grows roughly linearly with the error; in fact it grows linearly with both the η_{avg} and $\eta_{\max}$ (our theoretical analysis bounds performance only in terms of $\eta_{\max}$).

Table 1 summarizes further results, including the largest average and maximum η for which our algorithm improves on optimized IncSCC$^+$.

In Table 1, we also compare our algorithm with perfect predictions to the offline incremental algorithm from Section 2.2. While both algorithms are given the entire edge sequence, our algorithm can adapt to errors on-the-fly, while the offline algorithm is static. Despite this overhead, our running time is not only comparable but in fact slightly better than the offline algorithm. This demonstrates that our method is lightweight, while still allowing for potential error.

6 Conclusion

In this work, we design a new learned algorithm to maintain the strongly connected components in an incremental graph. Our theoretical bounds show that our algorithm is nearly optimal under reasonably good predictions and its performance degrades smoothly with respect to the prediction error.

We test our algorithm against the best heuristic on real-world datasets. Our experiments show that (1) theory is predictive of practice, and (2) it is possible to achieve strong provable guarantees against bad predictions, while also being able to match (and even beat) the practical performance of state-of-the-art heuristics when predictions are good.

References

1. Xingjian Bai and Christian Coester. Sorting with predictions. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL: https://openreview.net/forum?id=Qv7rWR9JWa.
2. Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. *Algorithmica*, 81(3):967–985, 2019. doi:10.1007/S00453-018-0452-3.
3. Michael A Bender, Jeremy T Fineman, Seth Gilbert, and Robert E Tarjan. A new approach to incremental cycle detection and related problems. *ACM Transactions on Algorithms (TALG)*, 12(2):1–22, 2015. doi:10.1145/2756553.
4. Ziyad Benomar and Christian Coester. Learning-augmented priority queues. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=1ATLLgvURu.
5. Aaron Bernstein, Aditi Dudeja, and Seth Pettie. Incremental scc maintenance in sparse graphs. In *29th Annual European Symposium on Algorithms (ESA 2021)*, 2021.
6. Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on theory of computing*, pages 365–376, 2019. doi:10.1145/3313276.3316335.
7. Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, st shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1165–1173, 2024. doi:10.1145/3618260.3649745.
8. Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. Predictive flows for faster ford-fulkerson. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proc. of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 7231–7248. PMLR, 23–29 July 2023. URL: https://proceedings.mlr.press/v202/davies23b.html.
9. Swati Dhingra, Poorvi S Dodwad, and Meghna Madan. Finding strongly connected components in a social network graph. *International Journal of Computer Applications*, 136(7):1–5, 2016.
10. Michael Dinitz, Sungjin Im, Thomas Lavastida, Ben Moseley, Aidin Niaparast, and Sergei Vassilvitskii. Binary search with distributional predictions. *Advances in Neural Information Processing Systems*, 37:90456–90472, 2024.
11. Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Proc. 34th Conference on Advances in Neural Information Processing Systems (NeurIPS)*, pages 10393–10406, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html.
12. Wenfei Fan, Chunming Hu, and Chao Tian. Incremental graph computations: Doable and undoable. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 155–169, 2017. doi:10.1145/3035918.3035944.
13. Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert E Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Transactions on Algorithms (TALG)*, 8(1):1–33, 2012. doi:10.1145/2071379.2071382.
14. Benjamin Charles Hardekopf. *Pointer analysis: building a foundation for effective program analysis*. The University of Texas at Austin, 2009.
15. Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. On the complexity of algorithms with predictions for dynamic graph problems. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 62:1–62:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ITCS.2024.62.

16 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proc. 44th Annual International Conference on Management of Data, (SIGMOD)*, pages 489–504. ACM, 2018. doi:10.1145/3183713.3196909.

17 Silvio Lattanzi, Ola Svensson, and Sergei Vassilvitskii. Speeding up bellman ford via minimum violation permutations. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 18584–18598. PMLR, 2023. URL: https://proceedings.mlr.press/v202/lattanzi23a.html.

18 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

19 Honghao Lin, Tian Luo, and David Woodruff. Learning augmented binary search trees. In *International Conference on Machine Learning*, pages 13431–13440. PMLR, 2022. URL: https://proceedings.mlr.press/v162/lin22f.html.

20 Quanquan C. Liu and Vaidehi Srinivas. The predicted-updates dynamic model: Offline, incremental, and decremental to fully dynamic transformations. In Shipra Agrawal and Aaron Roth, editors, *Proceedings of Thirty Seventh Conference on Learning Theory*, volume 247 of *Proceedings of Machine Learning Research*, pages 3582–3641. PMLR, 30 June–03 July 2024. URL: https://proceedings.mlr.press/v247/liu24c.html.

21 Samuel McCauley, Benjamin Moseley, Aidin Niaparast, Helia Niaparast, and Shikha Singh. Incremental approximate single-source shortest paths with predictions. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 117:1–117:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2025.117.

22 Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. Online list labeling with predictions. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Proc. 36th Conference on Neural Information Processing Systems (NeurIPS)*, volume 36, pages 60278–60290. Curran Associates, Inc., 2023.

23 Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. Incremental topological ordering and cycle detection with predictions. In *Forty-first International Conference on Machine Learning*, 2024. URL: https://openreview.net/forum?id=wea7nsJdMc.

24 Samuel McCauley, Benjamin Moseley, Helia Niaparast, and Shikha Singh. Stable matching with predictions: Robustness and efficiency under pruned preferences. *arXiv preprint*, 2026. arXiv:2602.02254.

25 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM (CACM)*, 65(7):33–35, 2022. doi:10.1145/3528087.

26 Helia Niaparast, Benjamin Moseley, and Karan Singh. Faster global minimum cut with predictions. In *Forty-second International Conference on Machine Learning*, 2025. URL: https://openreview.net/forum?id=FeZoO7mfG7.

27 Adam Polak and Maksym Zub. Learning-augmented maximum flow. *Information Processing Letters*, 186:106487, 2024. doi:10.1016/j.ipl.2024.106487.

28 Mateusz Radecki. My own algorithm — offline incremental strongly connected components in o(m log(m)). https://codeforces.com/blog/entry/91608, 2021. Accessed: 2025-09-30.

29 Liam Roditty. Decremental maintenance of strongly connected components. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1143–1150. SIAM, 2013. doi:10.1137/1.9781611973105.82.

30 Shinsaku Sakaue and Taihei Oki. Discrete-convex-analysis-based framework for warm-starting algorithms with predictions. In *35th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

31 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

32 Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3534–3557. SIAM, 2024. doi:10.1137/1.9781611977912.126.

33 Ali Zeynali, Shahin Kamali, and Mohammad Hajiesmaili. Robust learning-augmented dictionaries. In *Forty-first International Conference on Machine Learning*, 2024. URL: https://openreview.net/forum?id=XyhgssAo5b.

34 Yu Zhang, Xiaofei Liao, Xiang Shi, Hai Jin, and Bingsheng He. Efficient disk-based directed graph processing: A strongly connected component approach. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):830–842, 2017. doi:10.1109/TPDS.2017.2776115.

On Fréchet Traveling Salesmen Problems

Omrit Filtser
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Tzalik Maimon
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Michal Moiseev
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Abstract

The Fréchet distance is a well-studied distance measure between two curves. In this work, we demonstrate that the merit of Fréchet distance extends beyond evaluating similarity, and introduce a new setting in which it proves useful. Consider a situation where two agents are required to visit a given set of sites, while staying close to each other throughout their traversal. In this paper, we study problems where the goal is to construct two curves whose vertices are from a given set of points, under the constraint that the Fréchet distance between the curves is kept as small as possible. This problem can be viewed as a variant of the Traveling Salesman Problem (TSP), and thus may be of interest in routing, network planning and more. We present a near-linear algorithm for this problem under the discrete Fréchet distance, and explore several variants of the problem, including minimizing the lengths of the curves and balancing the number of sites assigned to each agent. Lastly, we prove that the problem is NP-hard under the continuous Fréchet Distance.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, traveling salesman problem

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.18

Related Version *Full Version*: https://arxiv.org/abs/2606.01147

Funding This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 2135/24).

Acknowledgements We would like to thank an anonymous reviewer for helpful comments and references, and in particular for suggesting the expected linear time algorithm in Section 3.

1 Introduction

The traveling salesman problem (TSP) asks for the shortest route that visits a given set of points. It is a classic NP-hard problem, and have various approximation algorithms (including a PTAS in the Euclidean plane [4, 23]). In some generalizations of TSP, e.g. the Vehicle Routing Problem or Multiple-TSP (see, e.g. [5, 7, 12, 24]), the goal is to plan short routes for a group of agents to visit all the points together. In these variants there is no restriction on the relations between the routes.

Consider a situation where two agents are asked to visit a given set of sites, they can split but they still need to remain close to each other throughout their motion. This can be required for example in order to keep then in some range of communication, or so that one agent can quickly get to other in case of trouble. Therefore, in this paper, we introduce and study a new set of problems where the goal is to plan routes for two (or more) agents, that together visit a given set of sites, while keeping the agents close to each other throughout the traversal. More precisely, our goal is to construct two curves from a given set of points, such that the (continuous or discrete) Fréchet distance between the curves is as small as possible. We also consider variations of this Fréchet-TSP type problem where the goal is to balance the load on the agents, in different ways.

© Omrit Filtser, Tzalik Maimon, and Michal Moiseev;
licensed under Creative Commons License CC-BY 4.0

The Fréchet distance [18], introduced by *Maurice Fréchet*, is a popular measure of similarity between curves. It is often described by an analogy of a person and a dog connected by a leash, both walking forward along two separate curves, while the leash keeps them at a bounded distance. The Fréchet distance is then the shortest length of a leash that allows them to traverse their respective curves. The Fréchet distance takes into account both the position and ordering of points, distinguishing it from other metrics like the Hausdorff distance. This property makes the Fréchet distance useful across a wide range of applications.

The discrete Fréchet distance focuses solely on the points of the curves rather than the edges between them. This variant can be described analogously by replacing the man and dog by two frogs that are "hopping" along the vertices of the curves while attempting to maintain a small distance between them. This discrete approach allows simpler and (slightly) faster algorithms for computing the distance.

Alt and Godau [3] showed that the Fréchet distance between a curve P of length n and a curve Q of length m, can be calculated in $O(nm \log(nm))$ time. Eiter and Mannila [16] showed that the discrete Fréchet distance can be computed in $O(mn)$ time. Only 20 years later, it has been shown that under the Strong Exponential Time Hypothesis (SETH), the Fréchet distance cannot be computed in strongly subquadratic time [9]. Very recently, Cheng, Huang, and Zhang [13], presented a strong subquadratic time algorithm that computes a $(7+\varepsilon)$-approximation for both the discrete and continuous distance.

A plethora of applications and variants of the Fréchet distance have been studied since then (see, e.g. [10, 14, 15, 21]). The variant most relevant to our work is the Fréchet distance between two point sets, recently studied by Buchin and Kilgus [11]. In this variant, the input objects are two sets of points rather than two curves, and the goal is to construct two curves (one for each set of points), to minimize the Fréchet distance between them. The main difference from our version is that we also need to find a partition of the point set into two sets. Buchin and Kilgus showed that under the discrete Fréchet distance, the problem is equivalent to computing the Hausdorff distance between the two sets, and thus can be solved in $O(nm \log(nm))$ time (where n, m are the sizes of the two sets). The continuous version, on the other hand, was shown to be NP-complete. They also provide an exponential time algorithm running in $O(k^a((m+n-a)+a\log a))$ where a denotes the number of points that can be matched only to edges ("floating" points in their terminology) and k is the maximum number of edges that can cover such a point.

Another closely related variant is the Curve-Point-Set Matching Problem (CPSM), where given a polygonal curve P, a set of points S, and a maximum distance $\delta > 0$, the objective is to find another polygonal curve Q, whose set of vertices is either a subset of S or contains all the points of S (also, either with or without repetitions, referred to as the non-unique/unique variant, respectively), such that the continuous or discrete Fréchet distance between the new curve Q and the original curve P is smaller than δ. Maheshwari, Arora, and Smid [22] addressed the continuous non-unique subset version of the problem, and presented an algorithm that runs in $O(nk^2)$ time. Wylie [26] studied CPSM under the discrete Fréchet distance, and proved that the unique version (where each point in S can appear only once in Q) is NP-complete for both the subset and all-point versions. This is in contrast to the non-unique cases which were shown to be polynomial-time solvable (in $O(nk)$ time, where n is the size of S and k the size of P). Accisano and Ungor [1] also showed NP-completeness under the continuous distance for the all-points variant, both in unique and non-unique settings.

Overview. To the best of our knowledge, the Fréchet-TSP problem described above has not been studied before. This gives rise to a rich family of TSP-style questions centered around the Fréchet distance, and we systematically outline several natural variants that capture different aspects of this setting. In Section 3, we present an algorithm for the discrete Fréchet-TSP problem that runs in $O(n \log n)$ time for a set of n points in a constant dimension. We utilize a combination of the properties of Nearest-Neighbor-Graph and Minimum-Spanning Tree (MST) over a Unit-Disk Graph (UDG). By utilizing the properties of UDG, we ensure an upper bound on the distance between the partitions we output. By utilizing the properties of the MST, we obtain a linear time partitioning. The initialization of these constructs are obtained in $O(n \log n)$ time. In Section 4, we consider a variant of the problem aiming to minimize the total length of the curves, and present a constant-factor approximation algorithm that also runs in $O(n \log n)$ time. In Section 5, we focus on balancing the number of vertices among the partitioned components, and present different strategies to achieve almost optimal balance. Here, we utilize the properties of the UDG again. Specifically, the upper bound on the number of kissing number of the graph. This in turn limits the number of cases in the possible partition we wish to balance. We handle these case-by-case showing that we can get the balance as close as at most 1 from optimal. Finally, in Section 6, we show that the continuous variant of Fréchet-TSP is NP-hard. Out proof is an adaptation of the method used by Buchin and Kilgus [11].

2 Notations and problem definition

A polygonal curve P in $\mathbb{R}^d$ is a continuous function $P : [1, n] \to \mathbb{R}^d$, such that for any integer $1 \leq i \leq n-1$ the restriction of P to the interval $[i, i+1]$ is a straight line segment. The points $P[0], \ldots, P[n]$ are the vertices of P, and the segments $\overline{P[i]P[i+1]}$ are the edges of P.

Let $P : [0, n] \to \mathbb{R}^d$ and $Q : [0, m] \to \mathbb{R}^d$ be two polygonal curves of with number of vertices n and m, respectively.

The (continuous) Fréchet distance. A reparameterization of a curve P is a continuous, non-decreasing, surjective function $f : [0, 1] \to [1, n]$ such that $f(0) = 1$ and $f(1) = n$. The Fréchet distance between P and Q is defined as $d_F(P, Q) = \inf_{f,g} \max_{t \in [0,1]} \|P(f(t)) - Q(g(t)\|$, where f is a reparameterization of P and g is a reparameterization of Q.

The discrete Fréchet distance. [1] A ***paired-walk*** along P and Q is a sequence of pairs $\pi = \{(P_i, Q_i)\}_{i=1}^{k}$, such that $P_1, ..., P_k$ and $Q_1, ..., Q_k$ partition P and Q, respectively, into (disjoint) non-empty subsequences of their vertices, and for any i it holds that either $|P_i| = 1$ or $|Q_i| = 1$. The ***cost*** of a paired walk W along P and Q is

$$\text{cost}(\pi) = \max_i \max_{(p,q) \in P_i \times Q_i} \|p - q\|.$$

For any pair $(p, q) \in P_i \times Q_i$, we say that p is ***matched*** to q in π.

The ***discrete Fréchet distance*** between P and Q is $\mathrm{d}_{dF}(P, Q) = \min_{\pi \in \Pi} \text{cost}(\pi)$, where Π is the set of all possible paired-walks along P and Q.

[1] For the simplicity of presentation, we follow the definition given in [6], which is equivalent to the original definition given in [16].

Partitioning a point set into curves. Let δ be a Fréchet-based distance measure for two curves in $\mathbb{R}^d$ (i.e., either the continuous or discrete Fréchet distance). Given a set S of points in $\mathbb{R}^d$, we say that two curves P, Q ***partition*** S if there is a partition of S into two sets A and B such that the set of vertices of P is exactly A and the set of vertices of Q is exactly B, and each point in S is used exactly once (either in P or in Q). The basic version of the problem that we wish to consider is the following.

▶ **Problem 1** (Fréchet-TSP). *Given a set S of points in $\mathbb{R}^d$, find two curves P, Q that partition S and such that $\delta(P, Q)$ is minimized.*

Note that if we do not require each point in S to be visited by exactly one of the agents, then the problem becomes trivial: we can pick any order on the points in S and set $P = Q$, so the distance between the paths is 0. In the full version of the paper, we discuss a variant in which we allow a point to be used more than once by the same agent, and show that this variant is equivalent to Problem 1 under the discrete Fréchet distance.

Denote by ε^* the distance between the curves in an optimal solution for the Fréchet-TSP problem. Clearly, there may be many different optimal solutions, all having distance ε^*. However, some solutions may be considered better than others, for example, if each curve covers roughly the same number of points from S (balanced partition – see Figure 1), or if the min-max length (or sum of lengths) of the curves is very small in comparison to other solutions (see Figure 2). We therefore define below different variants of the problem in which the goal is to find a "good" solution among those that achieve the optimal distance ε^*.

Figure 1 Left: a balanced partition, right: an imbalanced partition. Both have the same distance.

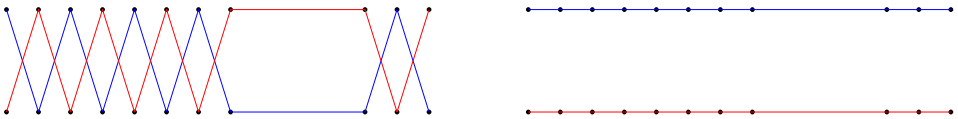


Figure 2 Left: two long curves, right: two shorter curves. Both have the same distance.

▶ **Problem 2** (balanced-Fréchet-TSP). *Given a set S of n points in $\mathbb{R}^d$, find two curves P, Q that partition S, such that $\delta(P, Q) = \varepsilon^*$, and $\max\{|P|, |Q|\}$ is minimized.*

For a curve P, denote by $\ell(P)$ the sum of the lengths of the edges of P.

▶ **Problem 3** (min-max-Fréchet-TSP). *Given a set S of points in $\mathbb{R}^d$, find two curves P, Q that partition S, such that $\delta(P, Q) = \varepsilon^*$, and $\max\{\ell(P), \ell(Q)\}$ is minimized.*

▶ **Problem 4** (min-sum-Fréchet-TSP). *Given a set S of points in $\mathbb{R}^d$, find two curves P, Q that partition S, such that $\delta(P, Q) = \varepsilon^*$, and $\ell(P) + \ell(Q)$ is minimized.*

Minimizing the length of the path is NP-hard, similar to the traveling salesman problem (TSP), which is NP-hard: for a reduction, simply double each point in an instance of TSP, leading to the following theorem:

▶ **Corollary 1.** *min-max-Fréchet-TSP and min-sum-Fréchet-TSP are NP-hard.*

We can thus aim to find an approximation algorithm for minimizing the length.

3 Discrete Fréchet-TSP

In this section, we focus on the discrete Fréchet distance (i.e. $\delta = \mathrm{d}_{dF}$). We begin by presenting an algorithm for the decision version of the problem: Given a set S of n points in $\mathbb{R}^d$, and a threshold $\varepsilon \geq 0$, decide whether there exist two curves P and Q that partition S and have $\mathrm{d}_{dF}(P,Q) \leq \varepsilon$.

Let $G_\varepsilon = (S, E)$ be the graph whose vertices are the points in S, and there is an edge $\{v, u\} \in E$ if and only if $\|v - u\| \leq \varepsilon$. By definition, G_ε is a unit-disk graph with radius ε. Let P, Q be two curves that partition S s.t. the edges in $P \cup Q$ are edges from G_ε. Any paired walk π along P and Q with $\mathrm{cost}(\pi) \leq \varepsilon$ can be reduced to a set of disjoint stars in G_ε simply by removing edges from π. Therefore, if there is such a walk π, then there is such a set of disjoint starts and vice versa. Thus, we can search for such a set.

▶ **Lemma 2.** *There exist two curves P, Q that partition S and have $d_{dF}(P,Q) \leq \varepsilon$ if and only if G_ε does not contain a vertex of degree 0.*

Proof. If G_ε contains a vertex v of degree 0, then there is no $u \in S$ with $\|v - u\| \leq \varepsilon$. Therefore, v cannot be matched to any other point in S in a paired-walk of cost at most ε. For the other direction, let C be a connected component of G_ε. We now show that if C contains more than one vertex, then we can construct two curves on the set of vertices of C as required. This finishes the proof, as we can concatenate the curves that were constructed for all the connected components, and get two curves with vertices from S such that each point in S is used exactly once. Let T_C be some spanning tree of C. We color the vertices of T_C red and blue, as follows: first, color all the leaves in red, and then, color the parents of all those leaves in blue. This coloring defines a set of stars, each has a blue center node and at least one red node. By removing these stars from $T_C = T_0$, we are left with a smaller tree, T_1. If T_1 is a single vertex, then color it red and add it to one of the stars of its child nodes. Otherwise, T_1 contains at least one edge. We then repeat the process on T_1 and obtain another set of stars, remove them from T_1 and get a smaller tree T_2. We continue this process until all the vertices are colored. This process results in a partition of the nodes of T_C into stars $S_1, \ldots, S_k$, each star contains exactly one blue node $\mathbf{blue}(S_i)$, and a non empty sequence of red nodes $\mathbf{red}(S_i)$. Let $P = \{\mathbf{blue}(S_1), \ldots, \mathbf{blue}(S_k)\}$ and $Q = \{\mathbf{red}(S_1), \ldots, \mathbf{red}(S_k)\}$. Since for each $i \in [k]$ we have $\|\mathbf{blue}(S_i) - v\| \leq \varepsilon$ for every $v \in \mathbf{red}(S_i)$, the sequence of stars corresponds to a paired-walk with cost at most ε, and therefore we get $\mathrm{d}_{dF}(P,Q) \leq \varepsilon$. ◀

Note that Lemma 2 provides an algorithm for computing a partition of S into two curves P, Q such that $\mathrm{d}_{dF}(P,Q) \leq \varepsilon$. Given a spanning forest of G_ε, the running time for constructing P, Q is $O(n)$, since each tree can be colored using a a simple BFS traversal. We conclude this in the following corollary.

▶ **Corollary 3.** *Given a spanning forest T of G_ε such that no vertex in T has degree 0, a partition P, Q of S with $d_{dF}(P,Q) \leq \varepsilon$ can be constructed in $O(n)$ time.*

We wish to find the partition P, Q of S that minimizes $\mathrm{d}_{dF}(P,Q)$. Denote by ε^* the distance between the curves in an optimal partition, i.e., there exists a partition P^*, Q^* of S with $\mathrm{d}_{dF}(P^*, Q^*) = \varepsilon^*$, and for any partition P, Q of S it holds that $\mathrm{d}_{dF}(P,Q) \geq \varepsilon^*$. In Claim 4 below, we show that ε^* is the furthest nearest neighbor distance, i.e., the length of the longest edge in the Nearest Neighbor Graph of S.

Denote by $\mathrm{NNG}(S)$ the Nearest Neighbor Graph (NNG) of S, i.e., the graph whose vertices are the points of S, and there is an edge $\{u, v\}$ is the graph if and only if v is a nearest neighbor of u in S. Note that a point u can have more than a single nearest neighbor.

In this case, we break ties by taking the point with the largest index to be the unique nearest neighbor. It is well-known that when applying such a tie-breaking rule, the NNG is a forest, and a subgraph of the Euclidean minimum spanning tree.

▷ Claim 4. Let L be the longest edge in $\mathrm{NNG}(S)$. Then $\varepsilon^* = L$.

Proof. Notice that $\mathrm{NNG}(S)$ is a spanning forest of G_L that does not contain any vertex of degree 0 (every point in S has a unique nearest neighbor). Therefore, by Corollary 3 we get that there exists a partition P, Q of S with $\mathrm{d}_{dF}(P, Q) \leq L$.

Assume by contradiction that there is a partition P', Q' of S with $\mathrm{d}_{dF}(P', Q') = \varepsilon^* < L$, and consider a paired-walk π along P and Q with cost ε^*. Then for any pair of points $v, u \in S$ that are matched in π, we have $\|v - u\| < L$. Let $\{w, x\}$ be the longest edge in $\mathrm{NNG}(s)$, so $\|w - x\| = L$, and w, x are not matched in π. Therefore, there exists $w', x' \in S$ such that w, w' are matched in π and x, x' are matched in π. We get that $\|w - w'\| < L$, so x is not a nearest neighbor of w, and $\|x - x'\| < L$, so w is not a nearest neighbor of x, a contradiction to $\{w, x\}$ being an edge of $\mathrm{NNG}(s)$. ◁

By Claim 4, all the edges of $\mathrm{NNG}(S)$ have length smaller or equal to the optimal distance ε^*, and therefore it is a spanning forest of G_{ε^*}. Thus, given $\mathrm{NNG}(S)$ as an input, the algorithm from Corollary 3 runs in $O(n)$ time. For $d = 2$, computing $\mathrm{NNG}(S)$ can be dome in $O(n \log n)$ time [17]. For general dimension d, computing $\mathrm{NNG}(S)$ can be done in $O(2^{O(d)} n \log n)$ time [25], so the overall running time for our problem is $O(n \log n)$ for any fixed dimension d.

We, therefore, obtain the following theorem.

▶ **Theorem 5.** *Given a set S of n points in $\mathbb{R}^d$, for any fixed dimension d, one can find two curves P, Q that partition S such that $d_{dF}(P, Q)$ is minimized in $O(n \log n)$ time.*

In the two remarks below, we suggest other ways to use Claim 4 and Corollary 3 for computing an optimal partition of S.

▶ Remark 6 (Using the Net and Prune framework). Har-Peled and Raichel [19] show that the furthest nearest neighbor distance (which, by Claim 4, equals ε^*) can be computed in expected linear time for points in any dimension d. Then, to apply Corollary 3, we need to compute a spanning forest T of G_ε such that no vertex in T has degree 0. For this we can use the Net and Prune framework of [19] as follows. Put S in a grid with cell diameter ε^*. Points that belong to the same grid cell form a clique in G_ε, and can be connected, for example, by some star graph. The lonely points, i.e., points that are alone in their cell, can be connected to a point in a neighbor cell (such a point must exist by the way we chose ε^*). Since the number of neighbor cells is $2^{O(d)}$, this results in an $O(2^d \cdot n)$ time algorithm for computing the spanning forest, and $O(2^{O(d)} \cdot n)$ expected time for computing the partition.

▶ Remark 7 (Using the MST). For points in dimension $d > 2$, we can use the the Minimum Spanning Tree of S ($\mathrm{MST}(S)$) instead of $\mathrm{NNG}(S)$ as follows. We compute $\mathrm{MST}(S)$, and then iterate the edges from the longest to shortest. If removing an edge from $\mathrm{MST}(S)$ does not create a vertex of degree 0, remove it, and otherwise stop. Since $\mathrm{NNG}(S) \subseteq \mathrm{MST}(S)$, and because in $\mathrm{NNG}(S)$ there is no vertex of degree 0, the resulting graph contains $\mathrm{NNG}(S)$ and the length of its edges is at most ε^*. We can then execute the algorithm from Corollary 3 on this residue graph. Since $\mathrm{MST}(S)$ can be computed in $O(n^{2-2/\lceil d/2 \rceil + 1} + \varepsilon)$ time [2], this is also the total running time of the algorithm. Notice that for $d \leq \log \frac{n^k}{\log n}$, where $0 < k < 1 - 1/O(\log \frac{n}{\log n})$, we prefer to run the algorithm that finds the residue graph over computing $\mathrm{MST}(S)$. This is because the running time for finding the residue graph is $O(n^{2-2/\lceil k \cdot \log \frac{n}{\log n} / 2 \rceil + 1} + \varepsilon)$ while computing $\mathrm{NNG}(S)$ is in $O(n^{1+k})$.

4 Minimizing the lengths

In this section, we focus on problems 3 and 4. Let S be a set of $n > 2$ points in the plane. Our goal is to find a partition P, Q with $\mathrm{d}_{dF}(P, Q) = \varepsilon^*$. In problem 3, the partition also minimizes $\max\{\ell(P), \ell(Q)\}$. In problem 4 it also minimizes $\ell(P) + \ell(Q)$.

Let $\mathrm{TSP}(S)$ be a path on S of minimum length. Denote by $\ell(G)$ the sum of edge lengths of a graph G embedded in the plane.

4.1 Min-max discrete Fréchet-TSP

In this section, we prove the following theorem.

▶ **Theorem 8.** *Given a set S of n points in the plane, two curves P, Q that partition S such that $d_{dF}(P, Q) = \varepsilon^*$ and $\max\{\ell(P), \ell(Q)\} \leq 2.75 \cdot \ell(\mathrm{TSP}(S))$ can be found in $O(n \log n)$ time.*

By Theorem 5, computing the value ε^* can be done in $O(n \log n)$ time. Let $\mathrm{MST}(S)$ be a minimum spanning tree of S, and observe that $\ell(\mathrm{MST}(S)) \leq \ell(\mathrm{TSP}(S))$. We thus focus on computing two curves P, Q that partition S such that $\mathrm{d}_{dF}(P, Q) = \varepsilon^*$ and $\max\{\ell(P), \ell(Q)\} \leq 2.75 \cdot \ell(\mathrm{MST}(S))$. For simplicity, we say that two points (or vertices) u, v are ***close*** if $\|u - v\| \leq \varepsilon^*$, and otherwise we say that u, v are ***far***.

▶ **Observation 9.** *Every vertex in $\mathrm{MST}(S)$ has at least one neighbor in $\mathrm{MST}(S)$ that is close to it.*

Proof. Assume by contradiction that all the neighbors of a vertex v in $\mathrm{MST}(S)$ are far from v. By Lemma 2, v has a neighbor u in G_{ε^*}, and therefore u is close to v. This contradicts the fact that the MST contains a nearest neighbor for each vertex (by Kruskal's algorithm). ◀

We now show how to construct two curves P and Q that partition S by traversing $\mathrm{MST}(S)$ in a DFS order, starting from a leaf vertex of $\mathrm{MST}(S)$ as a root, and coloring the vertices **red** and **blue**. The blue vertices will be in P, and the red vertices in Q. The order of the points along the curves would be the same order in which they where colored during the algorithm.

Consider a vertex v in the rooted tree $\mathrm{MST}(S)$. Let $I_{\mathrm{close}}(v)$ be the set of children of v that are close to v, and similarly, $I_{\mathrm{far}}(v)$ will be the set of children of v that are far from v. In addition, let $L(v) = \{u \mid u \in I_{\mathrm{close}}(v), I_{\mathrm{close}}(u) = \emptyset\}$. In other words, $L(v)$ is the set of ***lonely children*** of v – those that are either leaves in $\mathrm{MST}(S)$, or that do not have close children – and therefore must be in v's star. Notice that by Observation 9, all the nodes in $L(v)$ are close to v.

Let v be the current vertex visited by the DFS algorithm. The invariant of our recursive DFS algorithm is that if $L(v)$ is empty, then there must be at least one vertex in $I_{\mathrm{close}}(v)$. The algorithm has two main steps: in the first step we color vertices in red and blue, and the second step contains the recursive calls.

Coloring. In this step, we color the entire star that v belongs to using Algorithm 1. Note that v can be either a center or a leaf in that star. Also, Algorithm 1 colors exactly one vertex in **blue** and the rest in **red**. We progress in recursion on the vertices of the tree. In each recursive call, we activate the coloring algorithm on newly colored vertices in the following order:

1. First, activate the algorithm on children of vertices that are colored **red**, in reversed order (last colored first called).
2. then, activate the algorithm on children of the **blue** vertex.

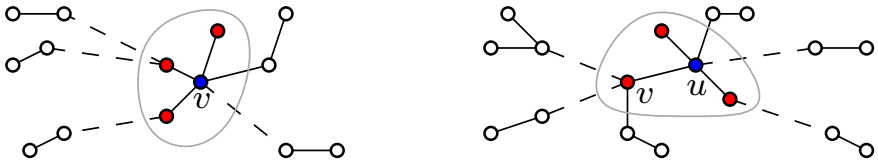


Figure 3 Coloring of MST(S). Dashed edges correspond to far neighbors. The star to which v belongs is marked in gray.

Algorithm 1 Coloring for Min-Length.

Input: Vertex v
Output: Color updates
if $L(v) \neq \emptyset$ **then**
  color the leaf nodes in $L(v)$ **red**;
  color all the other nodes in $L(v)$ **red**;
else
  Let u be an arbitrary vertex in $I_{\text{close}}(v)$;
  color v **red**;
  color u **blue**;
  color the leaf nodes in $L(u)$ **red**;
  color all the other nodes in $L(u)$ **red**;

First, notice that when the algorithm is called with a vertex v in a recursive step, then v and its entire subtree were not colored yet. Moreover, we show that the following invariant holds in each step of the algorithm.

▷ Claim 10. In each step of the algorithm, if $L(v)$ is empty, then there must be at least one vertex in $I_{close}(v)$.

Proof. Assume by contradiction that in some step of the algorithm both $L(v)$ and $I_{\text{close}}(v)$ are empty. Then, by Observation 9, v must have a parent t in MST(S) such that $v \in I_{\text{close}}(t)$, and t was already colored by the algorithm. Moreover, v is in $L(t)$, because $I_{\text{close}}(v) = \emptyset$. Therefore, it is not possible that t was colored **blue**, because then v would have been already colored **red** and would not be called recursively. In addition, t is not a lonely child of its parent, because $I_{\text{close}}(t) \neq \emptyset$. Therefore, if t was colored **red**, then it must be in step 2(a) of the algorithm, which means that $L(t)$ is empty, but this is not possible because $v \in L(t)$. ◁

Let $P = \{p_1, \ldots p_k\}$ (resp. $Q = \{q_1, \ldots q_m\}$) be the set of vertices that were colored **blue** (resp. **red**), in the order in which they were colored during the DFS scan.

▷ Claim 11. The running time for computing P and Q is $O(n \log n)$.

Proof. Calculating the Euclidean MST for the set S takes $O(n \log n)$ time. A standard DFS traversal over the MST also requires $O(n)$ time. The additional overhead in the algorithm comes from recursively scanning the $L(v)$ children of each vertex v. Since $|L(v)| \leq 5$ in the EMST, this adds at most $O(n)$ additional operations. Thus, the total running time remains $O(n \log n)$. ◁

▷ Claim 12. $\mathrm{d}_{dF}(P, Q) \leq \varepsilon^*$.

Proof. We show that in each step of the algorithm, we color a star in $\mathrm{MST}(S)$ with edges of length at most ε^*. The center is colored **blue**, and the leaves **red**. Let v be the current vertex. There are two cases:

- If $L(v)$ is not empty, then v is colored **blue** and the nodes in $L(v)$ are colored **red**. Since $L(v) \subseteq I_{\mathrm{close}}(v)$ (by Observation 9), this red-blue star has edges of length at most ε^*.
- If $L(v) = \emptyset$, then the algorithm picks a vertex $u \in I_{\mathrm{close}}(v)$ and color it **blue**. This vertex becomes the center of a star with its children $v \cup L(u)$ which are colored **red**. Since $L(v) \subseteq I_{\mathrm{close}}(u)$, we again obtain a red-blue star with edges of length at most ε^*.

Since the vertices of P and Q are ordered by the time they were colored, we get that the sequence of starts corresponds to a paired-walk of cost at most ε^* between P and Q, as required. ◁

Next, we bound the lengths of the curves P and Q in relation to $\mathrm{MST}(S)$. Notice that the order in which we color the **blue** vertices (the vertices of P) follows a classic DFS preordering, and therefore we clearly have $\ell(P) \leq 2 \cdot \ell(\mathrm{MST}(S))$. However, the order in which we color the **red** vertices slightly differs from a classic DFS preordering. However, since all the children of v that are colored in this step are close to v, the additional traversal overhead is small. The following claim together with Claim 12 implies Theorem 8.

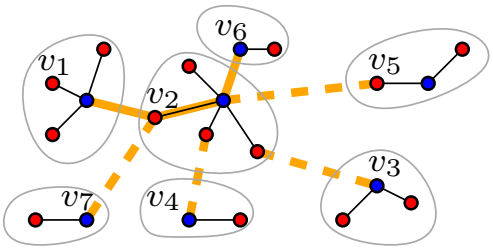


Figure 4 An illustration of the algorithm. The root vertex is v_1, and the algorithm runs on $v_1, \ldots, v_7$ in this order. The stars are marked in gray, and the dashed edges mark far vertices. The orange edges are the set W of edges that connect the stars.

▷ Claim 13. $\max\{\ell(P), \ell(Q)\} \leq 2.75 \cdot \ell(\mathrm{MST}(S))$.

Proof. Let T be the path on all the points in S, which is obtained by traversing the edges of $\mathrm{MST}(S)$ in the order in which the algorithm colors the vertices, regardless of their color. Clearly $\ell(T) \geq \max\{\ell(P), \ell(Q)\}$. We show that $\ell(T) \leq 2.75 \cdot \ell(\mathrm{MST}(S))$.

Recall that the degree of any vertex in $\mathrm{MST}(S)$ is at most 5. Since we choose the root of $\mathrm{MST}(S)$ to be a leaf, then for every vertex v in the rooted tree, $L(v)$ may contain at most 4 vertices. Consider a step of the algorithm were the current vertex is v. If v was colored **blue**, denote $L(v) = u_1, \ldots, u_k$ for $1 \leq k \leq 4$. Then, the subpath of T that was added in this step is $T_v = \{v, u_1, v, u_2, v, \ldots, u_k\}$. This is because, in T, we are moving from u_i to u_{i+1} through v, since $\{u_i, u_{i+1}\}$ is not an edge in $\mathrm{MST}(S)$. If v was colored **red**, the algorithm picked a vertex $u \in I_{close}(v)$. Denote $L(u) = u_1, \ldots, u_k$ for $1 \leq k \leq 4$. The subpath of T that was added in this step is $T_v = \{v, u, u_1, u, u_2, u, \ldots, u_k\}$.

We show how to charge the edges of $\mathrm{MST}(S)$ for each such star-subpath and for each of the edges that are connecting between star-subpaths. First, notice that in each star-subpath T_v, each edge of the star is traversed at most twice. Let $U = \cup_{x \in P} T_x$ be the set of edges that are charged for the star-subpaths themselves. Because the edges of the stars are disjoint, each edge in U is charged at most twice.

Let W be the set of edges that are traversed in T when connecting between any two star-subpaths. Each edge of W is charged at most twice because we traverse the stars following the classic DFS order. Therefore, in both sets U, W each edge is charged at most twice. Notice that U, W are distinct.

The edges in U have length at most ε^*, so edges of length larger than ε^* can only appear in W. In addition, edges that are incident to leaves in $\mathrm{MST}(S)$ appear only in U, because they do not connect between star-subpaths. We conclude that leaf-edges are charged only twice, and edges of length larger than ε^* are also charged only twice.

Consider an edge $\{v, u\} \in T_v$ such that $L(v)$ is empty and $u \in I_{close}(v)$. Notice that the edge $\{v, u\}$ is charged only once in U (it appears once in T_v) and once in W (to connect u with the centers of stars in v's subtree), so in total it is charged at most twice.

The only case left to handle is edges $\{x, u_i\}$ in a star such that $u_i \in L(x)$ (x can be either the current vertex or its child $u \in I_{\mathrm{close}}(v)$). In this case, $I_{close}(u_i) = \emptyset$. If u_i is not a leaf, then there exists an edge $\{u_i, w_i\}$ such that $w_i \in I_{far}(u_i)$. Since the algorithm recursively runs on w_i, the edges $\{u_i, w_i\}$ is in W. In other words, for every $\{x, u_i\} \in U$ there exists an edge $\{u_i, w_i\} \in W \setminus U$ such that $\|u_i - w_i\| > \varepsilon^*$, which is charged only twice. Therefore, if $\{x, u_i\}$ is charged four times (twice in U and twice in W), we charge it three times, and transfer the forth charge to $\{u_i, w_i\}$, so that both edges are charged only three times in total. Note that no other edge can transfer its charge to $\{u_i, w_i\}$, because w_i has to come after u_i in the order of the traversal.

This gives a bound of $3 \cdot \ell(\mathrm{MST}(S))$. To further improve the bound, notice that the last edge $\{x, u_k\}$ is charged only once in U. Moreover, it is also charged only once in W, because the children in its subtree are the first to be called recursively in this step, so $\{x, u_k\}$ only appears on the subpath that goes back to x. In the worst case, when $k = 4$, we charge the first three edges $\{x, u_1\}, \{x, u_2\}, \{x, u_3\}$ a total of 3 times (by transferring one charge to the corresponding edge in W), and $\{x, u_4\}$ is charged only twice. By choosing the farthest child to be colored last, i.e., $\|x - u_4\| \geq \max\{\|x - u_1\|, \|x - u_2\|, \|x - u_3\|\}$, each of the first three edges can transfer a charge of $\frac{1}{4}$ to $\{x, u_4\}$, resulting in a total charge of at most $2\frac{3}{4}$ per edge of $\mathrm{MST}(S)$, as claimed. ◁

Theorem 8 follows from Claim 11, Claim 12, and Claim 13.

4.2 Min-sum discrete Fréchet-TSP

In this section we consider the case where the goal is to minimize the sum of the lengths. The following theorem is a corollary of Theorem 8.

▶ **Theorem 14.** *Given a set S of n points in the plane, two curves P, Q that partition S such that $d_{dF}(P, Q) = \varepsilon^*$ and $\ell(P) + \ell(Q) \leq 4.75 \cdot \ell(\mathrm{MST}(S))$ can be found in $O(n \log n)$ time.*

Proof. Let P and Q be the curves obtained by the algorithm in the previous section. Then by Theorem 8 $\ell(Q) \leq 2.75 \cdot \ell(\mathrm{MST}(S))$. For P, note that it follows a DFS traversal of the $\mathrm{MST}(S)$ restricted to the centers. Since each edge is used at most twice and no vertex is repeated as in Q, this gives $\ell(P) \leq 2 \cdot \ell(\mathrm{MST}(S))$. Combining the two bounds yields $\ell(P) + \ell(Q) \leq 4.75 \cdot \ell(\mathrm{MST}(S))$. ◀

This result can be improved by using a $(1 + \varepsilon)$ approximation for TSP (for $\varepsilon > 0$) on one of the curves. We make use of the algorithm of Kisfaludi-Bak, Nederlof and Węgrzycki [20] which computes in $2^{O(\varepsilon^{1-d})} n \log n$ time. We get the following result.

▶ **Theorem 15.** *Given a set S of n points in $\mathbb{R}^d$, two curves P, Q that partition S such that $\ell(P) + \ell(Q) \leq (4+\varepsilon) \cdot \ell(\mathrm{MST}(S))$ can be found in $O(n \log n)$ time for constant d, ε.*

Proof. Given the set of stars $S_1, \ldots, S_k$ that were obtained in the proof of Lemma 2, each star S_i has a **blue** vertex that is connected in S_i to a set of **red** vertices. We run an algorithm that computes a $(1+\varepsilon)$ approximation for the TSP. On the **blue** vertices, and obtain a curve P with the length $(1+\varepsilon) \cdot \ell(\mathrm{TSP}(S))$.

We then construct the curve Q on the **red** vertices following the order of the **blue** vertices in P, i.e., for each vertex $v \in P$ that corresponds to the star S_i, we connect the red vertices of S_i, and then connect these red paths according to the TSP order on the **blue** vertices. We can bound the length of Q by the length of a curve traversing P with detours for traversing the **red** vertices. The sum of lengths of these detours is bounded by traversing each edge of a star twice, which is bounded by $2 \cdot \ell(\mathrm{MST}(S))$. We therefore get that $\ell(Q) \leq \ell(P) + 2 \cdot \ell(\mathrm{TSP}(S))$ and thus $\ell(Q) + \ell(P) \leq (4+\varepsilon) \cdot \ell(\mathrm{TSP}(S))$. ◀

4.3 Comparing to the optimal solution

In previous sections, we compared the solution obtained from our algorithm to $\mathrm{TSP}(S)$. We now show that it gives a constant approximation comparing to the optimal solution.

▶ **Observation 16.** *Let P, Q be two curves that partition S, such that $d_{dF}(P, Q) = \varepsilon$ for some $\varepsilon > 0$. Then $\ell(\mathrm{TSP}(S)) \leq \ell(P) + \ell(Q) + \varepsilon$.*

Proof. By concatenating P and Q, we obtain a path of at most $\ell(P) + \ell(Q) + \varepsilon$ that traverses all the points of S, which is a feasible solution for TSP on S. ◀

Denote by P^*, Q^* two curves that partition S with $\mathrm{d}_{dF}(P^*, Q^*) = \varepsilon^*$ such that $\max\{\ell(P^*), \ell(Q^*)\}$ is minimized.

▶ **Lemma 17.** *Let P, Q be two curves that partition S, such that $d_{dF}(P, Q) = d_{dF}(P^*, Q^*) = \varepsilon^*$. Denote by L_P (reps. L_Q) the maximum length of an edge in P (resp. Q). If $|P|, |Q| \geq 2$, then $\varepsilon^* \leq L = \max\{L_P, L_Q\}$.*

Proof. Assume by contradiction that $\varepsilon^* > L$. That means that all edges of both P and Q are of strictly smaller length than ε^*. Denote $P = \{p_1, \ldots, p_k\}$ and $Q = \{q_1, \ldots, q_m\}$. If $k \geq 2$, then the discrete Fréchet distance between $P_{\mathrm{odd}} = \{p_1, p_3, p_5, \ldots\}$ and $P_{\mathrm{even}} = \{p_2, p_4, p_6, \ldots\}$ is at most L. Similarly, if $m \geq 2$, then the Fréchet distance between $Q_{\mathrm{odd}} = \{q_1, q_3, q_5, \ldots\}$ and $Q_{\mathrm{even}} = \{q_2, q_4, q_6, \ldots\}$ is at most L. Therefore, the discrete Fréchet distance between $P_{\mathrm{odd}} \circ Q_{\mathrm{odd}}$ and $P_{\mathrm{even}} \circ Q_{\mathrm{even}}$ is at most L, in contraction to the optimality of ε^*. ◀

Note that if one of P, Q is a single vertex, the above lemma may not be correct. Let $S = \{p, q_1, q_2, \ldots\}$ such that $\|p - q_i\| = \varepsilon$ for some $\varepsilon > 0$. Let $\|q_i - q_j\| \ll \varepsilon$ for all $q_i, q_j \in Q$. Then ε is the optimal Fréchet distance. And yet, the lemma does not hold for $P = \{p\}$ and $Q = \{q_1, q_2, \ldots\}$.

An immediate corollary of Observation 16 and Lemma 17 is that $\ell(\mathrm{TSP}(S)) \leq \ell(P) + \ell(Q) + \varepsilon \leq 3 \cdot \max\{\ell(P), \ell(Q)\}$, and $\ell(\mathrm{TSP}(S)) \leq 2 \cdot (\ell(P) + \ell(Q))$. Therefore, by Theorem 8 we have the follwiing corrolary.

▶ **Corollary 18.** *Given a set S of n points in the plane, we can find in $O(n \log n)$ time two curves P, Q that partition S such that $d_{dF}(P, Q) = \varepsilon^*$ and $\ell(P) + \ell(Q) = O(\ell(P^*) + \ell(Q^*))$, or $\max\{\ell(P), \ell(Q)\} = O(\max\{\ell(P^*), \ell(Q^*)\})$.*

5 Balancing the number of vertices

In this section, we address Problem 2 (balanced-Fréchet-TSP) under the discrete Fréchet distance. Clearly, it is possible that there is no pair of curves that partition S and have both $\mathrm{d}_{dF}(P,Q) \leq \varepsilon^*$ (the optimal distance) and $\max\{|P|,|Q|\} = \lceil n/2 \rceil$ (see, e.g. Figure 5).

We show that for points in the plane our solution to discrete-Fréchet-TSP can be adjusted such that $\max\{|P|,|Q|\} \leq \lfloor n/2 \rfloor + 2$. In other words, we prove the following theorem.

▶ **Theorem 19.** *Given a set S of points in the plane, there always exists two curves P,Q that partition S, such that $d_{dF}(P,Q) \leq \varepsilon^*$, $|P| \geq |Q|$ and $|P| - |Q| \leq 4$ (and in case that n is odd, $|P| - |Q| \leq 3$). Moreover, such curves can be found in $O(n \log n)$ time.*

Proof. Consider the set of stars $S_1, \ldots, S_k$ that were obtained in the proof of Lemma 2, and that can be computed in $O(n \log n)$ time by Theorem 5). Each star S_i has a blue vertex $\mathbf{blue}(S_i)$ which is connected in the spanning tree T_C to a set of red vertices $\mathbf{red}(S_i)$. For Theorem 5 we are using the nearest neighbor graph $\mathrm{NNG}(S)$, applying the unique nearest neighbor rule, and therefore the maximum degree in $\mathrm{NNG}(S)$ is at most 5. Therefore, for every $1 \leq i \leq k$, we have $\mathbf{red}(S_i) \leq 5$.

Notice that flipping the colors of the vertices in a star S_i does not change the correctness of the algorithm, and we still obtain two curves that partition S and have distance at most ε^*. Therefore, we can perform the following procedure. Let P and Q be the curves obtained from the algorithm of Theorem 5, then by construction $|Q| \geq |P|$. Set $w = |Q| - |P|$, and iterate over the stars $S_1, \ldots, S_k$: if $w \geq 5$, flip the colors in the current star, and update w. The algorithm terminates when $w \leq 4$, or after S_k is flipped.

Since initially each star S_i has exactly one vertex in P (its center $\mathbf{blue}(S_i)$) and at most 5 vertices in Q ($\mathbf{red}(S_i)$), a flip adds $c = |\mathbf{red}(S_i)| - 1 \leq 4$ vertices to P, and removes c vertices from Q, which in total reduces w by $2c \leq 8$. Therefore, after a flip we have $w \geq -3$. Assume by contradiction that the algorithm terminates with $w \geq 5$. Then all the stars $S_1, \ldots, S_k$ were flipped, but this is a contradiction because in the first step we had $w = |P| - |Q| > 0$, and thus after flipping all the stars we have $w = |Q| - |P| < 0$.

We conclude that when the algorithm terminates, we have $-3 \leq w \leq 4$. If $w < 0$, we flip all the stars, and get $0 \leq w = |Q| - |P| \leq 4$, as required. Finally, note if n is odd, then w must be odd, and therefore in this case we have $0 \leq w = |Q| - |P| \leq 3$. ◀

▶ Remark 20. In fact, the maximum degree of the NNG for points in d dimensions is equal the kissing number of spheres in d dimensions, and therefore Theorem 19 can be generalized to higher dimensions accordingly.

5.1 Relaxing the distance requirement

The example in Figure 5 shows a set S for which there is no balanced partition with distance ε^*. However, notice that if we relax the requirement on the distance between the curves and allow it to be up to $2\varepsilon^*$, then we can split each star into smaller stars by connecting pairs of leaves, so that the maximum degree of a star becomes two, which allows for a balanced partition. Below we show that by allowing an even smaller relaxation (the discrete Fréchet distance will be at most $\sqrt{3} \cdot \varepsilon^*$), we can always obtain optimally balanced curves.

▶ **Theorem 21.** *Given a set S of points in the plane, there always exists two curves P,Q that partition S, such that $d_{dF}(P,Q) \leq \sqrt{3} \cdot \varepsilon^*$ and $\max\{|P|,|Q|\} = \lceil n/2 \rceil$. Moreover, such curves can be found in $O(n \log n)$ time.*

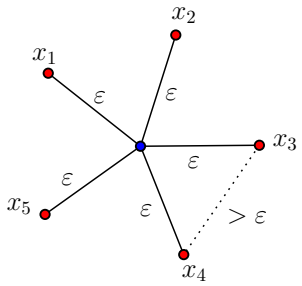


Figure 5 A set of 6 points for which the optimal solution for Fréchet-TSP is ε, $|P| = 1$ and $|Q| = 5$.

Proof. Consider the set of stars $S_1, \ldots, S_k$ that were obtained in the proof of Lemma 2. If the degree of each star is at most two (i.e. $|\mathbf{red}(S_i)| \leq 2$ for every $1 \leq i \leq k$), then by applying arguments similar to the proof of Theorem 19, we get that $\max\{|P|, |Q|\} = \lceil n/2 \rceil$ as required.

Otherwise, let S_i be a star such that $|\mathbf{red}(S_i)| \geq 3$. Let $c = \mathbf{blue}(S_i)$, then there is at least one pair of nodes $u, v \in \mathbf{red}(S_i)$ such that the smaller angle $\angle ucv$ at at most $\frac{2\pi}{3}$. Thus, by the law of cosines, the distance between them is

$$\|v - u\| = \sqrt{\|c - v\|^2 + \|c - u\|^2 - 2 \cdot \cos(\frac{2\pi}{3}) \cdot \|c - v\| \cdot \|c - u\|}.$$

Since the length of any edge in S_i is at most ε^*, we have

$$\|v - u\| \leq \sqrt{2(\varepsilon^*)^2 - 2(\varepsilon^*)^2 \cos(\frac{2\pi}{3})} = \sqrt{2(\varepsilon^*)^2 + (\varepsilon^*)^2} = \sqrt{3} \cdot \varepsilon^*.$$

We now split S_i into two stars: remove v, u from $\mathbf{red}(S_i)$ and create a new star, S_{k+1}, with $\mathbf{red}(S_{k+1}) = \{v\}$ and $\mathbf{blue}(S_{k+1}) = u$.

We continue this process until all our stars have degree at most two. Notice that any star that we add consists of a single pair of points with distance at most $\sqrt{3} \cdot \varepsilon^*$. Therefore, when applying the arguments from the proof of Theorem 19 as before, we obtain $\max\{|P|, |Q|\} = \lceil n/2 \rceil$, and $\mathrm{d}_{dF}(P, Q) \leq \sqrt{3} \cdot \varepsilon^*$. ◀

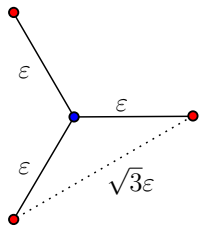


Figure 6 A set of 4 points for which the optimal solution for Fréchet-TSP is ε, $|P| = 1$ and $|Q| = 3$. By allowing the distance to be $\sqrt{3}\varepsilon$, we can obtain a solution where $|P| = |Q| = 3$.

The example in Figure 6 shows that the bounds in Theorem 19 and Theorem 21 are tight, as we conclude in the observation below.

▶ **Observation 22.** *There exists a set S of points in the plane such that: (i) for any two curves P, Q that partition S and have $d_{dF}(P, Q) \leq \varepsilon^*$ it holds that $|P| - |Q| \geq 4$, and (ii) for any two curves P, Q that partition S and have $|P| - |Q| = 0$ it holds that $d_{dF}(P, Q) \leq \sqrt{3} \cdot \varepsilon^*$.*

5.2 Tighter balancing

In the proof of Theorem 19 we flip the colors of some stars in order to obtain a difference of four between $|P|$ and $|Q|$. However, for some instances there might still be a different way to arrange the stars (i.e., a different paired-walk) that allows for a more balanced partition. Therefore, to further improve the difference, we now show how to split some of the starts (as we do in the proof of Theorem 21) in order to reduce the difference between $|P|$ and $|Q|$, while still having $\mathrm{d}_{dF}(P,Q) \leq \varepsilon^*$.

Given a star subgraph H of G_{ε^*}, define its ***balance*** as $\mathbf{b}(H) = \mathbf{deg}(H) - 1$, where $\mathbf{deg}(H)$ is the degree of the center node of H. We call a star H with balance $\mathbf{b}(H)$ a $\mathbf{b}(H)$-star (e.g., a 0-star, a 1-star, etc.). For a set X of star subgraphs of G_{ε^*}, denote $\mathbf{b}(X) = \sum_{H \in X} \mathbf{b}(H)$.

Let $\alpha = \{H_1, \ldots, H_t\}$ be a set of disjoint stars $H_1, \ldots, H_t$ in G_{ε^*} that together cover S. We call such a set a ***star cover*** of S. Consider a valid coloring of the stars in α in two colors, then as in the proof of Lemma 2 this coloring yields two curves that partition S and a paired-walk along them with cost at most ε. Intuitively, $\mathbf{b}(S_i)$ is the amount that a star contributes to either $|P|$ and $|Q|$. Therefore, the following problem is equivalent to Problem 2 (balanced-Fréchet-TSP).

▶ **Problem 5.** *Given a set S of n points in the plane and a value $\varepsilon > 0$, find a star cover $\alpha = H_1, \ldots, H_t$ of S, that can be partitioned into two sets of stars ρ and β, that minimizes the difference*

$$\Delta_\alpha(\rho, \beta) = |\boldsymbol{b}(\rho) - \boldsymbol{b}(\beta)| .$$

We thus focus on solving the problem defined above. For a given star cover α, an ***optimally balanced partition*** of α is a partition of α into two sets of stars ρ and β that minimizes $\Delta_\alpha \coloneqq \Delta_\alpha(\rho, \beta)$. In the full version of the paper, we show that when $|S| = n$ is even, any star cover α of S with $\Delta_\alpha = 4$ consists of only 0-stars and 4-stars. We then present an algorithm that runs in $O(n^2)$ time, and either returns a star cover α of S with $\Delta_\alpha \leq 2$, or reports (correctly) that the optimal star cover has $\Delta_\alpha = 4$.

▶ **Theorem 23.** *Given a set S of n points in the plane, where n is even, there exists an algorithm that runs in $O(n^2)$ time, which returns a star cover α of S s.t. either:*

1. *$\Delta_\alpha = 4$ and it is the minimum possible value of any star cover of S (i.e. α must be optimal).*
2. *$\Delta_\alpha \leq 2$.*

6 Continuous Fréchet-TSP is hard

As mentioned in the introduction, Buchin and Kilgus [11] show that the continuous Fréchet distance between 2 point sets is NP-hard. Similarly, we show that the Fréchet-TSP problem under the continuous Fréchet distance is also NP-hard. The reduction is quite similar to the reduction described in [11], and for completeness, we include a sketch of the proof. We reduce from a restricted version of 3-SAT, called $(3, B2)$-SAT, in which each clause contains exactly three literals, and each literal (that is, a variable or its negation) appears exactly twice in the entire formula. As in [11], we assume that in the $(3, B2)$-SAT instance no two clauses share more than one literal (this is possible by the construction in [8]).

Given a $(3, B2)$-SAT formula, we construct a set S of points in the plane such that the formula is satisfiable if and only if there exist two curves P, Q that partition S and have $d_F(P,Q) \leq \varepsilon$. Our construction is as follows. Each clause in the formula is represented by a

single point, where three line segments intersect, with one segment for each literal in the clause. This is possible because no two clauses share more than one literal. Since each literal appears at most twice in the formula, the line segment corresponding to a literal passes through the two points representing the clauses containing that literal. The variable gadgets are then created in a way for each clause point, one curve must visit it, and the other curves must pass close to it. This is possible because no two clauses share more than one literal. Since each literal appears in at most two clauses, the line segment corresponding to a literal passes through the two points representing the clauses containing that literal. This ensures that each clause is "covered" by the appropriate combination of curves according to the truth assignment encoded by the variable gadgets.

To make this construction concrete, we now describe the variable gadgets in detail. For every input variable x in the formula, we build a gadget similar to the one described in [11]. Each gadget has two points x_1, x_2 that represent the positive literal, and two points $\bar{x}_1, \bar{x}_2$ that represent the negative literal, see Figure 7 below.

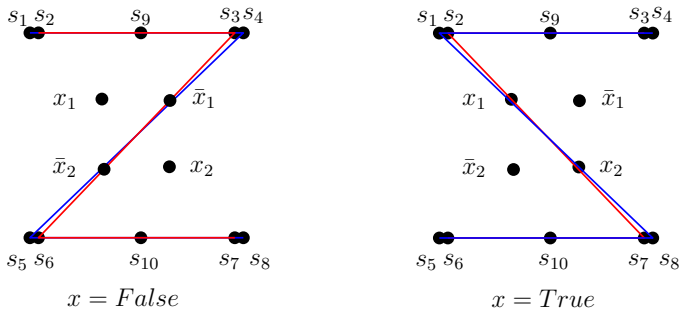


Figure 7 Variable x gadget.

The key difference for the 1-set case, compared to the original two–point-set case in [11], is that the vertices are not yet assigned to the curves. In order to force the partition to be as in [11], we place the points at the corners of the variable gadgets at distance $\varepsilon > 0$, for small enough ε. If the formula is stisfiable, then the optimal Fréchet distance will be ε,because each pair of corner points can be matched while still passing through all the clause points, as illustrated in Figure 7.

More precisely, the gadgets consists of four pairs of points (s_1, s_2), (s_3, s_4), (s_5, s_6) and (s_7, s_8), each pair of points is placed so the distance between them is very small ε. This ensures that both curves are forced to pass through all four corners, and, in order to preserve the uniqueness of each point along the curves, they cannot complete a full loop around the gadget and return to their starting corner. As a result, the traversal of the gadget is restricted, so that each curve may follow only one of the two diagonals. This restriction allows the gadget to enforce that exactly one of the line segments corresponding to x or $\bar{x}$ is visited by one of the curves (and the other one passes close enough to it), corresponding to the chosen assignment of the 3SAT variable.

For the clauses in the $(3, B2)$-Linear-SAT formula, we intersect the 3 different gadgets in one of the $x_1, x_2, \bar{x}_1, \bar{x}_2$ representing the variables in the clause, this is constructed in the same way as [11], as once the variable gadget are built the difference of the points being preassigned a curve or not does not affect the construction anymore, Figure 8.

▶ **Theorem 24.** *There is no polynomial-time solution for uniquely partitioning S into curves P, Q with minimal Fréchet distance, unless $P = NP$.*

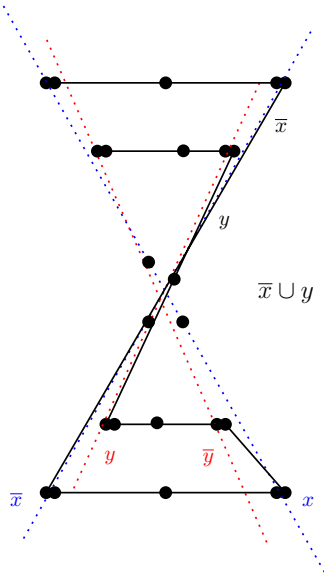


Figure 8 $(3, B2)$-SAT to 1 set continuous Fréchet problem.

▶ Remark 25. Partitioning S into curves P, Q with minimal weak Fréchet distance, also cannot be done in polynomial time unless $P = NP$. This follows directly from the proof over the Fréchet distance that also applies for the week case. Notice that the proof does not apply for the non-unique case, where we allow the same point to repeat on the same curve, but not in both because then we can just take identical curves over all set S.

References

1 Paul Accisano and Alper Üngör. Hardness results on curve/point set matching with fréchet distance. *CoRR*, abs/1211.2030, 2012. `arXiv:1211.2030`.

2 Pankaj K Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 203–210, 1990. `doi:10.1145/98524.98567`.

3 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995. `doi:10.1142/S0218195995000064`.

4 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998. `doi:10.1145/290179.290180`.

5 Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. `doi:10.1016/j.omega.2004.10.004`.

6 Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3d polygonal chains under the discrete fréchet distance. In *LATIN 2008: Theoretical Informatics*, pages 630–641, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. `doi:10.1007/978-3-540-78773-0_54`.

7 Mark de Berg, Kevin Buchin, Bart MP Jansen, and Gerhard Woeginger. Fine-grained complexity analysis of two classic tsp variants. *ACM Transactions on Algorithms (TALG)*, 17(1):1–29, 2020. `doi:10.1145/3414845`.

8 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electron. Colloquium Comput. Complex.*, TR03-049, 2003. URL: `https://eccc.weizmann.ac.il/eccc-reports/2003/TR03-049/index.html`.

9 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014. doi:10.1109/FOCS.2014.76.

10 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Bettina Speckmann. Locally correct fréchet matchings. *Comput. Geom.*, 76:1–18, 2019. doi:10.1016/J.COMGEO.2018.09.002.

11 Maike Buchin and Bernhard Kilgus. Fréchet distance between two point sets. *Comput. Geom.*, 102:101842, 2022. doi:10.1016/J.COMGEO.2021.101842.

12 Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, 2021. doi:10.1016/j.cosrev.2021.100369.

13 Siu-Wing Cheng, Haoqiang Huang, and Shuo Zhang. Constant approximation of fréchet distance in strongly subquadratic time. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 2329–2340. ACM, 2025. doi:10.1145/3717823.3718157.

14 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

15 Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *J. Math. Imaging Vis.*, 27(3):203–216, 2007. doi:10.1007/S10851-006-0647-0.

16 Thomas Eiter, Heikki Mannila, et al. Computing discrete fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, Technische Universität Wien, 1994.

17 David Eppstein, Michael S Paterson, and F Frances Yao. On nearest-neighbor graphs. *Discrete & Computational Geometry*, 17(3):263–282, 1997. doi:10.1007/PL00009293.

18 Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 1906.

19 Sariel Har-Peled and Benjamin Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *Journal of the ACM (JACM)*, 62(6):1–35, 2015. doi:10.1145/2831230.

20 Sándor Kisfaludi-Bak, Jesper Nederlof, and Karol Węgrzycki. A gap-ETH-tight approximation scheme for Euclidean TSP. *Journal of the ACM*, 72(6):1–48, 2025.

21 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Comput. Geom.*, 44(2):110–120, 2011. doi:10.1016/J.COMGEO.2010.09.008.

22 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Staying close to a curve. In *CCCG*, 2011.

23 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999. doi:10.1137/S0097539796309764.

24 David Mount and Mary Monroe. A ptas for the min-max euclidean multiple tsp, December 2021. doi:10.48550/arXiv.2112.04325.

25 Pravin M Vaidya. An o (n log n) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4:101–115, 1989. doi:10.1007/BF02187718.

26 Tim Wylie. Discretely following a curve. In *International Conference on Combinatorial Optimization and Applications*, pages 13–24. Springer, 2013. doi:10.1007/978-3-319-03780-6_2.

Constant-Factor Approximations for Doubly Constrained Fair k-Center, k-Median and k-Means

Nicole Funk
Department of Computer Science, University of Cologne, Germany

Annika Hennes
Faculty of Mathematics and Natural Sciences, Heinrich Heine University Düsseldorf, Germany

Johanna Hillebrand
Faculty of Mathematics and Natural Sciences, Heinrich Heine University Düsseldorf, Germany

Sarah Sturm
Department of Computer Science, University of Bonn, Germany

Abstract

We study discrete k-clustering problems in general metric spaces that are constrained by a combination of two different fairness conditions within the *demographic fairness* model. Given a metric space (P, d), where every point in P is equipped with a protected attribute, and a number k, the goal is to partition P into k clusters with a designated center each, such that a center-based objective function is minimized and the attributes are fairly distributed with respect to the following two fairness concepts: 1) *group fairness*: We aim for clusters with balanced numbers of attributes by specifying lower and upper bounds for the desired attribute proportions. 2) *diverse center selection*: Clusters have natural representatives, i.e., their centers. We ask for a balanced set of representatives by specifying the desired number of centers to choose from each attribute.

Dickerson, Esmaeili, Morgenstern, and Pena [13] denote the combination of these two constraints as *doubly constrained fair clustering*. They present algorithms whose guarantees depend on the best known approximation factors for either of these problems. Currently, this implies an 8-approximation with a small additive violation on the group fairness constraint. For k-center, we improve this approximation factor to 4 with a small additive violation. This guarantee also depends on the currently best algorithm for DS-fair k-center given by Jones, Nguyen and Nguyen [21]. For k-median and k-means, we propose the first constant-factor approximation algorithms. Our algorithms transform a solution that satisfies diverse center selection into a doubly constrained fair clustering using an LP-based approach. Furthermore, our results are generalizable to other center-selection constraints, such as matroid k-clustering and knapsack constraints.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Computing methodologies → Cluster analysis; Theory of computation → Rounding techniques; Theory of computation → Linear programming

Keywords and phrases Clustering, Fairness, Approximation Algorithms, k-center, k-median, k-means

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.19

Related Version *Extended Version*: `https://arxiv.org/abs/2604.16061`

Supplementary Material *Software*: `https://github.com/FairClusteringResearch/doubly_fair_clustering`

Funding *Nicole Funk*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 559931366.
Annika Hennes: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project 456558332.
Johanna Hillebrand: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project 459420781.
Sarah Sturm: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 459420781.

© Nicole Funk, Annika Hennes, Johanna Hillebrand, and Sarah Sturm;
licensed under Creative Commons License CC-BY 4.0

Acknowledgements We thank Melanie Schmidt for her valuable feedback and advice, Anna Arutyunova for proofreading the manuscript, and Lukas Drexler and Elmar Langetepe for productive initial discussions. We are also grateful to the three anonymous reviewers whose constructive comments helped improve this paper.

1 Introduction

With algorithms being used in a wide range of applications that can have a significant impact on individuals and society, fairness in machine learning has become an increasingly important topic in recent years. Much of the work is inspired by the disparate impact effect, which describes how machine learning algorithms may produce biased outcomes that disproportionately affect individual groups based on certain protected attributes. Clustering is a fundamental task in unsupervised learning, making this issue relevant in this context. Arguably, the most popular center-based clustering objectives are k-center, k-median, and k-means. For the unconstrained versions in general metric spaces, the best known approximation factors are 2 for k-center [19, 15], $2+\epsilon$ for k-median [11] and $3+2\sqrt{2}+\epsilon \approx 5.83$ for k-means [7].

Directly addressing the disparate impact effect, *demographic fairness* notions aim to ensure that protected attributes are fairly represented in the clusters. We say that data points belong to the same group if they share the same protected attribute value. Throughout, we will assume that any data point belongs to exactly one group, i. e., that the groups are disjoint. We also identify groups with different colors. Since the work of Chierichetti, Kumar, Lattanzi and Vassilvitskii [10] initiated the rigorous study of fair clustering, a significant amount of research has emerged on this topic [9, 12], spanning a variety of different fairness notions. Here, we focus on two specific demographic fairness constraints: *group fairness* and *diverse center selection*. Our goal will be to construct clusterings that satisfy the *combination* of these fairness constraints, but first, we will discuss them individually.

Group fairness. Group fairness concerns the composition of clusters, namely ensuring that the proportion of points from any group in each cluster is within certain bounds. For group fair k-center, k-median, and k-means there exist: a 3-approximation with additive violation of at most 1 [4], a $(4+\epsilon)$-approximation with violation of ≤ 1 [4, 3], and a 10.66-approximation with violation of ≤ 3 [3], respectively.

Diverse center selection. Complementing the group fairness notion, diverse center selection aims to ensure fair representation of the groups. As centers are naturally interpreted as representatives of the clusters, this constraint requires that from each group, a certain number of points be selected as centers. We also say that a clustering is *diversity-aware* when the selected centers satisfy this constraint. For diversity-aware k-center, k-median, and k-means with exact numbers of points per center, there exist: a 3-approximation [21], a 7.081-approximation [29, 23], and a 256-approximation [29, 30] (and a 64-approximation if the distances are Euclidean [29, 35]), respectively.

Doubly constrained fair clustering. Most works study these two constraints in isolation, but one could argue that, in many applications, it is desirable to achieve both constraints simultaneously. Group fairness alone can be useful to guarantee a range of different opinions and life experiences in a working environment or in a voting group, i. e., a cluster. This can boost productivity or prevent manipulation, such as gerrymandering, but loses its effect if the chosen representatives of the groups are not as diverse and act in their own interests. At

the other extreme, selecting only diverse centers without addressing the distributions within the clusters leaves the possibility of manipulation or peer pressure, mitigating the advantages of diverse centers.

Dickerson, Esmaeili, Morgenstern, and Pena [13] were the first to study the combination of these two constraints for the k-center problem and coined the term *doubly constrained fair clustering*. They present algorithms that sequentially solve the two problems to achieve solutions for the doubly constrained problem. The guarantees depend on the order in which the two problems are considered. By first computing a group fair solution, and transforming it into a doubly fair solution, they get a $2\alpha_{\mathrm{GF}}$-approximation for doubly fair k-center, where α_{GF} is the approximation guarantee of the algorithm used to compute the group fair clustering. Note that their approach requires an algorithm for group fair clustering with no additive violation, which currently does not exist for k-center. Therefore, we will only compare to their results for the other direction, where they first compute an α_{DS}-approximation for diversity-aware k-center and then transform it into a doubly constrained fair solution, which yields a $2(\alpha_{\mathrm{DS}}+1)$-approximation overall. While one could easily arrange for diverse center selection by moving centers into their own singleton clusters, to ensure group fairness, clusters usually need to contain multiple points in addition to their selected center.

Our contributions. We improve on known results for k-center and construct the first constant-factor approximation algorithms for doubly constrained fair k-median and k-means.

- We improve the currently best known approximation guarantee for doubly constrained fair k-center [13] by a factor of 2. Using the currently best approximation factor for diversity-aware k-center [21], this implies a 4-approximation for doubly constrained fair k-center with an additive violation of at most 2 for the group fairness constraint.
- We present the first polynomial-time constant-factor approximation algorithms for doubly constrained fair k-median and k-means. For k-median, our algorithm currently implies a 10.081-approximation, and for k-means, it yields a $291+2\sqrt{290}\approx 325.06$-approximation, both with an additive violation of at most 2 for the group fairness constraint.
- Our algorithms generalize to combinations of the group fairness constraint with other constraints on the centers, such as matroid and knapsack constraints or individual fairness.

Techniques. Our algorithms for all three objectives follow the same framework:

First, we independently compute a diversity-aware center set using an external approximation algorithm and a (fractional) group fair solution by solving a linear program. Next, we combine the two solutions by rerouting assignments so that only centers from the diversity-aware clustering get assigned points. Finally, we compute an integral solution by solving a max flow instance for k-center and a min cost flow instance for k-median and k-means.

The idea for the overall framework and the rerouting of assignments is inspired by the work of Ahmadian, Epasto, Kumar and Mahdian [2], who study a special case of group fairness with only upper bounds on the ratios of colors and provide a 3-approximation with additive violation of at most 2 for k-center without over-representation. We significantly adapt their techniques to handle our doubly constrained setting.

First, we must incorporate lower bounds into the group fairness constraint. While adapting the linear program is straightforward, it requires a more complex rerouting of assignments to ensure that the lower bounds are still satisfied after reassigning the points.

Second, we need to handle the fact that we have to combine two different solutions. They compute a set of centers that have pairwise distances of at least some appropriate threshold, which guarantees that there are at most k centers. The LP solution guarantees that every

point has a close LP center, and the lower bound on the pairwise distances of the centers guarantees that every new center has a close LP center that is not as close to other centers. This allows them to reroute all the mass assigned to an old center in the LP solution to the closest new center, and every new center gets assigned some positive mass. In our case, we do not have such a guarantee, as the diversity-aware solution does not provide a lower bound on the pairwise distances between the centers. Just reassigning the mass in the same way would not work as some centers from the diversity-aware solution might not get assigned any points, violating the requirement that all centers should be active. Therefore, we need to reroute the mass in a more complex way. We ensure that every cluster is assigned some mass, which may require splitting the incoming mass of an LP center appropriately among multiple centers from the diversity-aware solution while maintaining group fairness. Then, we reassign the remaining mass by the approach described above. For the other objectives, we need to refine the rerouting approach further to keep the solution's cost small.

We compare our results with those of [13] for the direction in which they transform a diversity-aware solution into a doubly constrained fair solution. Our algorithm performs the transformation in the same direction, and no applicable algorithm exists for the other direction. Their approach is similar to the framework described above, but differs in that they incorporate the centers computed by the diversity-aware clustering algorithm directly into their linear program for group fairness. Solving the linear program with the given centers causes the problem that some centers might not get assigned any points, i. e., they are not active. To overcome this issue, they replace inactive centers with points from suitable groups inside clusters that are large enough to be split into two group fair clusters. The additional factor of 2 in their approximation guarantee comes from this splitting step, which is not necessary in our approach. They finalize their solution by rounding the assignments by computing a max-flow, while we need to reroute assignments to other centers first before solving a max-flow instance.

1.1 Problem definition

Let (P, d) be a general metric space with $|P| = n$ and $k \in \{1, \dots, n\}$ an integer. A k-clustering is a partition of P into k clusters $C_1, \dots, C_k$ optimizing some objective function. We are considering center-based objectives, where a clustering is induced by a set of centers $\mathcal{C} \subseteq P$ with $|\,\mathcal{C}\,| \leq k$ and an assignment $\varphi\colon P \to \mathcal{C}$ mapping points to centers. Note that we study the *discrete* variant of k-clustering, where the set of centers must be a subset of the point set. The cluster with center c is given by $C = \varphi^{-1}(c)$. In the k-center objective, our goal is to minimize the maximal distance of any point to its assigned center, i. e., we want to minimize the maximal radius $\max_{p \in P} d(p, \varphi(p))$ among the clusters. In k-median, the objective is to minimize the sum of distances of points to their assigned center, i. e., $\sum_{p \in P} d(p, \varphi(p))$, and in k-means it is the sum of squared distances of points to centers that we want to minimize, i. e., $\sum_{p \in P} d(p, \varphi(p))^2$.

We study clusterings that satisfy two types of demographic fairness constraints. Both fairness notions rely on a coloring of the points. Let m be some number of colors. For $1 \leq h \leq m$, we denote the group of points of color h by P_h. We assume that every point in P is assigned exactly one color $1 \leq h \leq m$, i.e., $P = P_1 \,\dot\cup \dots \dot\cup\, P_m$. On the one hand, we want to achieve a fair distribution of colors in the clusters. This means that every color should constitute a certain fraction in every cluster. For every color h, we are given a lower bound ℓ_h and an upper bound u_h on these fractions. To relax this problem, we further allow a small additive violation of these fractional bounds.

▶ **Definition 1** (Group fair cluster). *For all $1 \leq h \leq m$, let ℓ_h, u_h be values with $0 \leq \ell_h \leq u_h \leq 1$. Let $\rho \in \mathbb{N}$. We say that a cluster C is* group fair with additive violation ρ *if*

$$\ell_h|C| - \rho \leq |C \cap P_h| \leq u_h|C| + \rho$$

for all colors $1 \leq h \leq m$. We say that a cluster is group fair *if $\rho = 0$.*

We say that a clustering fulfills the *group fairness constraint (GF)* if every cluster is group fair. If the lower and upper bounds on the ratios of the groups equal the respective proportions in the overall dataset, we also speak of *exact preservation of ratios* or short *exact group fairness*. Additionally, we aim for a fair representation of the colors, formalized by lower and upper bounds on the number of centers per group.

▶ **Definition 2** (Diverse center selection). *Given $L_h, U_h \in \mathbb{N}$ for $h \leq m$ such that $\sum_{h \leq m} L_h \leq k \leq \sum_{h \leq m} U_h$, we say that a set of centers C satisfies the* diverse center selection (DS) *constraint if $L_h \leq |C \cap P_h| \leq U_h$ for all $h \leq m$.*

The definition in its most general form is also called *fair-range* clustering; the special case with only lower bounds is also referred to as *minority protection*; we denote the case $L_h = U_h$ for all $h \in H$ by *exact diverse center selection*. All guarantees with fixed approximation guarantees stated in this paper hold for exact diverse center selection; some even hold for more general cases, see also the section on related work. Our algorithm does not depend on the specific combinatorial structure of the center set and therefore also works for the generalized version, provided appropriate approximation algorithms are available. Throughout, we will consider only instances with feasible solutions.

1.2 Our results

We have two main results; the first is a constant factor approximation algorithm for the doubly constrained fair k-center problem.

▶ **Theorem 3.** *Let γ_{center} be the approximation factor of a given approximation algorithm for DS-fair k-center. There exists an algorithm that computes a $(\gamma_{center}+1)$-approximation with GF-violation of ≤ 2 for the doubly constrained fair k-center problem in polynomial time.*

Using the currently best approximation factor of 3 for k-center with DS by Jones, Nguyen and Nguyen [21], this yields a 4-approximation.

▶ **Corollary 4.** *There exists an algorithm that computes a 4-approximation for the doubly constrained fair k-center problem with a GF-violation of at most 2.*

The second result extends the techniques of the k-center algorithm to obtain approximation algorithms for the doubly constrained fair k-median problem and the doubly constrained fair k-means problem.

▶ **Theorem 5.** *There exists a polynomial time algorithm that computes a*

- *$(\gamma_{med}+3)$-approximation with GF-violation of at most 2 for the doubly constrained fair k-median problem, where γ_{med} is the approximation factor of a given approximation algorithm for DS-fair k-median.*
- *$(\sqrt{1+(\sqrt{\gamma_{means}}+1)^2}+1)^2$-approximation with GF-violation of at most 2 for the doubly constrained fair k-means problem, where γ_{means} is the approximation factor of a given approximation algorithm for DS-fair k-means.*

Currently the best algorithm for center fair k-median computes a 7.081-approximation [29, 23] and for k-means a 256-approximation [29, 30], resulting in the following:

▶ **Corollary 6.** *There exists an algorithm that computes a*

- 10.081-*approximation with GF-violation of at most* 2 *for the doubly constrained fair* k-median *problem.*
- $291 + 2\sqrt{290} \approx 325.06$-*approximation with GF-violation of at most* 2 *for the doubly constrained fair* k-means *problem.*

For the special case of doubly constrained fair *k-means* under euclidean distance we can use the 64-approximation [29, 35] to achieve an $83 + 2\sqrt{82} \approx 101.11$-approximation with GF-violation of at most 2.

We remark that, as we use the approximation algorithms for clustering with diverse centers as opaque-box[1] algorithms, our results can also be applied to combine the group fairness constraint with different notions of fairness. Such a notion would need to apply only to the set of centers, e. g., matroid and knapsack constraints [27, 35] or the notion of individual fairness [22, 30].

1.3 Related work

Besides the two specific fairness definitions described above, there is an abundance of different fairness notions. These include, but are not limited to, more constraints focusing on fair distributions of points, such as balance [10], bounded representation [2], and colorful clustering [20]. Other fairness concepts focus on distances, like individual fairness [24] and socially fair clustering [1].

Chierichetti, Kumar, Lattanzi and Vassilvitskii [10] were the first to study the group fairness constraint (albeit not using this name) for the special case of two colors and a specific balance notion. Bercea, Groß, Khuller, Kumar, Rösner, Schmidt and Schmidt [4] presented the first approximation algorithms for group fair clustering for general metric spaces and general bounds on the proportions of groups for an arbitrary number of groups. For the fair k-center problem, there is a a deterministic 3-approximation algorithm with additive violation of at most 1 [4] and randomized 3-approximation with no additive violation in expectation [18]. For the exact variant, Bercea et al. [4] provide a 5-approximation algorithm with no additive violation for exact preservation of ratios, and show that it is NP-hard to approximate the assignment version of this problem, i. e., finding the optimal k-center assignment for a given set of centers, better than a factor of 3. For k-median, Bercea et al. [4] and Bera et al. [3] both give algorithms implying a $(4 + \epsilon)$-approximation with an additive violation of at most 1 for the group fairness constraint using the recent $(2 + \epsilon)$-approximation for k-median [11]. For k-means, a $5 + 4\sqrt{2} + \epsilon \approx 10.66$-approximation with violation of ≤ 3 is achieved by the algorithm from Bera, Chakrabarty, Flores and Negahbani [3], using the best known approximation for unconstrained k-means [7]. All these results assume that the groups are disjoint. Approximation algorithms for the generalization of non-disjoint groups also exist [3, 18]. Further related for group fairness include the integration of privacy constraints [26], the learning-augmented setting [32], streaming, and the distributed model [5].

We now look at the related work for the diverse center selection constraint. For the k-center objective, there exists a 3-approximation for exact center diversity [21], and a 3-approximation for a slight variant of diverse center selection that deals with lower and upper

[1] `https://www.acm.org/diversity-inclusion/words-matter`

Table 1 Comparison of approximation algorithms for doubly constrained fair k-clustering in general metric spaces. All results hold for exact center diversity; some of them hold for more general cases. Here, Δ denotes the maximum cluster size in the computed solution, and add. GF viol. denotes the additive violation of the group fairness constraint. The approximation ratios for this work use the currently best diversity-aware algorithms as subroutines: a 3-approximation for k-center [21], a 7.081-approximation for k-median [29, 23], and a 256-approximation for k-means [29, 35].

Objective	Reference	Approx. ratio	Add. GF viol.	Runtime
k-center	Dickerson et al. [13]	8	≤ 3	poly
	This work	**4**	≤ 2	poly
k-median	Zhang et al. [33]	$O(\Delta)$	≤ 5	poly
	Wu et al. [31]	$4+\varepsilon$	≤ 1	FPT
	This work	**10.081**	≤ 2	poly
k-means	Zhang et al. [33]	$O(\sqrt{\Delta})$	≤ 5	poly
	This work	$\mathbf{291 + 2\sqrt{290} \approx 325.06}$	≤ 2	poly

bounds on fractions of the actual proportions of a group in the data set [25]. If color groups are non-disjoint, the problem is inapproximable to any factor, assuming $P \neq NP$ [29]. For the most general version of diverse center selection, there exists an FPT 3-approximation that can even handle outliers [14]. For k-median and k-means, a result for general p-norm objectives implies $e^{O(1)}$-approximations for general diverse center selection. In the exact case, Thejaswi, Ordozgoiti and Gionis [29] show that the problem for the k-median objective can be reduced to the matroid median problem in polynomial time. This result can also be generalized to other objectives, implying a 7.081-approximation for k-median and a 256-approximation for k-means with exact diverse center selection, using the best known guarantees for the matroid version of the respective objective [23, 30]; where the k-means variant allows for a 64-approximation if the distances are Euclidean [29, 35]. In the FPT-regime, randomized $(1+\frac{2}{e}+\epsilon)$- and $(1+\frac{8}{e}+\epsilon)$-approximations for general diversity-aware k-median and k-means can be achieved, respectively [28]. For Euclidean spaces, $(1+\epsilon)$-approximations are possible for both objectives under the diverse center selection constraint [34]. Further related work in the realm of diverse center selection includes distributed models [6] and the streaming setting [6, 16].

For doubly constrained fair clustering, the landscape of related work is sparser. Only for k-center, there exists a constant-factor approximation algorithm [13]. For k-median, there is an FPT-time approximation algorithm by [31] that achieves an approximation ratio of $4+\epsilon$ with an additive violation of at most 1 for the group fairness constraint. For general p-norm objectives, there is a polynomial-time algorithm by [33] that achieves an approximation ratio of $O(\Delta^{1/p})$ with an additive violation of at most 5, where Δ is the maximum cluster size in the computed solution. Table 1 summarizes the results for doubly constrained fair clustering in general metric spaces, comparing our results to the state-of-the-art.

Lastly, we provide a brief overview of additional important center constraints that our algorithm can handle. There exists a 3-approximation for matroid k-center [8], a 7.081-approximation for matroid median [23], and a 256-approximation for matroid means [30], which is improved to 64 in Euclidean spaces [35]. For knapsack constraints, the factors are 3 [19], $6.387+\epsilon$ [17] and $1128+\epsilon$ [35] for k-center, k-median and (Euclidean) k-means.

2 A 4-approximation for k-center with center diversity and group fairness constraints

First, we give an overview of the algorithm, followed by a detailed description of the individual steps. As an initial step, compute a fractional solution (x, y) to the k-center clustering problem with only the group fairness constraint using linear programming, where x represents a fractional assignment of points to centers, and y indicates which points receive mass from other points in x, i.e., are used as centers in (x, y). Furthermore, we compute a set of centers $\mathcal{C}_{\text{DS}}$ satisfying center diversity using a known approximation algorithm for this problem. As a next step, we reroute the mass in (x, y) such that only the points in $\mathcal{C}_{\text{DS}}$ are used as centers. This yields a fractional solution satisfying center diversity and group fairness. Lastly, we convert the fractional assignment to an integral one by using a max flow with lower bounds. To discuss the algorithm in more detail, we divide it into four components, which are later formalized in Algorithm 1.

1. Compute a set of centers $\mathcal{C}_{\text{DS}}$ that fulfills center diversity
2. Compute a fractional group fair solution (x, y)
3. Reroute mass in x such that only points in $\mathcal{C}_{\text{DS}}$ receive positive mass $\rightarrow (x', y')$
4. Compute an integral point assignment $\rightarrow (x'', y'')$

2.1 Center diversity

To guarantee our solution fulfills the DS constraint we use an existing algorithm for diversity-aware k-center as an opaque-box algorithm. In the following, let $\mathcal{C}_{\text{DS}}$ be the returned set of centers that satisfy the DS constraint, and γ_{center} be the approximation factor of the algorithm. Currently, the best known algorithm for diversity-aware k-center is the 3-approximation by Jones et al. [21] with $O(nk)$ running time. If the combination of group fairness with another constraint is desired, a corresponding approximation algorithm can be used instead.

2.2 Group fairness

We formulate a linear program to obtain an initial fractional group fair clustering. We use a matrix of variables x representing the assignment from points to centers, i.e., x_{ij} denotes the amount of mass sent from point $j \in P$ to point $i \in P$, i. e., the amount by which a point j is assigned to a center i. Additionally, we compute a vector of variables y, where y_i can be interpreted as the amount to which $i \in P$ is opened as a center.

$$\sum_{i \in P} x_{ij} = 1 \qquad \forall j \in P \qquad \text{(LP-1.1)}$$

$$x_{ij} \leq y_i \qquad \forall i, j \in P \qquad \text{(LP-1.2)}$$

$$\sum_{i \in P} y_i \leq k \qquad \text{(LP-1.3)}$$

$$\sum_{j \in P_h} x_{ij} \leq u_h \sum_{j \in P} x_{ij} \qquad \forall h \in H, i \in P \qquad \text{(LP-1.4)}$$

$$\sum_{j \in P_h} x_{ij} \geq \ell_h \sum_{j \in P} x_{ij} \qquad \forall h \in H, i \in P \qquad \text{(LP-1.5)}$$

$$x_{ij} = 0 \qquad \forall j \in P, i \in P : d(i, j) > \lambda \qquad \text{(LP-1.6)}$$

$$0 \leq x_{ij}, y_i \leq 1 \qquad \forall i, j \in P \qquad \text{(LP-1.7)}$$

We denote the linear program by LP(λ). Note that the k-center objective function is not linear. However, we can control the cost of a feasible solution via constraint (LP-1.6), which ensures that points are not assigned to centers farther than λ away. Combining this with a search over all possible values of λ allows us to drop the objective function. The group fairness (as described in Definition 1) is enforced by constraints (LP-1.4) and (LP-1.5).

This linear program is a correct formalization of k-center clustering with group fairness constraint. To see this, we prove that any feasible k-center clustering satisfying group fairness corresponds to a feasible solution for LP(λ) when λ is at least the objective value of the clustering. The proof of the following two lemmas can be found in the full version.

▶ **Lemma 7.** *Let $(\mathcal{C}, \varphi)$ be a feasible k-center clustering satisfying GF with objective value $\lambda^{\mathcal{C}}$. Then, (x, y) defined by*

$$x_{ij} = \begin{cases} 1 & \text{if } i = \varphi(j) \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad y_i = \begin{cases} 1 & \text{if } i \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

is a feasible solution for LP($\lambda^{\mathcal{C}}$).

▶ **Lemma 8.** *Let (x, y) be a feasible integral solution for LP(λ). Then, $(\mathcal{C}, \varphi)$ defined by*

$$\mathcal{C} = \{i \in P \mid y_i = 1\} \quad \text{and} \quad \varphi(j) = \arg\max_{i \in P} x_{ij} \quad \text{for all } j \in P$$

is a feasible k-center clustering satisfying group fairness with objective value λ.

We choose the value of λ as follows. It should be large enough such that LP(λ) allows for a feasible solution. For this, let λ^{LP} denote the minimal value such that LP(λ^{LP}) has a solution. Further, λ should be large enough such that the γ_{center}-approximation for k-center with DS has cost upper bounded by $\gamma_{\text{center}} \cdot \lambda$. For this, let λ^{DS} be minimal such that the γ_{center}-approximation returns a solution with exactly k centers and value of at most $\gamma_{\text{center}} \cdot \lambda^{\text{DS}}$. Now, we set $\lambda := \max\{\lambda^{\text{LP}}, \lambda^{\text{DS}}\}$. We can upper bound the value of λ by the value of an optimal solution satisfying both GF and DS.

▶ **Lemma 9.** *Let OPT_{GF+DS} be the value of an optimal k-center solution satisfying GF and DS. Then, $\lambda := \max\{\lambda^{LP}, \lambda^{DS}\} \leq \text{OPT}_{GF+DS}$.*

Proof. Let OPT_{GF} and OPT_{DS} denote the values of an optimal k-center clustering satisfying GF and DS, respectively. By Lemma 7, LP(OPT_{GF}) contains a feasible solution. This implies $\lambda^{\text{LP}} \leq \text{OPT}_{\text{GF}}$. Let cost_{DS} denote the cost of the solution computed by the γ_{center}-approximation for k-center with DS. It is $\text{cost}_{\text{DS}} \leq \gamma_{\text{center}} \cdot \text{OPT}_{\text{DS}}$. We set λ^{DS} to the minimal value such that $\text{cost}_{\text{DS}} \leq \gamma_{\text{center}} \cdot \lambda^{\text{DS}}$. Then, $\lambda^{\text{DS}} \leq \text{OPT}_{\text{DS}}$.

Any solution satisfying both GF and DS is also a feasible solution for GF or DS, respectively. Hence, $\text{OPT}_{\text{GF}} \leq \text{OPT}_{\text{GF}+\text{DS}}$ and $\text{OPT}_{\text{DS}} \leq \text{OPT}_{\text{GF}+\text{DS}}$. Putting everything together yields $\max\{\lambda^{LP}, \lambda^{\text{DS}}\} \leq \max\{\text{OPT}_{\text{GF}}, \text{OPT}_{\text{DS}}\} \leq \text{OPT}_{\text{GF}+\text{DS}}$. ◀

2.3 Rerouting fractional assignments

Let (x, y) be the (fractional) group fair solution computed by solving the LP optimally, and $\mathcal{C}_{\text{DS}}$ be the set of diverse centers computed by a γ_{center}-approximation for k-center with DS constraint. If $x_{ij} > 0$ for some $j \in P$, the point i can be considered as a center in this solution. We want to reroute mass to the centers in $\mathcal{C}_{\text{DS}}$, such that positive mass is only sent to points in $\mathcal{C}_{\text{DS}}$. Meanwhile, we want to ensure that all centers in $\mathcal{C}_{\text{DS}}$ get some positive

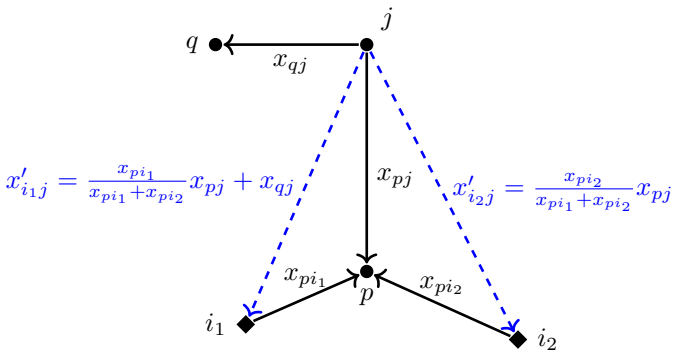


Figure 1 Rerouting the mass outgoing at a point j. There are two centers $i_1, i_2 \in \mathcal{C}_{\mathrm{DS}}$. The point j covers both cases: there is a point $p \in N(\mathcal{C}_{\mathrm{DS}})$ and a point $q \notin N(\mathcal{C}_{\mathrm{DS}})$ that j sends mass to. The mass x_{pj} is split and rerouted to the two centers $i_1, i_2 \in \mathcal{C}_{\mathrm{DS}}$ proportionally to the mass they send to p. Center i_c gets an $\frac{x_{pi_c}}{x_{pi_1}+x_{pi_2}}$-fraction of x_{pj} for $c = 1, 2$. Let us assume that $i_1 = \operatorname{argmin}_{i \in \mathcal{C}_{\mathrm{DS}}} d(q, i)$. Then, the mass x_{qj} is fully rerouted to i_1. The original mass assignments are indicated using solid black lines; the rerouted mass assignments are shown with blue dashed lines. Omitted edges correspond to 0 assignments.

mass. That is, we construct (x', y') such that $\{i \in P \mid \sum_{j \in P} x'_{ij} > 0\} = \mathcal{C}_{\mathrm{DS}}$, while ensuring that the cost does not increase too much. In this context, we need the notion of a point's *neighborhood*. For a point $j \in P$, we define the set $N(j) \coloneqq \{i \in P \mid x_{ij} > 0\}$ of all the points that j sends positive mass to (i. e., is assigned to in the fractional solution). For a set of points $P' \subseteq P$, we define $N(P') \coloneqq \bigcup_{j \in P'} N(j)$.

The overall idea is to take all the mass incoming at some point $p \in P$ in the LP solution (x, y) and reroute it to a "nearby" center in $\mathcal{C}_{\mathrm{DS}}$ if this is possible. We have to be careful that this does not change the color proportions. Then, as (x, y) fulfills group fairness, the newly formed clusters are also group fair. Consider a point $p \in P$ that gets some mass in the current solution, i.e., $\sum_{j \in P} x_{pj} > 0$. Depending on p's position relative to the centers in $\mathcal{C}_{\mathrm{DS}}$, we decide how to reroute the incoming mass at p.

We want to prevent any center in $\mathcal{C}_{\mathrm{DS}}$ from becoming empty (i.e., it gets no mass assigned). By the first constraint of the LP, $\sum_{i \in P} x_{ij} = 1$, every point, particularly every center in $\mathcal{C}_{\mathrm{DS}}$, must possess some outgoing mass in the LP solution (x, y). That is, for every $c \in \mathcal{C}_{\mathrm{DS}}$, the set $N(c)$ is non-empty. To ensure that mass is sent to a point $c \in \mathcal{C}_{\mathrm{DS}}$, we reroute mass sent to a point $p \in N(c)$ to c. However, these sets might overlap, and if we sent all incoming mass at $p \in N(c)$ to c, there might not be any mass left to send to some other center that has p in its neighborhood. To circumvent this problem, we proportionally split the mass sent to a point $p \in N(\mathcal{C}_{\mathrm{DS}})$ among all points $c \in \mathcal{C}_{\mathrm{DS}}$ with $p \in N(c)$. A center $i \in \mathcal{C}_{\mathrm{DS}}$ gets as much mass from $j \in P$ with $x_{p,j} > 0$ as it sends to p, proportional to the masses sent from all centers $i' \in \mathcal{C}_{\mathrm{DS}}$ to p.

In the remaining case that $p \notin N(\mathcal{C}_{\mathrm{DS}})$, p gets mass solely from points that are not desired centers themselves. In this case, we can safely reroute all mass $\sum_{j \in P} x_{pj}$ to some nearby center in $\mathcal{C}_{\mathrm{DS}}$. To keep the distances small, we choose the center $i \in \arg\min_{i' \in \mathcal{C}_{\mathrm{DS}}} d(p, i')$ that is closest to p. Formally, a new fractional solution (x', y') is defined as follows:

$$y'_i = \begin{cases} 1 & \text{if } i \in \mathcal{C}_{\mathrm{DS}} \\ 0 & \text{otherwise} \end{cases} \qquad \text{(rerouting-}y\text{)}$$

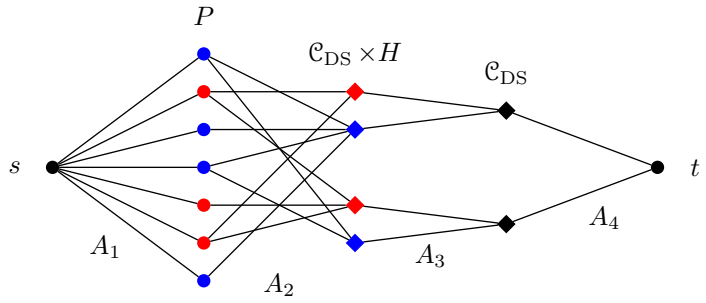


Figure 2 Flow network to find the final assignment of points to centers. Every point in $p \in P$ has a node representing it, which has an edge in A_1 to the node s. In the figure, these nodes are colored blue or red to represent the color h_p of p. Every such node has an edge in A_2 to every node corresponding to a pair of a center $c \in \mathcal{C}_{\text{DS}}$ and the color h_p, i. e., $(c, h_p) \in \mathcal{C}_{\text{DS}} \times H$, which are colored accordingly in the figure. Each pair of center and color (c, h_p) has an edge in A_3 to a node representing the center c in the pair. Each of these nodes corresponding to a center $c \in \mathcal{C}_{\text{DS}}$ has an edge in A_4 connecting it to the node t.

and

$$x'_{ij} = \begin{cases} \sum_{p \in N(i)} \frac{x_{pi}}{\sum_{c \in \mathcal{C}_{\text{DS}} \colon p \in N(c)} x_{pc}} x_{pj} + \sum_{p \in \theta^{-1}(i)} x_{pj} & \text{if } i \in \mathcal{C}_{\text{DS}} \\ 0 & \text{otherwise,} \end{cases} \qquad \text{(rerouting-}x\text{)}$$

where $\theta \colon P \setminus N(\mathcal{C}_{\text{DS}}) \to \mathcal{C}_{\text{DS}}$ with

$$\theta(p) = \begin{cases} p & \text{if } p \in \mathcal{C}_{\text{DS}} \\ i & \text{with } i \in \operatorname{argmin}_{i' \in \mathcal{C}_{\text{DS}}} d(p, i'), \text{ otherwise} \end{cases}$$

Figure 1 schematically visualizes how the mass emitted from a point j is rerouted to centers in $\mathcal{C}_{\text{DS}}$. We can show that the rerouted solution (x', y') is a feasible solution to $\text{LP}((\gamma_{\text{center}}+1)\lambda)$ such that y' is integral. Further, every $c \in \mathcal{C}_{\text{DS}}$ gets a mass of at least 1 assigned. We defer the proof to the full version.

▶ **Lemma 10.** *Let (x, y) be a feasible solution for LP(λ) and let (x', y') be the solution achieved through the rerouting described above. Then the following properties hold:*

1. *(x', y') is a feasible solution to LP($(\gamma_{center}+1)\lambda$)*
2. *$\sum_{j \in P} x'_{ij} \geq 1$ for all $i \in \mathcal{C}_{DS}$*
3. *y' is integral.*

2.4 Final fair assignment

After the rerouting step (see Section 2.3), we have an assignment that is group fair and satisfies the diverse center selection constraint. However, the assignment x' is still fractional. To achieve the final integral assignment of points to centers, we build a network graph and solve a max flow problem with lower bounds on it, similar to [2]. This is done as follows:

Let $G = (V, E)$, where $V = \{s, t\} \cup P \cup \{(i, h) \mid i \in \mathcal{C}_{\text{DS}}, h \in H\} \cup \mathcal{C}_{\text{DS}}$ and $E = A_1 \cup A_2 \cup A_3 \cup A_4$. The set of edges $A_1 = \{(s, j) \mid j \in P\}$ connects all points in P to the source. Each point $j \in P$ is connected to a center-color-node (i, h), if $j \in P_h$ and $x'_{ij} = 1$. More formally $A_2 = \{(j, (i, h)) \mid i \in \mathcal{C}_{\text{DS}}, j \in P_h, x'_{ij} > 0\}$. The edges in A_1 and A_2 have capacity 1. Each of the color-center-nodes (i, h) is connected to its corresponding center i, thus $A_3 = \{((i, h), i) \mid i \in \mathcal{C}_{\text{DS}}, h \in H\}$. The capacities on these edges ensure that each center

receives (nearly) the same amount of flow of each color as it does in the fractional group fair solution. Since $\sum_{j\in P_h} x'_{ij}$ is not necessarily integral, lower and upper bounds on the edges are introduced; the lower bound is $\lfloor\sum_{j\in P_h} x'_{ij}\rfloor$ and the upper bound is $\lceil\sum_{j\in P_h} x'_{ij}\rceil$. Lastly, the set of edges $A_4 = \{(i,t) \mid i \in \mathcal{C}_{\text{DS}}\}$ connects all center points to the sink t. These edges are used to ensure that each cluster center receives (nearly) as much total flow as it received in the fractional solution, but again $\sum_{j\in P} x'_{ij}$ is not necessarily integral. Each edge has the lower bound $\lfloor\sum_{j\in P} x'_{ij}\rfloor$ and the upper bound $\lceil\sum_{j\in P} x'_{ij}\rceil$ (see Figure 2).

The important property of a max flow we use here is that in a network with integral capacities and integral demands, there always exists a feasible assignment that is integral. Since there is a flow with value $|P|$ (using x'), there is also an integral flow with value $|P|$. Let x'' be the integral assignment from the flow. Our final solution is given by the pair (y', x''). This step can introduce an additive violation of 2.

▶ **Lemma 11** (Group fairness – k-center). *For each center i and each color h it holds that*

$$l_h \sum_{j\in P} x''_{ij} - 2 \leq \sum_{j\in P_h} x''_{ij} \leq u_h \sum_{j\in P} x''_{ij} + 2.$$

Proof (analogous to [2]). The full proof for Lemma 11 can be found in the full version. ◀

Algorithm 1 Doubly constrained fair k-center.

Input : point set P, distance metric d, $k \in \mathbb{N}$, group fairness bounds $\ell_h, u_h \in \mathbb{N}$ for all $h \in H$, algorithm for center constraint $\mathcal{A}_{center}$
Output : center set $\mathcal{C}$ and assignment φ

$\mathcal{C}_{\text{DS}}, \text{cost}_{\text{DS}} \leftarrow$ centers and objective value returned by $\mathcal{A}_{center}(P, d, k)$
$R \leftarrow \{d(p,q) \mid p, q \in P\}$
$\lambda^{\text{DS}} \leftarrow \min\{r \in R \mid r \geq \text{cost}_{\text{DS}} / \gamma_{\text{center}}\}$
$\lambda^{\text{GF}} \leftarrow \min\{r \in R \mid \text{LP}(r) \text{ has a solution}\}$
$\lambda \leftarrow \max\{\lambda^{\text{LP}}, \lambda^{\text{DS}}\}$
$(x, y) \leftarrow$ optimal (fractional) solution for LP(λ) under GF constraints $\ell, u \in \mathbb{N}^{|H|}$
$(x', y') \leftarrow$ rerouting of (x, y) as specified in Equations (rerouting-x) and (rerouting-y)
$(x'', y'') \leftarrow$ final integral assignment via max flow as described in Section 2.4
$\mathcal{C} \leftarrow \{i \in P \mid y''_i = 1\}$
for $j \in P$ **do**
  $\varphi(j) \leftarrow \arg\max_{i\in P} x''_{ij}$
return $(\mathcal{C}, \varphi)$

This construction results in Theorem 3. We defer the full proof to the full version of this paper.

▶ **Theorem 3.** *Let γ_{center} be the approximation factor of a given approximation algorithm for DS-fair k-center. There exists an algorithm that computes a $(\gamma_{center}+1)$-approximation with GF-violation of ≤ 2 for the doubly constrained fair k-center problem in polynomial time.*

3 An Algorithm for k-median/k-means with center diversity and group fairness constraints

In this section, we look at the doubly fair clustering problem under the k-median and k-means objective functions. The basic structure of the algorithm we use for these objectives is the same as for the k-center objective described in Section 2. We again use a fractional

LP solution that fulfills group fairness and reroute it such that it uses centers we obtain via an opaque-box algorithm for center diversity. The resulting fractional assignment is then rounded via a min cost flow procedure. In the following, we describe the steps of the algorithm for these cost functions and prove its correctness. For notational simplicity we describe the k-median cost of a solution (x, y) by $\text{cost}(x, y) = \sum_{i \in P} \sum_{j \in P} x_{ij} d(i, j)$ and the k-means cost by $\text{cost}^2(x, y) = \sum_{i \in P} \sum_{j \in P} x_{ij} d(i, j)^2$.

3.1 Center diversity

We find a center fair solution $(\mathcal{C}_{\text{DS}}, \varphi)$ via an opaque-box algorithm for the respective problem. For notation reasons, we can interpret this solution as an integral assignment (x^{DS}, y^{DS}) with $y_i^{DS} = 1$ if and only if $i \in \mathcal{C}_{\text{DS}}$ and $x_{ij}^{DS} = 1$ iff $\varphi(j) = i$. Let γ_{med} and γ_{means} be the resulting approximation factors, respectively. The currently best available approximation algorithms for diverse center selection for these objectives are the following: For k-median with DS, Thejaswi et al. [29] give a reduction to the matroid median problem, allowing a ρ-approximation algorithms for the matroid median problem to be used to compute a ρ-approximation for k-median with DS. Currently the best known algorithm for the matroid median problem is the 7.081-approximation algorithm by Krishnaswamy et al. [23].

The reduction by Thejaswi et al. [29] can also be modified to reduce k-means with DS to the facility location problem with ℓ_p-norm cost under matroid constraint, where $p = 2$, introduced by Vakilian and Yalçıner [30]. For this, we only need to set the opening costs of every facility to 0 in addition to the construction of the k-median reduction. As the number of opened facilities is bounded by the partition matroids constructed in the reduction, the arguments by Thejaswi et al. apply here as well. Vakilian and Yalçıner [30] also give a 16^p approximation for the facility location problem with ℓ_p-norm cost under matroid constraint, meaning we can use it to obtain a 256-approximation for k-means with DS. For the special case of euclidean distance can also use the construction by Thejaswi et al. [29] to reduce from the matroid means problem to k-means with DS. This allows us to use the 64-approximation algorithm for matroid means under Euclidean Distance by Zhao et al. [35].

3.2 Group fairness

The following relaxed LP finds a feasible assignment satisfying the GF constraint and minimizes the k-median objective function

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j \in P} x_{ij} \cdot d(i,j)^2 && && \text{(LP-Med/LP-Means)} \\
\text{subject to} \quad & \sum_{i \in P} x_{ij} = 1 && \forall j \in P && \text{(LP-2.1)} \\
& x_{ij} \leq y_i && \forall i, j \in P && \text{(LP-2.2)} \\
& \sum_{i \in P} y_i \leq k && && \text{(LP-2.3)} \\
& \sum_{x \in P^h} x_{ij} \leq \alpha_h \sum_{x \in P} x_{ij} && \forall h \in H, i \in P && \text{(LP-2.4)} \\
& \sum_{x \in P^h} x_{ij} \geq \beta_h \sum_{x \in P} x_{ij} && \forall h \in H, i \in P && \text{(LP-2.5)} \\
& 0 \leq x_{ij}, y_i \leq 1 && \forall i, j \in P && \text{(LP-2.6)}
\end{aligned}$$

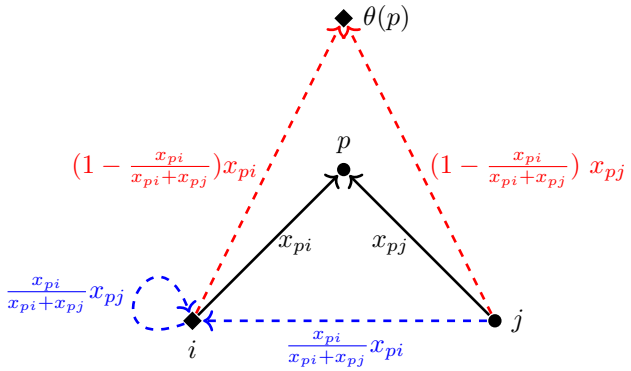


Figure 3 Rerouting of the mass arriving on p. The solution (x, y) is depicted by the black arrows. The blue dashed arrows show the rerouting back to $i \in \mathcal{C}_{\text{DS}}$ from $p \in N(i)$. The red dashed arrows show the nearest neighbor rerouting of the remaining mass.

Note that the LP is in large parts identical to the one in section 2. The only difference is that both the k-median and k-means objective function are linear, so instead of using Equation (LP-1.6) we now add it as proper objective function to the LP. Which objective function (LP-Med or LP-Means) is used depends on if we are interested in the k-median or the k-means problem. Analogously to Lemma 7 and Lemma 8 it can be shown how to obtain clustering solutions from the LP and the other way round.

In the following we denote by (x, y) an optimal solution to the above relaxed LP. Since the algorithm largely works the same for both objectives, we often do not specify which one is used except for cases where it is relevant. Note that the cost of (x, y) is a lower bound for the cost of the optimal doubly fair solution. This holds for both the k-median and the k-means cost function.

3.3 Rerouting

We use the center set $\mathcal{C}_{\text{DS}}$ from Section 3.1 and reroute the point assignments of (x^{LP}, y^{LP}) in a way that the following hold:

- the resulting solution (x', y') is a feasible solution to the LP
- for all $i \in \mathcal{C}_{\text{DS}}$ it holds $\sum_{j \in P} x_{ij} \geq 1$
- we can bound the cost of the resulting solution by the following:
  - $\text{cost}(x', y') \leq 3 \cdot \text{cost}(x, y) + \text{cost}(x^{DS}, y^{DS})$ in the k-median case
  - $\text{cost}^2(x', y') \leq (1 + p^2 + (1 + \frac{1}{p^2})(2 + q^2))\,\text{cost}^2(x, y) + (1 + \frac{1}{p^2})(1 + \frac{1}{q^2})\,\text{cost}^2(x^{DS}, y^{DS})$ for $p, q > 0$ in the k-means case
- all y'_i are integral

We first focus on the second requirement. For $i \in \mathcal{C}_{\text{DS}}$ we look at the set $N(i) := \{p \in P \mid x_{pi} > 0\}$ which is the set of points that get assigned positive mass from i in (x, y). The main idea is to redirect the mass that is assigned to a point $p \in N(i)$ such that i gets assigned a share of this mass that is equivalent to x_{pi}. After we have done this for all i, p, we reroute the remaining mass that arrives at p by sending it to the nearest neighbor of p in $\mathcal{C}_{\text{DS}}$. For this we use the same mapping function $\theta : \{i \in P \mid y_i > 0\} \to \mathcal{C}_{\text{DS}}$ as in Section 2.3.

We denote by $r_p^i := \frac{x_{pi}}{\sum_{l \in P} x_{pl}}$ the ratio of x_{pi} to the total mass of assignments on p. Note that $r_p^i = 0$ if $x_{pi} = 0$. Using this notation we define (x', y') the following way:

$$y'_i = \begin{cases} 1 & \text{if } i \in \mathcal{C}_{\text{DS}} \\ 0 & \text{otherwise} \end{cases}, \quad x'_{ij} = \begin{cases} \sum_{p \in N(i)} r_p^i x_{pj} + \sum_{p \in \theta^{-1}(i)} (1 - \sum_{l' \in \mathcal{C}_{\text{DS}}} r_p^{l'}) x_{pj} & \text{if } i \in \mathcal{C}_{\text{DS}} \\ 0 & \text{otherwise} \end{cases}$$

An example for this rerouting can be seen in Figure 3. The main difference of this rerouting step to the rerouting in Section 2.3 is that a point $i \in \mathcal{C}_{\text{DS}}$ now does not get all the mass that is assigned to a point $p \in N(i)$ (even if he is the only center sending mass there), but only a fraction of it that is equivalent to x_{pi}. This property will later be important when we are bounding the cost of this solution.

The following lemmas show that (x', y') do indeed have the desired properties. We defer the corresponding proofs to the full version of this paper. Note that the solutions (x', y') depend on the objective function (k-median or k-means), since it is used in the LP in Section 3.2 and the respective opaque-box algorithm in Section 3.1.

▶ **Lemma 12.** *(x', y') is a feasible solution to the LP in Section 3.2. It holds $\sum_{j \in P} x'_{ij} \geq 1$ for all $i \in \mathcal{C}_{DS}$. All y'_i in this solution are integral.*

▶ **Lemma 13** (Cost k-median)**.** *For the k-median cost of (x', y') the following holds:*

$$cost(x', y') \leq 3 \cdot cost(x, y) + cost(x^{DS}, y^{DS}).$$

▶ **Lemma 14** (Cost k-means)**.** *For the k-means cost of (x', y') the following holds for $p, q > 0$:*

$$cost^2(x', y') \leq (1 + p^2 + (1 + \frac{1}{p^2})(2 + q^2))\ cost^2(x, y) + (1 + \frac{1}{p^2})(1 + \frac{1}{q^2})\ cost^2(x^{DS}, y^{DS}).$$

3.4 Final fair assignment

In this section, we show how to round the fractional assignments x' to get an integral solution with small additive violation of the group fairness constraint. We show that the resulting solution (x'', y') has the same cost as (x', y').

For this, we use the same min cost flow construction as Bercea et al. use in [4] to round their essentially fair clustering solutions. Similarly to the construction used in Section 2.4 the vertex set V is given by $V = \{s, t\} \cup P \cup \{(i, h) \mid i \in \mathcal{C}_{\text{DS}}, h \in H\} \cup \mathcal{C}_{\text{DS}}$ and the set E of edges is given by $E = A_1 \cup A_2 \cup A_3 \cup A_4$, where $A_1 = \{(s, j) \mid j \in P\}$, $A_2 = \{(j, (i, h) \mid j \in P, i \in \mathcal{C}_{\text{DS}}, h \in H, x'_{ij} > 0\}$, $A_3 = \{((i, h), i) \mid i \in \mathcal{C}_{\text{DS}}, h \in H\}$ and $A_4 = \{(i, t) \mid i \in \mathcal{C}_{\text{DS}}\}$. All edges have unit capacity. A_2 is the only edge set that has nonzero cost. An edge $(p, (i, h)) \in A_2$ has cost $d(p, i)$ in the k-median case and cost $d(p, i)^2$ in the k-means case. The node balances are defined as follows:

- s has balance $|P|$
- All $p \in P$ have balance 0
- All $(i, h) \in \mathcal{C}_{\text{DS}} \times H$ have balance $-(\lfloor \sum_{j \in P_h} x'_{ij} \rfloor)$
- All $i \in \mathcal{C}_{\text{DS}}$ have balance $-(\lfloor \sum_{j \in P} x'_{ij} \rfloor - \sum_{h \in H} \lfloor \sum_{j \in P_h} x'_{ij} \rfloor)$
- t has balance $-(|P| - \sum_{i \in \mathcal{C}_{\text{DS}}} \lfloor \sum_{j \in P} x'_{ij} \rfloor)$

The structure of this min cost flow can be seen in Figure 2 since it is identical to the structure of the max flow instance used in Section 2.4. Since all capacities and balances are integral, we can find an integral solution to the flow instance that has minimal cost. We call the assignment we extract from this MinCostflow-solution x''. We show that it satisfies group fairness with an additive violation of 2:

▶ **Lemma 15** (Group fairness – k-median, k-means)**.** *For each center i and each color h it holds that*

$$l_h \sum_{j \in P} x''_{ij} - 2 \leq \sum_{j \in P_h} x''_{ij} \leq u_h \sum_{j \in P} x''_{ij} + 2.$$

The final solution is then obtained via the pair (x'', y'). We defer the proof that this results in Theorem 5 to the full version.

▶ **Theorem 5.** *There exists a polynomial time algorithm that computes a*

- $(\gamma_{med}+3)$*-approximation with GF-violation of at most* 2 *for the doubly constrained fair* k-median *problem, where* γ_{med} *is the approximation factor of a given approximation algorithm for DS-fair* k*-median.*
- $(\sqrt{1+(\sqrt{\gamma_{means}}+1)^2}+1)^2$*-approximation with GF-violation of at most* 2 *for the doubly constrained fair* k-means *problem, where* γ_{means} *is the approximation factor of a given approximation algorithm for DS-fair* k*-means.*

4 Conclusion

We study the doubly constrained fair k-clustering problem, requiring both group fairness within clusters and diverse center selection among representatives. Via a modular and extendable approach that uses an LP-based rerouting and rounding routine, we obtain a 4-approximation for k-center and the first constant-factor approximations for k-median and k-means in general metrics. Natural next steps include proving nontrivial lower bounds and eliminating the additive violation. Further, developing non-sequential algorithms that enforce both fairness constraints directly might enable better approximation guarantees.

References

1. Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 504–514. ACM, 2021. doi:10.1145/3442188.3445913.
2. Sara Ahmadian, Alessandro Epasto, Ravi Kumar, and Mohammad Mahdian. Clustering without over-representation. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, KDD '19, pages 267–275, New York, NY, USA, 2019. ACM. doi:10.1145/3292500.3330987.
3. Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4955–4966, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/fc192b0c0d270dbf41870a63a8c76c2f-Abstract.html.
4. Ioana Oriana Bercea, Martin Groß, Samir Khuller, Aounon Kumar, Clemens Rösner, Daniel R. Schmidt, and Melanie Schmidt. On the cost of essentially fair clusterings. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, Massachusetts Institute of Technology, Cambridge, MA, USA, September 20-22, 2019*, volume 145 of *LIPIcs*, pages 18:1–18:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.18.
5. Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Fast and accurate fair k-center clustering in doubling metrics. In Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee, editors, *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, pages 756–767. ACM, 2024. doi:10.1145/3589334.3645568.

6 Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Federico Soldà. Scalable and space-efficient robust matroid center algorithms. *J. Big Data*, 10(1):49, 2023. doi:10.1186/s40537-023-00717-4.

7 Moses Charikar, Vincent Cohen-Addad, Ruiquan Gao, Fabrizio Grandoni, Euiwoong Le, and Ernest Van Wijland. An Improved Greedy Approximation for (Metric) k-Means. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 233–240, Los Alamitos, CA, USA, December 2025. IEEE Computer Society. doi:10.1109/FOCS63196.2025.00016.

8 Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016. doi:10.1007/s00453-015-0010-1.

9 Anshuman Chhabra, Karina Masalkovaite, and Prasant Mohapatra. An overview of fairness in clustering. *IEEE Access*, 9:130698–130720, 2021. doi:10.1109/ACCESS.2021.3114099.

10 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5029–5037, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/978fce5bcc4eccc88ad48ce3914124a2-Abstract.html.

11 Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, Chris Schwiegelshohn, and Ola Svensson. A $(2+\epsilon)$-approximation algorithm for metric k-median. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 615–624. ACM, 2025. doi:10.1145/3717823.3718299.

12 John Dickerson, Seyed A. Esmaeili, Jamie Morgenstern, and Claire Jie Zhang. Sok: Fair clustering: Critique, caveats, and future directions. In *IEEE Conference on Secure and Trustworthy Machine Learning, SaTML 2025, Copenhagen, Denmark, April 9-11, 2025*, pages 698–713. IEEE, 2025. doi:10.1109/SaTML64287.2025.00044.

13 John P. Dickerson, Seyed A. Esmaeili, Jamie H. Morgenstern, and Claire Jie Zhang. Doubly constrained fair clustering. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/2ab87e2179b8ea209b52463802d62560-Abstract-Conference.html.

14 Ameet Gadekar. Tight FPT approximations for fair k-center with outliers. *CoRR*, abs/2602.05476, 2026. doi:10.48550/arXiv.2602.05476.

15 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

16 Longkun Guo, Zeyu Lin, Chaoqi Jia, and Chao Chen. Improved streaming algorithm for fair k-center clustering. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 36946–36954. AAAI Press, 2026. doi:10.1609/aaai.v40i43.41022.

17 Anupam Gupta, Benjamin Moseley, and Rudy Zhou. Structural iterative rounding for generalized k-median problems. *Math. Program.*, 212(1):581–634, 2025. doi:10.1007/s10107-024-02119-7.

18 Elfarouk Harb and Ho Shan Lam. KFC: A scalable approximation algorithm for k-center fair clustering. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/a6d259bfbfa2062843ef543e21d7ec8e-Abstract.html.

19 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

20 Xinrui Jia, Kshiteej Sheth, and Ola Svensson. Fair colorful k-center clustering. *Math. Program.*, 192(1):339–360, 2022. doi:10.1007/s10107-021-01674-7.

21 Matthew Jones, Huy L. Nguyen, and Thy Dinh Nguyen. Fair k-centers via maximum matching. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4940–4949. PMLR, 2020. URL: http://proceedings.mlr.press/v119/jones20a.html.

22 Christopher Jung, Sampath Kannan, and Neil Lutz. Service in Your Neighborhood: Fairness in Center Location. In Aaron Roth, editor, *1st Symposium on Foundations of Responsible Computing (FORC 2020)*, volume 156 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FORC.2020.5.

23 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 646–659. ACM, 2018. doi:10.1145/3188745.3188882.

24 Sepideh Mahabadi and Ali Vakilian. Individual fairness for k-clustering. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 6586–6596. PMLR, 2020. URL: http://proceedings.mlr.press/v119/mahabadi20a.html.

25 Huy Le Nguyen, Thy Dinh Nguyen, and Matthew Jones. Fair range k-center. *CoRR*, abs/2207.11337, 2022. doi:10.48550/arXiv.2207.11337.

26 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018*, volume 107 of *LIPIcs*, pages 96:1–96:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.96.

27 Chaitanya Swamy. Improved approximation algorithms for matroid and knapsack median problems and applications. *ACM Trans. Algorithms*, 12(4):49:1–49:22, 2016. doi:10.1145/2963170.

28 Suhas Thejaswi, Ameet Gadekar, Bruno Ordozgoiti, and Aristides Gionis. Diversity-aware clustering: Computational complexity and approximation algorithms. *CoRR*, abs/2401.05502, 2024. doi:10.48550/arXiv.2401.05502.

29 Suhas Thejaswi, Bruno Ordozgoiti, and Aristides Gionis. Diversity-aware k-median: Clustering with fair center representation. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José Antonio Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part II*, volume 12976 of *Lecture Notes in Computer Science*, pages 765–780. Springer, 2021. doi:10.1007/978-3-030-86520-7_47.

30 Ali Vakilian and Mustafa Yalçiner. Improved approximation algorithms for individually fair clustering. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics (AISTATS 2022), 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 8758–8779. PMLR, 2022. URL: https://proceedings.mlr.press/v151/vakilian22a.html.

31 Xiaoliang Wu, Qilong Feng, Junyu Huang, and Jianxin Wang. Parameterized approximation algorithm for doubly constrained fair clustering. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, pages 592–600. ijcai.org, 2025. doi:10.24963/ijcai.2025/67.

32 Xiaoliang Wu, Ting Liang, Junyu Huang, and Qilong Feng. Learning-augmented approximation algorithms for group fair k-center clustering. *Frontiers of Computer Science*, 20:2010408, 2026. doi:10.1007/s11704-025-50627-2.

33 Lunhao Zhang, Pengzhi Gao, and Peng Zhang. Doubly constrained fair clustering for general p-norms. In Fedor V. Fomin and Mingyu Xiao, editors, *Computing and Combinatorics - 31st International Computing and Combinatorics Conference, COCOON 2025, Chengdu, China, August 15-17, 2025, Proceedings, Part I*, volume 15983 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2025. doi:10.1007/978-981-95-0215-8_8.

34 Zhen Zhang, Xiaohong Chen, Limei Liu, Jie Chen, Junyu Huang, and Qilong Feng. Parameterized approximation schemes for fair-range clustering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL: http://papers.nips.cc/paper_files/paper/2024/hash/6ee0ebd0983f0926496b7d4e0d48b8e3-Abstract-Conference.html.

35 Ao Zhao, Yang Zhou, and Qian Liu. Improved approximation algorithms for matroid and knapsack means problems. *International Journal of Foundations of Computer Science*, 36(05):709–729, 2025. doi:10.1142/S012905412246008X.

Near-Linear and Parameterized Approximations for Maximum Cliques in Disk Graphs

Jie Gao
Computer Science Department, Rutgers University, Piscataway, NJ, USA

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Panos Giannopoulos
Department of Computer Science, City St George's, University of London, UK

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, Germany

Satyam Singh
Department of Computer Science, Aalto University, Finland

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Meirav Zehavi
Institute for the Theory of Computing, Ben Gurion University of the Negev, Beer-Sheva, Israel

Abstract

A *disk graph* is the intersection graph of (closed) disks in the plane. We consider the classic problem of finding a maximum clique in a disk graph. For general disk graphs, the complexity of this problem is still open, but for unit disk graphs, it is well known to be in P. The currently fastest algorithm runs in time $O(n^{7/3+o(1)})$, where n denotes the number of disks [19, 28]. Moreover, for the case of disk graphs with t distinct radii, the problem has also recently been shown to be in XP. More specifically, it is solvable in time $O^*(n^{2t})$ [28]. In this paper, we present algorithms with improved running times by allowing for approximate solutions and by using randomization:

- **(i)** for unit disk graphs, we give an algorithm that, with constant success probability, computes a $(1-\varepsilon)$-approximate maximum clique in expected time $\tilde{O}(n/\varepsilon^2)$; and
- **(ii)** for disk graphs with t distinct radii, we give a parameterized approximation scheme that, with a constant success probability, computes a $(1-\varepsilon)$-approximate maximum clique in expected time $\tilde{O}(f(t)\cdot(1/\varepsilon)^{O(t)}\cdot n)$, for some (exponential) function $f(t)$.

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

Keywords and phrases Maximum Clique, Disk Graphs, Unit Disk Graphs, FPT Approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.20

Related Version *Full Version*: `https://arxiv.org/pdf/2512.09899`

Funding *Jie Gao*: Research on this paper was supported by NSF under grants CNS-2515159, IIS-2229876, DMS-2220271, DMS-2311064, and CCF-2118953.
Paweł Gawrychowski: Research on this paper was supported by the Polish National Science Centre under grant 2023/51/B/ST6/01505.
Satyam Singh: Research on this paper was supported by the Research Council of Finland, Grant 363444.
Meirav Zehavi: Research on this paper was supported by he Israel Science Foundation under grant 1470/24, and European Research Council under grant 101039913 (PARAPATH).

Acknowledgements The work in this paper was initiated at the Lorentz Center workshop on Fine-Grained & Parameterized Computational Geometry, held in February 2025.

© Jie Gao, Paweł Gawrychowski, Panos Giannopoulos, Wolfgang Mulzer, Satyam Singh, Frank Staals, and Meirav Zehavi;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

A *disk graph* is the intersection graph of closed disks in the plane, where the vertices are the disks, and two vertices are connected by an edge if and only if the two corresponding disks intersect. A *unit disk graph* is a disk graph in which all disks have the same radius, which can be assumed to be 1. Disk graphs and unit disk graphs are probably the most extensively studied classes of geometric intersection graphs [5, 6, 8, 15, 22, 35]. In particular, they are popular models of wireless communication networks, where each disk represents a wireless station, and the respective radii encode the transmission ranges. Theoretically, unit disk graphs and disk graphs are of special interest, because they generalize the familiar class of planar graphs. Indeed, the well-known circle-packing theorem (also called the the Koebe–Andreev–Thurston theorem [30]) shows that every planar graph is a disk graph. Unlike planar graphs, disk graphs (or unit disk graphs) can be dense, with potentially $\Theta(n^2)$ edges. Thus, algorithms on disk graphs often use an implicit representation (of the coordinates and radii of the disks) and derive edges on-demand.

Due to the special structure of disk graphs, it is an active research direction to study efficient approximation algorithms for classical NP-hard problems on disk graphs or unit disk graphs, such as maximum independent set [12, 25] and minimum dominating set [23, 32]. In this paper, we study the maximum clique problem on unit disk graphs and disk graphs. For a given graph $G = (V, E)$, the problem asks for a vertex set $S \subseteq V$ of maximum cardinality such that the induced subgraph on S is a clique. The maximum clique problem on general graphs is a classical NP-hard problem and is also hard to approximate [20]. However, maximum clique can be solved in polynomial time on unit disk graphs and the problem has been actively studied recently. We first review prior work on this problem and then present our new results.

Related work. Clark, Colbourn, and Johnson [15] gave an elegant $O(n^{4.5})$-time algorithm for finding a maximum clique in a unit disk graph. The algorithm guesses in quadratic time the pair of most distant (diameter) disk centers in a maximum clique and considers the subgraph induced by all the disks whose centers are at most as distant from both candidate diameter centers. The induced graph is co-bipartite, hence computing a maximum clique is equivalent to finding a maximum independent set in the complement bipartite graph. This in turn reduces to finding a maximum matching. By reducing the necessary number of pairs of centers to search for and/or reducing the running time for finding a maximum independent set, the running time has been successively improved to $O(n^{3.5} \log n)$ by Breu [7], $O(n^3 \log n)$ by Eppstein [18], $O(n^{2.5} \log n)$ by Espenant, Keil and Mondal [19] and, most recently, to $O(n^{7/3+o(1)})$ by Chan [11] as noted in Keil and Mondal [28]. In particular, the latter result is based on the clever divide and conquer approach by Espenant, Keil and Mondal [19], which considers only a linear number of center pairs, and on the observation that, for each such pair, one can use a small bi-clique cover of size[1] $\tilde{O}(n^{4/3})$ of the resulting bipartite graph to compute a maximum matching in time almost linear to the size of cover. Very recently, Tkachenko and Wang [35] gave an algorithm that runs in time $O(n \log n + nK^{4/3+o(1)})$, where K is the size of the maximum clique; this is an improvement to the above bound by Chan [11] when $K = o(n)$.

For general disk graphs, the complexity of computing the maximum clique has been a long-standing open problem [3, 19, 22]. Recently, Keil and Mondal [28] considered the case where there are t distinct disk radii and gave an algorithm that runs in polynomial time for

[1] The standard $\tilde{O}(\cdot)$ notation hides polylog factors in n.

every constant t. In particular, the running time is $O(2^t n^{2t}(f(n)+n^2))$, where $f(n)$ is the time to compute a maximum matching in an n-vertex bipartite graph. A maximum matching in a bipartite graph can be computed in time $\tilde{O}(m)$, where m is the number of edges in the graph, using the near-linear time max-flow algorithm in the recent breakthrough in [13] with standard reductions to bipartite matching, e.g., the $O(m \log m)$-time fractional flow rounding technique in [26]; alternatively, it can be computed in time $\tilde{O}(n^2)$ by a combinatorial algorithm [14]. Further, the problem admits a randomized EPTAS[2] with a running time of $2^{\tilde{O}(1/\varepsilon^3)} n^{O(1)}$ and an exact sub-exponential time algorithm [5].

Finally, the problem has also been studied for intersection graphs of other classes of objects in the plane and has been shown to be NP-hard for rays [9], strings [29], ellipses and triangles [3], and for sets of both rectangles and unit disks [6].

Our contribution. The above recent exciting results and the absence of any non-trivial lower bounds for the problem leave some natural open questions.

For improving the currently best $O(n^{7/3+o(1)})$-time bound for unit disk graphs, any algorithm that follows the classical line of approach by Clark, Colbourn, and Johnson [15] would effectively need to reduce the number of candidate disk centers to $o(n)$ or compute a maximum matching in the resulting complement bipartite graph in $o(n^{4/3})$ time. Both of these options seem challenging. On the other hand, the algorithm by Tkachenko and Wang [35] does not bring an improvement for instances with $\Omega(n)$-size cliques, and dense instances are in fact the hardest for the problem, see for example [5].

In this paper, we show that further improvement is possible, albeit by allowing for $(1-\varepsilon)$-approximate solutions and by using randomization. Our first result is as follows:

(i) For unit disk graphs, there is an algorithm that computes with constant success probability a $(1-\varepsilon)$-approximate maximum clique in expected time $\tilde{O}(n/\varepsilon^2)$; this result is presented in Section 3.

In particular, we first random sample two disk centers inside a constant-size neighborhood of a properly defined grid-cell. We show that these centers can help us identify a small region that contains (the centers of) a $(1-\varepsilon)$-approximate maximum clique of the neighborhood, with $\Omega(\varepsilon)$ probability. Moreover, the disks with centers in that small region induce a co-bipartite graph. To save time, our second step, which is given in Section 2, is to actually compute a $(1-\varepsilon)$-approximate maximum clique in this co-bipartite graph in time that is almost linear in the number of disks of the graph.

Our second result is for the general disk graph setting. As mentioned above, the EPTAS in [5] runs in $2^{\tilde{O}(1/\varepsilon^3)} \cdot O(n^4)$ time while, for disk graphs with t distinct radii, the algorithm by Keil and Mondal [28] runs in $O^*(n^{2t})$ time[3]. This raises the question of whether such exponential dependencies w.r.t. t (on n) and ε can be improved or even removed altogether. We show that this is possible by considering parameterized $(1-\varepsilon)$-approximations w.r.t. t and ε and, as in the case of unit disk graphs, by using randomization.

(ii) For disk graphs with t distinct radii, there is an efficient parameterized approximation scheme (EPAS)[4] that computes with a constant success probability a $(1-\varepsilon)$-approximate maximum clique in $\tilde{O}(f(t) \cdot (1/\varepsilon)^{O(t)} \cdot n)$ time, for some (exponential) function $f(t)$; this result is presented in Section 4.

[2] EPTAS stands for efficient polynomial-time approximation scheme.
[3] The standard notation $O^*(\cdot)$ hides polynomial factors.
[4] An efficient parameterized approximation scheme (EPAS), for some parameter k, is a $(1-\varepsilon)$-approximation algorithm that runs in $f(k,\varepsilon)n^{O(1)}$ expected time [21].

Our algorithm is based on the main idea behind the exact algorithm by Keil and Mondal. Suppose that for each of the radii one can guess the leftmost and rightmost disks in an optimal solution. Then, one can find two sets of disks L and R such that $L \cup R$ induces a co-bipartite graph and contains a maximum clique. The algorithm by Keil and Mondal [28] enumerates all possible choices to find the leftmost and rightmost disks for each radius. To speed up the running time for $(1-\varepsilon)$-approximation, we bring in three more ideas. First, we can assume w.l.o.g that for disks of radius r_i that appear in the optimal clique, at least $\Theta(\varepsilon k^*/t)$ of them appear in the max clique, where k^* is the optimal clique size, which can be approximated by a factor of $1/5$ in $O(n \log n)$ time (see Section 4.2) – otherwise, we can skip disks of radius r_i completely without losing more than ε fraction. Second, for each radius r_i we can randomly sample two disks a_i and b_i – if we repeat this enough times, one such pair is close to, in terms of rank, the leftmost and rightmost disks of radius r_i in the optimal solution. Last, as in the case of unit disks, we compute a $(1-\varepsilon)$-approximate clique in the resulting co-bipartite graph using the algorithm from Section 2.

We note here that our result for unit disk graphs is not subsumed by the one for t distinct radii since the latter gives a worse dependency on ε for $t = 1$.

Definitions and notation

Let $\mathcal{D}$ be a set of n disks in the plane, and let $G(\mathcal{D})$ be the *intersection graph* or *disk graph* of $\mathcal{D}$ where

$$G(\mathcal{D}) = (\mathcal{D}, \{\{D, D'\} \mid D, D' \in \mathcal{D}, D \neq D', \text{ and } D \cap D' \neq \emptyset\}).$$

A *unit disk graph* is a disk graph where all disks have unit radius, which we will assume to be 1. We will focus on unit disk graphs and on disk graphs where the disks have t different radii, for some given $t > 1$.

Let $D(c, r)$ be a disk with center c and radius r. If the radius is clear from the context, we may simply write $D(c)$. We refer to the intersection of two disks $D(c, r) \cap D(c', r')$ as their *lens*. The line through c and c' (oriented towards c') splits this lens into two *half-lenses*; a region left of this line and a region right of this line. See Figure 1.

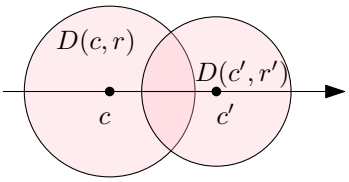


Figure 1 The lens $D(c, r) \cap D(c', r')$ of two intersecting disks $D(c, r), D(c', r')$.

Let a and b be two points in the plane. The Euclidean distance between a and b is denoted by $\text{dist}(a, b)$. The *vertical slab* of a and b is the region between the vertical lines through a and b, that is, the set of all points whose x-coordinates lie between the x-coordinates of a and b.

A graph is *co-bipartite* if its complement is bipartite. Let X and Y be sets of disks such that $G(X)$ and $G(Y)$ are both cliques. As observed in [15], $G(X \cup Y)$ is co-bipartite. Let A be a set. We denote its cardinality by $|A|$. Let $X^* \subseteq \mathcal{D}$ be a maximum-size clique in $G(\mathcal{D})$, i.e., a subset of disks that all pairwise intersect. A clique $X \subseteq \mathcal{D}$ of size at least $(1-\varepsilon)|X^*|$ is a $(1-\varepsilon)$-approximate (maximum) clique.

2 Approximating the Maximum Clique in a co-Bipartite Disk Graph

In this section, we give a near-linear-time algorithm for computing a $(1-\varepsilon)$-approximate clique in a co-bipartite disk graph $G := G(X \cup Y)$. In such a graph, every $x, x' \in X$ and every $y, y' \in Y$ are adjacent, while $x \in X$ and $y \in Y$ are adjacent if and only if the distance between the corresponding centres is at most the sum of the corresponding radii. We begin with the straightforward transformation: a clique in G corresponds to an independent set in the complement of $\overline{G}$, which is bipartite. Thus, it is enough to compute a $(1-\varepsilon)$-approximate independent set in a bipartite graph $\overline{G}$. In this graph, we have an edge (x, y), for $x \in X$ and $y \in Y$, if and only if $\mathrm{dist}(x, y) > r(x) + r(y)$, where $r(\cdot)$ denotes the radius and $\mathrm{dist}(\cdot, \cdot)$ is the distance between the centres.

We first focus on computing the *cardinality* of a $(1-\varepsilon)$-approximate independent set in $\overline{G}$. Recovering the corresponding independent set is a bit more involved and will be explained later. For any graph, the complement of an independent set is a vertex cover. Denoting by n the number of nodes in $\overline{G}$, this immediately gives us that the cardinality of the maximum independent set I^* is equal to n minus the cardinality of the minimum vertex cover C^*. We first observe that a similar property holds for their approximations.

▶ **Proposition 1.** *Let c be an $(1+\varepsilon)$-approximation of $|C^*|$, that is, $|C^*| \le c \le (1+\varepsilon)|C^*|$. Then, $n - c$ is an $(1-\varepsilon)$-approximation of $|I^*|$, that is, $(1-\varepsilon)|I^*| \le n - c \le |I^*|$.*

Proof. Because $|C^*| = n - |I^*|$, we have $n - c \le |I^*|$. We observe that $|I^*| \ge n/2$, as one of the sides of $\overline{G}$ consists of at least $n/2$ nodes forming an independent set. Then, $n - c \ge n - (1+\varepsilon)|C^*| = n - (1+\varepsilon)(n - |I^*|) = |I^*| - \varepsilon \cdot n + \varepsilon|I^*| \ge (1-\varepsilon)|I^*|$. ◀

We now focus on obtaining a $(1+\varepsilon)$-approximation of $|C^*|$. For a bipartite graph, Kőnig's theorem states that the cardinality of the minimum vertex cover is equal to the cardinality of the maximum matching. Thus, denoting by M^* the maximum matching, it is enough to compute a $(1-\varepsilon/2)$-approximation of $|M^*|$, denoted m. Indeed, we can assume $\varepsilon \le 1$, and then $(1-\varepsilon/2)|M^*| \le m \le |M^*|$ implies $|M^*| \le m/(1-\varepsilon/2) \le |M^*|/(1-\varepsilon/2) \le (1+\varepsilon)|M^*|$, so $m/(1-\varepsilon/2)$ can be returned as a $(1+\varepsilon)$-approximation of $|C^*|$.

To design the algorithm, we need an efficient data structure for maintaining a dynamic set of points $S \subseteq \mathbb{R}^2$. Each $s \in S$ has its associated weight $w_s \in \mathbb{R}$, and we define a distance function $\delta : \mathbb{R}^2 \times S \to \mathbb{R}$ as $\delta(p, s) = w_s + \mathrm{dist}(p, s)$. The query returns, for a given point $x \in \mathbb{R}^2$, the furthest point in S, that is, $\arg\max_{s \in S} \delta(x, s)$. This can be seen as a point location query in the furthest site additively weighted Voronoi diagram of S.

▶ **Lemma 2.** *We can maintain a dynamic set of of weighted points S of size n such that an insertion takes amortized expected time $O(\log^2 n)$, a deletion takes amortized expected time $O(\log^4 n)$, and so that a point location query in the furthest site additively weighted Voronoi diagram of S can be answered in worst-case deterministic time $O(\log^2 n)$.*

Proof. As explained by Agarwal, Efrat, and Sharir [2, Section 9.1], the problem reduces to maintaining the upper envelope of a set of n totally defined continuous bivariate functions of constant description complexity in three dimensions, under insertions, deletions, and vertical ray shooting queries. The discussion of Agarwal, Efrat, and Sharir [2, Section 9.1] also shows that the complexity of these upper envelopes (and thus of the corresponding farthest-site Voronoi diagrams) is linear. Thus, the result follows from the data structure given in [27, Theorem 8.3] and its subsequent improvement [31]. ◀

With the above data structure in hand, we can describe the algorithm for computing a $(1-\varepsilon)$-approximate matching in $\overline{G}$. As explained above, returning n minus its cardinality gives us the cardinality of a $(1-\varepsilon)$-approximate clique in G.

▶ **Lemma 3.** *Let G be a co-bipartite disk graph on n vertices. A $(1-\epsilon)$-approximate matching of $\overline{G}$ can be computed in expected $O((n/\varepsilon)\log^4 n)$ time.*

Proof. Let the given co-bipartite graph be $G(X \cup Y)$. We will call X the *left side*, and Y the *right side*. We recall that the algorithm of Hopcroft and Karp [24] proceeds in phases while maintaining the current matching M (initially empty). In the beginning of each phase, we run a breadth-first search to partition the nodes of the graph into *layers*. We start the search from the unmatched nodes on the left side of the graph that form layer 0. On even layers of the search, we can traverse any unmatched edge. On odd layers of the search, we can only traverse a matched edge. The search terminates after the first layer containing an unmatched node on the right side of the graph (or when no further layers can be created, meaning that the current matching M is a maximum matching). This gives us the length k of a shortest augmenting path. Next, the algorithm finds a maximal set of node-disjoint augmenting paths of length k. This can be implemented by running a depth-first search from the unmatched nodes in the last layer. The depth-first search is only allowed to follow an edge leading to a yet unvisited node in the previous layer, and the paths in its tree must alternate between matched and unmatched edges, as in the breadth-first search.

The breadth-first search and the depth-first search run in time linear in the size of the graph, and the length of a shortest augmenting path increases by at least 2 in every phase. It is well-known that if the shortest augmenting path is of length at least $2/\varepsilon$ then the current matching M is a $(1-\varepsilon)$-approximate maximum matching. Thus, it is enough to execute $1/\varepsilon$ phases to obtain such a matching. However, the number of edges in our graph might be even quadratic in n, so to finish the proof we need to show how to implement a single phase in $O(n\log^4 n)$ expected time.

We first explain how to implement the breadth-first search in expected $O(n\log^4 n)$ time. To this end, we maintain an instance of the data structure from Lemma 2, initially containing the set S of all points corresponding to the nodes on the right side of the graph. We set $w_s = -r$ for a point $s \in S$ corresponding to a disk of radius r. Then, on every even layer, we iterate over the nodes there, and for each such node u corresponding to a disk $D(c,r)$, we need to iterate over the yet unvisited nodes v such that (u,v) is an edge. This is equivalent to obtaining (and removing) all points $s \in S$ such that $\delta(c,s) > r$, which in turn can be implemented by repeatedly retrieving $\arg\max_{s\in S}\delta(c,s)$, checking if $\delta(c,s) > r$, and if so, removing s from S and repeating. On every odd layer, we do not need the data structure, as each node has at most one incident matched edge. The overall number of operations on the data structure is $O(n)$, and the deletions dominate, so the total time for the breadth-first search is as claimed. For the depth-first search, we maintain a separate instance of the data structure from Lemma 2 for each even layer ℓ, initially containing the set S_ℓ of all points corresponding to its nodes. Then, if we are currently at node v on an odd layer ℓ, we need to iterate over the yet unvisited nodes u from layer $\ell-1$ such that uv is an edge. Let the disk corresponding to v be $D(c,r)$, Then, this is again equivalent to retrieving (and removing) all points $s \in S_{\ell-1}$ such that $\delta(c,s) > r$, and it can be implemented as above. The overall number of operations on all data structures is $O(n)$, making the total time as required. ◀

For the case of unit disk graphs, Lemma 3 can be improved. For this, we just need a data structure that plays the role of the one in Lemma 2 and that is optimized for unit disk graphs. The following lemma shows how to do this, adapting a method by Efrat, Itai, and Katz [17] to our setting:

▶ **Lemma 4.** *Let S be a set of n unit disks. Then, we can preprocess S in $O(n \log n)$ time into a data structure that supports the following operations:*

- *find(q): given $q \in \mathbb{R}^2$, find a disk in the data structure that does not contain q, or report that no such disk exists (i.e., q lies in the intersection of all the disks in the data structure); and*
- *delete(s): delete the unit disk s from the data structure.*

The time for a find-operation is $O(\log n)$, and the total time to delete all the disks from the data structure is $O(n \log n)$.

Proof sketch. Our data structure is very similar to the structure by Efrat, Itai, and Katz [17, Section 5.1], and we refer the reader to their paper for further details.

Let $\mathcal{G}$ be a regular grid with cells of edge length $1/2$. During the preprocessing phase, we locate for every disk center in S the cell of $\mathcal{G}$ that contains it. Let C be a nonemtpy grid cell, and let S_C be the unit disks whose centers lie in C. Since the diameter of C is less than 1, the disks in S_C have non-empty intersection (e.g., the center of C must lie in all the disks of S_C). We compute the intersection P_C of S_C. The intersection P_C is a convex polygon-like structure whose boundary consists of arcs from S_C, where every disk from S_C contributes at most one arc. The arcs appear along the upper and the lower boundary of P_C in reverse order of the x-coordinates of the corresponding disk centers. Thus, the total complexity of P_C is $O(|S_C|)$, and it can be computed in time $O(|S_C| \log |S_C|)$, using a divide-and-conquer strategy. As in the data structure by Efrat, Itai, and Katz, we compute for each P_C two binary trees that represent the structure of the upper and the lower boundary of P_C, respectively, as in the well-known data structure by Overmars and van Leeuwen [33]. In these trees, the leaves correspond to the disks from S_C, in reverse x-order of their centers. The inner nodes represent the upper (or the lower) boundary of the intersection of the disks in the corresponding subtree. The boundary of each inner node is obtained by intersecting the boundaries of the two children. These boundary curves intersect in a single point. With each inner node, we store the intsersection point of the boundary curves of its children, as well as the part of the boundary curve for this node that does not appear on the parent boundary . The binary trees can be computed in $O(n \log n)$ time, and the total time for the preprocessing phase is thus $O(n \log n)$ (see the paper by Efrat, Itai, and Katz for the details).

A find-query can be answered as follows: given $q \in \mathbb{R}^2$, we first check if there is a non-empty grid cell C that is far enough from q such that q must lie outside of P_C. If such a cell C exists, we report an arbitrary disk from P_C and are done (this takes $O(\log n)$ time, assuming that the nonempty grid cells are stored in a suitable data structure). Otherwise, there is a constant number of remaining candidate cells, and for each such candidate cell C, we determine whether q lies outside of P_C. This can be done using the tree structures for C to determine whether q lies above or below the upper and the lower boundary of P_C, by descending into the binary tree, using the intersection points of the child boundaries as a guidance. If q lies outside of P_C, we can use the result of the tree-searches to obtain a disk in S_C that does not contain q, as desired. Again, this takes $O(\log n)$ time.

To delete a a disk s from S, we find the cell C that contains the center of s, and we delete s from S_C. For this, we must update the tree structures that represent the intersection P_C. The details for the algorithm and its analysis are given by Efrat, Itai, and Katz. Briefly, the idea is to find the parts of the boundary that are occluded by the deleted disk, and to promote these parts up in the tree. The main observation is that every piece of the boundary can rise only upwards into the tree, and that the tree has only $O(\log n)$ levels, resulting in a total deletion time of $O(n \log n)$. ◀

Using Lemma 4, we get the following improved version of Lemma 3 for unit disk graphs:

▶ **Lemma 5.** *Let G be a co-bipartite unit disk graph on n vertices. A $(1-\epsilon)$-approximate matching of $\overline{G}$ can be computed in expected $O((n/\varepsilon)\log n)$ time.*

Proof. The proof is identical to the proof of Lemma 3, but using the data structure from Lemma 4 instead of Lemma 2. ◀

Lemma 3 and Lemma 5 allow us to compute the cardinality of the sought $(1-\varepsilon)$-approximate clique. Computing the corresponding subset of nodes is a bit more involved.

▶ **Theorem 6.** *Let G be a co-bipartite disk graph with n vertices. We can compute a $(1-\varepsilon)$-approximate clique of G in expected $O((n/\varepsilon)\log^4 n)$ time. When G is a unit disk graph, such a clique can be computed in expected $O((n/\varepsilon)\log n)$ time.*

Proof. We provide the analysis for general disk graphs. The case of unit disk graphs is handled in the same way, using the faster data structure from Lemma 4.

Our strategy is to find a $(1+\varepsilon)$-approximate vertex cover in $\overline{G}$ in $O((n/\varepsilon)\log^4 n)$ expected time, and we return its complement. By Proposition 1, this constitutes a $(1-\varepsilon)$-approximate clique in G.

To obtain the approximate vertex-cover, we proceed as follows: recall that X and Y are the nodes on the left and on the right side of G. As in the proof of Lemma 3, we start with executing $\ell = 1/\varepsilon$ phases of the Hopcroft-Karp algorithm. This takes expected time $O((n/\varepsilon)\log^4 n)$, and it produces a $(1-\varepsilon)$-approximate maximum matching M. Next, we again run a breadth-first search to partition the nodes into layers. Now, however, we do not need to terminate the search after reaching the first layer containing an unmatched node of Y. The BFS gives us a partition of X and Y into subsets

$$X_0, Y_0, X_1, Y_1, X_2, Y_2, \ldots$$

corresponding to the consecutive layers, where $X_0, X_1, \cdots \subseteq X$, and $Y_0, Y_1, \cdots \subseteq Y$, as well as the remaining part $X' = X \setminus \bigcup_i X_i$ and $Y' = Y \setminus \bigcup_i Y_i$. Because we have run ℓ phases of the Hopcroft-Karp algorithm, the layers $Y_0, Y_1, \ldots, Y_{\ell-1}$ do not contain any unmatched nodes, and hence $\sum_i |Y_i| = |M|$. We choose $i \in \{0, 1, \ldots, \ell-1\}$ such that $|Y_i| \leq |M|/\ell$, and we define the following set K:

$$K := (X \setminus (X_0 \cup X_1 \cup \cdots \cup X_i)) \cup (Y_0 \cup Y_1 \cup \cdots \cup Y_i).$$

We will now establish that K is a vertex cover for G, with size $(1+\varepsilon)|M|$.

Consider an edge $e = xy$, where $x \in X$ and $y \in Y$. We need to show that $x \in K$ or $y \in K$. If $x \in K$, we are done. Thus, suppose that $x \notin K$, and we need to establish that $y \in K$. Since $x \notin K$, we have $x \in X_j$, for some $j \in \{0, 1, \ldots, i\}$. We consider two cases.

- **Case 1: $e \in M$**: Because $x \in X_j$ is matched, we must have $j \geq 1$ and we have reached x through an augmenting path ending with e. Thus, $y \in Y_{j-1} \subseteq K$.
- **Case 2: $e \notin M$**: We can extend an augmenting path ending at x with e to obtain an augmenting path ending at y. Thus, $y \in Y_0 \cup Y_1 \cup \cdots \cup Y_j \subseteq K$.

This shows that K is a vertex cover.

We now move to analyzing the size of K. First, we argue that every node in K is an endpoint of an edge in M. We prove this separately for the nodes in X and in Y.

- **Case 1: $x \notin X_0 \cup X_1 \cup \cdots \cup X_i$:** Such a node is matched, as it does not belong to X_0.
- **Case 2: $y \in Y_0 \cup Y_1 \cup \cdots \cup Y_i$:** None of the sets $Y_0, Y_1, \ldots, Y_{\ell-1}$ contain an unmatched node, so such a node is matched.

Next, we consider the situation when an edge $xy \in M$ has two endpoints in K. Then, $x \notin X_0 \cup X_1 \cup \cdots \cup X_i$ and $y \in Y_j$, for some $j \in \{0, 1, \ldots, i\}$. Now, it is possible to extend an augmenting path ending at y by the matched edge yx to obtain an augmenting path ending at x, so $x \in X_{j+1}$. Thus, $j = i$, and the considered matched edge connects $x \in X_{i+1}$ and $y \in Y_i$. By the choice of i, there are at most $|M|/\ell$ such edges. This allows us to bound the number of nodes in K by $|M| + |M|/\ell = (1 + \varepsilon)|M|$.

Together, this shows that K is an $(1 + \varepsilon)$-approximate vertex cover.

As in Lemma 3, running the breadth-first search takes $O(n \log^4 n)$ expected time. After that, constructing K can easily be done in additional linear time. ◀

3 Unit Disk Graphs

We describe an algorithm that computes an $(1 - \varepsilon)$-approximate clique in a unit disk graph with probability at least $1 - \delta$ in $O((n/\varepsilon^2) \log(n) \log(1/\delta))$ expected time. Let P be a set of n points in the plane, and let $\mathcal{D}(P) = \{D(p, 1) \mid p \in P\}$ be the set of unit disks whose centers are given by P. For convenience, we will sometimes identify disks with their centers. A set $X \subseteq P$ is said to form a clique if all the disks in $\mathcal{D}(X)$ intersect pairwise.

Let $\mathcal{G}$ be a regular grid where each grid cell has diameter 2, and let $\mathcal{C}$ be the collection of grid cells that contain at least one point from P. Let $C \in \mathcal{C}$ be such a nonempty cell. The *extension* of C, denoted by C^+, is the 5×5-block of grid cells whose central cell is C. Furthermore, let $P_C \subseteq P$ be the set of disk centers that lie in any cell of the extension C^+; for an illustration, see Figure 2. Each point of P_C is uniquely assigned to exactly one cell of C^+: if a point lies on the boundary shared by multiple cells of C^+, it is assigned arbitrarily to one of them. Then, a simple diameter argument shows that every clique in P must be contained within the extension of some nonempty cell.

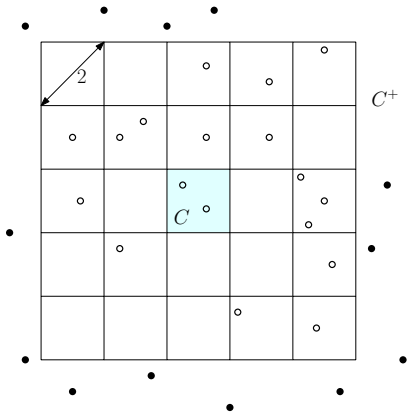


Figure 2 A cell C (the central cell) and its extension C^+, where each cell in C^+ has a diameter 2. All points (both solid and hollow) belong to P, while the hollow points also belong to P_C.

▷ Claim 7. Let X be a clique in P. Then, there exists a cell $C \in \mathcal{C}$ such that $X \subseteq P_C$.

Proof. Let $p \in X$, and let $C \in \mathcal{C}$ be the cell that contains p. Since every other disk center in X has distance at most 2 from p, it follows that X must lie in C^+. Hence, we have $X \subseteq P_C$, as claimed. ◁

Furthermore, a simple packing argument shows that every extension of a nonempty cell C contains a clique of size linear in $|P_C|$.

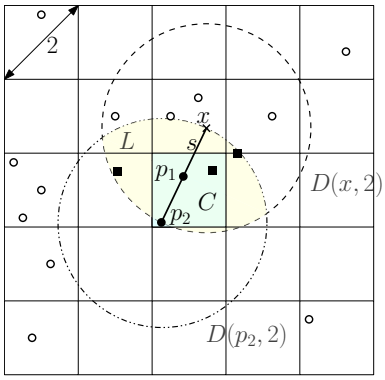


Figure 3 The cell C is represented by the light-blue colored cell with $p_1, p_2 \in C$. The thick black line denotes the line segment $s = \overline{p_2 x}$ of length 2, passing through p_1. The dotted disk represents $D(x, 2)$, while the dash-dotted disk represents $D(p_2, 2)$. The yellow-shaded region corresponds to the lens $L = D(x, 2) \cap D(p_2, 2)$, formed along the axis s.

▷ Claim 8. Let X^* be a maximum clique in P_C. Then, we have $|X^*| \geq |P_C|/25$.

Proof. The extension C^+ consists of 25 grid cells with pairwise disjoint interiors, where each cell has diameter 2. Hence, for any such cell $C' \subset C^+$, the set $P_C \cap C'$ must form a clique, since all these disks are pairwise intersecting. Recall that if a disk center lies on the boundary of a cell, it is assigned arbitrarily to one of the adjacent cells so that each point in P_C is uniquely assigned to exactly one cell in C^+. Since there are 25 such cells, there exists at least one cell in C^+ containing at least $|P_C|/25$ disk centers. It follows that P_C contains a clique of size at least $|P_C|/25$, and therefore the maximum clique X^* is at least of size $|P_C|/25$. ◁

Now, let $C \in \mathcal{C}$. We select two disk centers $p_1, p_2 \in P_C$ independently and uniformly at random. Let s be the line segment of length 2 that starts at p_2 and is directed toward p_1, so that p_1 lies either on s or on its extension along the same direction. Let x be the other endpoint of s, and let $L = D(x, 2) \cap D(p_2, 2)$ be the lens with axis s; see Figure 3 for an illustration. We determine the set $P_L = L \cap P_C$ of disks centers that lie in L, and we find an $(1 - \varepsilon/2)$-approximate maximum clique X for P_L, using Theorem 6 from Section 2. The lens L is partitioned by segment s into two parts: the part to the left of s and the part to the right of s. Without loss of generality, the points on s belong to the left part. Any two points in the same part have a distance at most 2. Thus, all disks with centers in the same part form a clique. In other words, the intersection graph of unit disks centered at P_L is a co-bipartite graph. Consequently, as we show below, the clique X is an $(1 - \varepsilon)$-approximate clique for P_C with probability $\Omega(\varepsilon)$.

▶ **Lemma 9.** *Let X^* be a maximum clique in P_C. Then, with probability $\Omega(\varepsilon)$, we have $|X| \geq (1 - \varepsilon)|X^*|$.*

Proof. First, by Claim 8, we know that $|X^*| \geq |P_C|/25$. Thus, with probability at least $1/25$, we have that $p_1 \in X^*$. Now, we sort the centers in X^* in the increasing order of their distances from p_1, and let π^* be the resulting ordering of X^*. Let X^+ be those centers from X^* that have rank at least $(1 - \varepsilon/2)|X^*|$ in π^*, and let $X^- = X^* \setminus X^+$ (in other words, X^+ contains the $(\varepsilon/2)|X^*|$ points in X^* that are furthest away from p_1, and X^- contains the $(1 - \varepsilon/2)|X^*|$ remaining points). Since

$$|X^+| = \frac{\varepsilon}{2}|X^*| \geq \frac{\varepsilon}{50}|P_C|,$$

it follows that with probability at least $\varepsilon/50$, the second sampled center p_2 lies in X^+.

Since p_1 and p_2 are sampled independently, we can conclude that with probability at least $\varepsilon/1250$, it holds that $p_1 \in X^*$ and that $p_2 \in X^+$. Henceforth, we assume that indeed this is the case. Since p_1 and p_2 both lie in X^*, the distance $\mathrm{dist}(p_1, p_2)$ between p_1 and p_2 is at most 2. From this, it follows that $D(p_1, \mathrm{dist}(p_1, p_2)) \subseteq D(x, 2)$ since by the choice of x, we have $\mathrm{dist}(x, p_1) + \mathrm{dist}(p_1, p_2) = 2$, which implies that for every point $y \in D(p_1, \mathrm{dist}(p_1, p_2))$, the triangle inequality gives

$$\mathrm{dist}(x, y) \leq \mathrm{dist}(x, p_1) + \mathrm{dist}(p_1, y) \leq \mathrm{dist}(x, p_1) + \mathrm{dist}(p_1, p_2) = 2,$$

as needed.

As $p_2 \in X^+$, and since all the points in X^+ are at least as far away from p_1 as all the points in X^-, it follows that $X^- \subseteq D(p_1, \mathrm{dist}(p_1, p_2)) \subseteq D(x, 2)$. Moreover, since $X^- \subseteq X^*$ and $p_2 \in X^*$, we also have $X^- \subseteq D(p_2, 2)$. This implies that X^- lies in the lens L defined by $D(x, 2)$ and $D(p_2, 2)$. In particular, P_L contains a clique of size $(1 - \varepsilon/2)|X^*|$, and since X is a $(1 - \varepsilon/2)$-approximate clique for P_L, we have

$$|X| \geq (1 - \varepsilon/2)(1 - \varepsilon/2)|X^*| = (1 - \varepsilon + \varepsilon^2/4)|X^*| \geq (1 - \varepsilon)|X^*|.$$

This means that X is an $(1 - \varepsilon)$-approximate maximum clique for P_C, as claimed. ◀

We can now present the complete algorithm.

▶ **Theorem 10.** *Let P be a set of n points in the plane, let $\varepsilon > 0$ be a parameter. There is a randomized algorithm that runs in expected time $O((n/\varepsilon^2) \log n)$ and computes an $(1 - \varepsilon)$-approximate clique for $\mathcal{D}(P)$ with probability at least $1/2$.*

Proof. Let $C \in \mathcal{C}$ be a nonempty grid cell. By Lemma 9, and using Theorem 6, one can compute, in $O((|P_C|/\varepsilon) \log n)$ expected time, a clique for P_C that, with probability $\Omega(\varepsilon)$, is an $(1 - \varepsilon)$-approximate clique for P_C. By repeating this process $O(1/\varepsilon)$ times, and selecting the largest of the resulting cliques, one can compute in $O((|P_C|/\varepsilon^2) \log n)$ expected time, a clique that is an $(1 - \varepsilon)$-approximate clique for P_C with probability at least $1/2$.

We do this for P_C of every nonempty cell $C \in \mathcal{C}$, and we return the largest of the resulting cliques. (Note that all non-empty cells can be easily found in $O(n \log n)$ time [36]). By Claim 7, there exists a cell $C \in \mathcal{C}$ such that P_C contains a max-clique of P, so with probability at least $1/2$, this gives an $(1 - \varepsilon)$-approximate maximum clique for P. The total running time is proportional to

$$\sum_{C \in \mathcal{C}} \frac{|P_C|}{\varepsilon^2} \log n = \frac{\sum_{C \in \mathcal{C}} |P_C|}{\varepsilon^2} \log n = O((n/\varepsilon^2) \log n),$$

since every point from P appears in $O(1)$ sets P_C. The theorem follows. ◀

By a standard argument, for any value $\delta > 0$, we can raise the probability of success to $1 - \delta$ by independently repeating the algorithm of Theorem 10 $O(\log(1/\delta))$ times and by taking the best solution. This increases the running time by an $O(\log(1/\delta))$ factor.

4 Disk graphs with different radii

We first review the algorithm by Keil and Mondal [28]. The main idea is simple and elegant, and it goes as follows. Suppose that there are t distinct radii $r_1, \ldots, r_t$, in increasing order. One can enumerate all radii that appear in a maximum clique and, for each radius r_i, enumerate all disk pairs to find the one with centers a_i, b_i that are, respectively, leftmost and

rightmost (in the optimum) along the x-axis. Let X be the set of these $2t$ disks. For each pair of centers a_i, b_i, consider their vertical slab and find the set L_i of all radius-r_i disks that intersect all disks in X and have their centers within the slab and above the line segment a_ib_i. Similarly, find the set R_i of all radius-r_i disks that intersect all disks in X and have their centers in the slab and below the line segment a_ib_i. Take $L = L_1 \cup L_2 \cup \cdots \cup L_t$ and $R = R_1 \cup R_2 \cup \cdots \cup R_t$. A simple geometric argument shows that the graph induced by $L \cup R$ is co-bipartite, and thus, similar to the case of unit disks, one can find a maximum clique by finding a maximum matching. The algorithm runs in time $O(2^t n^{2t}(f(n) + n^2))$, where $f(n)$ is the time to compute a maximum matching in an n-vertex bipartite graph. The factor n^{2t} comes from the enumeration to find the leftmost and rightmost disks in the optimal solution for every radius r_i.

In this section, we give an efficient parameterized approximation scheme w.r.t. the number t of different radii and ε. There are two places where we can speed up the algorithm described above if approximate solutions are allowed. First, instead of finding the leftmost and rightmost disks for each radius by enumeration, we use random sampling to find two centers a_i, b_i that are close (in terms of rank) to the leftmost and rightmost disks in the optimal clique. Second, as in the case of unit disk graphs, we can replace finding a maximum clique by finding an approximate maximum clique using Theorem 6.

Let $\mathcal{D}_i$ be the set of disks of radius r_i and let D^* be a maximum clique, with $|\mathcal{D}^*| = k^*$. In the next subsection, we will assume that $\mathcal{D}^*$ contains at least εk disks from each $\mathcal{D}_i$, for some parameter $k \in \mathbb{N}$. Later, in Section 4.2, we will drop this assumption using brute-force.

4.1 A special case: Disks of every radius contribute to the maximum clique

In this section, we will compute an $(1-\varepsilon)$-approximate maximum clique that has at least one disk from each radius.

Our algorithm is simple: we (independently) pick a disk $D(o, r_1)$ uniformly at random $m_1 = \Theta(n/k\varepsilon)$ times and compute the subset of disks $\mathcal{D}'$ that intersect $D(o, r_1)$, where r_1 is the smallest radius. With each choice of $D(o, r_1)$ and the corresponding $\mathcal{D}'$, we repeat the following procedure $m_2 = \Theta((k^*/k\varepsilon^2)^{2t})$ times:

1. For each radius r_i: uniformly at random pick two disks $D(a_i, r_i)$ and $D(b_i, r_i)$ from $\mathcal{D}'_i = \mathcal{D}_i \cap \mathcal{D}'$. Let X denote the set of the selected disks for all r_i. Without loss of generality, assume that a_i is to the left of b_i. We interpret a_i as the leftmost disk of radius r_i in the clique, and b_i as the rightmost disk in the clique.
2. Compute the subset of disks $\mathcal{D}'' \subseteq \mathcal{D}'$ that intersect all disks in X (by explicitly checking if a disk $D(c, r) \in \mathcal{D}'$ intersects all disks in X).
3. Form two sets of disks L and R. For each vertical slab with a_i, b_i on the boundary, find all the radius-r_i disks $L_i \subseteq \mathcal{D}''$ with center within the slab and above the line segment a_ib_i. Similarly, find all the radius-r_i disks $R_i \subseteq \mathcal{D}''$ with centers in the slab and below the line segment a_ib_i. Take $L = L_1 \cup L_2 \cup \cdots \cup L_t$ and $R = R_1 \cup R_2 \cup \cdots \cup R_t$.
4. Use the algorithm from Theorem 6 to compute an $(1-\varepsilon/2)$-approximate clique in $G(L \cup R)$. Note that this is possible, as $G(L \cup R)$ is co-bipartite, see Lemma 12 below.

We finally return the largest clique among all the repeated $m_1 m_2$ runs.

For the analysis, we start with the following observation.

▶ **Lemma 11.** *For a disk $D(o, r_1)$, let $\mathcal{D}'$ be the set of disks that intersect $D(o, r_1)$, Then, $|\mathcal{D}'| \leq 6k^*$.*

Proof. Consider the six cones of angle $\pi/3$ that partition $\mathbb{R}^2$ and have their apex at o. Any two disks $D(c_i, r_i)$ and $D(c_j, r_j)$ with the centers in the same cone and such that both intersect $D(o, r_1)$ must also pairwise intersect. To see that, we have $\text{dist}(o, c_i) \leq r_1 + r_i$ and $\text{dist}(o, c_j) \leq r_1 + r_j$. We also have that $\angle c_i o c_j \leq \pi/3$. Thus, $\text{dist}(c_i, c_j) \leq \max\{\text{dist}(o, c_i), \text{dist}(o, c_j)\} \leq r_i + r_j$, since $r_1 \leq r_i, r_j$. Hence, the disks from $\mathcal{D}'$ that are in a single cone form a clique. There must be a cone containing at least $|\mathcal{D}'|/6$ centers from $\mathcal{D}'$, and, consequently, we have $k^* \geq |\mathcal{D}'|/6$. ◀

The probability that by random sampling one selects a specific disk $D(o, r_1)$ in $\mathcal{D}^*$ is at least $\varepsilon k/n$. Since we repeat the algorithm $m_1 = \Theta(n/k\varepsilon)$ times, with a constant probability $D(o, r_1) \in \mathcal{D}^*$. In the following, we assume that $D(o, r_1)$ indeed appears in $\mathcal{D}^*$, and then use a similar argument as in the unit disk case to argue that from each radius r_i, we discard at most an $\varepsilon/2$-fraction of the disks from $\mathcal{D}^* \cap \mathcal{D}_i$ in our clique.

Let $c_1, \ldots, c_h$ be the centers of the disks in $\mathcal{D}^* \cap D_i$, ordered from left to right, and note that by our assumption on $\mathcal{D}^*$, we have $h \geq \varepsilon k$. By Lemma 11, the probability that $a_i \in \{c_1, \ldots, c_{(\varepsilon/4)h}\}$ is

$$\frac{(\varepsilon/4)h}{|D_i'|} \geq \frac{(\varepsilon/4)h}{|D'|} \geq \frac{\varepsilon h}{24k^*} \geq \frac{\varepsilon^2 k}{24k^*}.$$

Similarly, the probability that b_i is among the rightmost $(\varepsilon/4)h$ centers of $\mathcal{D}^* \cap \mathcal{D}_i$ is also at least $\varepsilon^2 k/24k^*$.

The following technical lemma, stated for our setting, asserts that we include at least $h - 2(\varepsilon/4)h = (1 - \varepsilon/2)h$ disks from $\mathcal{D}^* \cap \mathcal{D}_i$ in $L \cup R$.

▶ **Lemma 12.** *Assume that $D(o, r_1)$ and $D(a_i, r_i)$, $D(b_i, r_i)$, for all i are from $\mathcal{D}^*$. The disks in L form a clique. Similarly, the disks in R form a clique. Further, for every i, all disks in $\mathcal{D}^*$ of radius r_i with centers within the vertical slab within a_i and b_i are included in $L \cup R$.*

Proof. The proof that L and R form a clique respectively follows directly from [28, Lemma 3.1]. Next, all disks in D^* intersect $D(o, r_1)$, and $D(a_i, r_i), D(b_i, r_i)$, for all i. Thus the disks of radius r_i of D^* with the conditions specified in the lemma meet the requirements for L_i or R_i and thus are included in $L \cup R$. ◀

Hence, with probability at lest $(\varepsilon^2 k/24k^*)^2$ we include at least a $(1 - \varepsilon/2)$ fraction of the disks of $\mathcal{D}^* \cap D_i$ in $L \cup R$. The probability that we include $(1 - \varepsilon/2)$ fraction of all disks (e.g. over all t radii) is thus at least $(\varepsilon^2 k/24k^*)^{2t}$.

We now compute a $(1 - \varepsilon/2)$-approximate clique in $G(L \cup R)$, so the size is at least $(1 - \varepsilon/2)(1 - \varepsilon/2)k^* \geq (1 - \varepsilon)k^*$.

Repeating this $m_2 = \Theta((k^*/k\varepsilon^2)^{2t})$ times, the algorithm succeeds with a constant probability. If $k = \Theta(k^*)$, then $m_2 = O(1/\varepsilon^{4t})$.

Now we can conclude with the full algorithm and its runtime.

▶ **Lemma 13.** *With a constant success probability, we can compute a $(1 - \varepsilon)$-approximate maximum clique of n disks of t distinct radii, in which every radius appears at least εk times, in expected time*

$$O\left(n \log n + \frac{n}{k\varepsilon}(t \log n + k^*) + \frac{nk^*}{k\varepsilon}\left(\frac{k^*}{k\varepsilon^2}\right)^{2t}\left(t + \frac{\log^4 k^*}{\varepsilon}\right)\right)$$

where k^* is the size of the maximum clique.

Proof. For each disk $D(o, r_1)$, finding the disks that intersect it can be done easily in $O(n)$ time. But if we repeat this step $m_1 = \Theta(n/k\varepsilon)$ times, this will be too costly. Instead, we pre-process the disks into a data structure and answer a 2D circular range query, which returns the points within a query disk. In particular, for each radius r_i we process the centers of the disks in $\mathcal{D}_i$ into a size $O(|\mathcal{D}_i|)$ data structure using the construction in [1] such that one can report all centers of the disks in $\mathcal{D}_i$ within distance $r_1 + r_i$ from o in $O(\log n + \ell_i)$ time, where ℓ_i is the size of the output. The construction time for the data structure for $\mathcal{D}_i$ is $O(|\mathcal{D}_i| \log |\mathcal{D}_i|)$. Thus total construction time for all t different radii is $O(n \log n)$. We answer tm_1 queries and the total query time is $O(tm_1 \log n + m_1 k^*)$.

Note that Steps 1–4 of our algorithm operate on $\mathcal{D}'$ for each choice of $D(o, r_1)$ with $|\mathcal{D}'| = O(k^*)$ disks. Thus, the running time for Steps 1–4 is $O(2tk^* + f_{\varepsilon/2}(k^*))$, where $f_\varepsilon(n)$ is the time for computing a $(1-\varepsilon)$-approximate clique in an n-by-n co-bipartite disk graph. Applying Theorem 6, we have $f_\varepsilon(n) = O((n/\varepsilon)\log^4 n)$ expected time, so we have one iteration of Steps 1–4 to be $O(tk^* + (k^*/\varepsilon)\log^4 k^*)$. Summing all the computation costs, we have,

$$O\left(n \log n + \frac{n}{k\varepsilon}(t \log n + k^*) + \frac{nk^*}{k\varepsilon}\left(\frac{k^*}{k\varepsilon^2}\right)^{2t}\left(t + \frac{\log^4 k^*}{\varepsilon}\right)\right). \qquad \blacktriangleleft$$

4.2 The complete algorithm

▶ **Theorem 14.** *Let $\mathcal{D}$ be a set of n disks, let t be the number of distinct radii in $\mathcal{D}$, and let $\varepsilon \in (0, 1)$ be a parameter. With a constant success probability, we can compute a $(1-\varepsilon)$-approximate maximum clique in $G(\mathcal{D})$ in expected time $\tilde{O}\left((\frac{t}{\varepsilon})^{O(t)} n\right)$.*

Proof. First, we compute a 1/5-approximate maximum clique $\hat{D}$. This can be achieved in $O(n \log n)$ time as follows. It is well-known that every set of pairwise intersecting disks can be stabbed by four points [16, 34]. This implies that there exists a point that is covered by at least 1/4 of the disks in a maximum clique[5]. Thus, a point of maximum depth is covered by at least so many disks as well. Computing a point of maximum depth in an arrangement of disks is 3SUM-hard [4]; however, a point of $(1-\varepsilon)$-approximate maximum depth can be computed, with high probability, in $O((1/\varepsilon)^2 n \log n)$ expected time [4]. We therefore run this algorithm for $\varepsilon = 1/2$ and get the approximate clique of the desired size. Thus, we take $k = |\hat{D}|$, where $k^*/5 \le k \le k^*$.

Consider the optimal solution D^* with k^* disks and assume that all disks of radius r_i are removed from D^* if there are at most $\varepsilon k^*/(2t)$ of them in it. This leaves a clique $\bar{D} \subseteq D^*$ with $(1-\varepsilon/2)k^* \le |\bar{D}| \le k^*$. Note that $\bar{D}$ has at least $\varepsilon k^*/2t \ge \varepsilon k/2t$ of disks in radius r_i, if r_i appears at all. We now aim to find a $1-\varepsilon/5$ approximation to $\bar{D}$, which is a $1-\varepsilon$ approximation to D^*. Of course, we do not know in advance what those radii are that appear less than $\varepsilon k^*/(2t)$ in D^*, which is some (fixed but unknown) optimal solution, and therefore we will brute-force through all subsets of radii, at the cost of an extra factor of 2^t in the running time.

Let $\mathcal{R}$ denote the ordered set of distinct radii appearing in $\mathcal{D}$. For each subset R of radii, we run the algorithm from Lemma 13 with parameters $\varepsilon/2$ and k/t. We output the best solution. The correctness and the approximation factor follow immediately from Lemma 13

[5] Carmi, Katz and Morin [10] showed that given a set of pairwise intersecting disks one can compute a set of four stabbing points in linear time but it's not obvious how to find a point as above in the same time.

and the fact that we enumerate all radius subsets, one of which matches the set of radii that appear more than $\varepsilon k^*/(2t)$ times in the optimal solution D^*. For that particular run, Lemma 13 guarantees that we find a $1-\varepsilon$ approximation to D^* with a constant probability.

The running time comes from repeating Lemma 13 with parameters $\varepsilon/2$ and k/t, for 2^t times. Notice that the construction of the circular range query data structure is only done once at the initialization step. But all other steps will be repeated 2^t times. Further, $k^*/5 \leq k \leq k^*$. The total running time is obtained from applying Lemma 13.

$$O\left(n\log n + \frac{2^t t^2 n\log n}{k^*\varepsilon} + \frac{t2^t n}{\varepsilon} + \frac{2^{7t}t^{2t+1}n\log^4 k^*}{\varepsilon^{4t+2}} + \frac{nt2^{4t+1}}{\varepsilon^{4t+1}}\right) = \tilde{O}\left(\left(\frac{t}{\varepsilon}\right)^{O(t)} n\right). \quad \blacktriangleleft$$

As in the case of unit disks, we can increase the probability of success to any desired value by independently repeating the algorithm sufficiently many times.

References

1. Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA 2009)*, pages 180–186, 2009. doi:10.1137/1.9781611973068.21.
2. Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 2000. doi:10.1137/S0097539795295936.
3. Christoph Ambühl and Uli Wagner. The clique problem in intersection graphs of ellipses and triangles. *Theory Comput. Syst.*, 38(3):279–292, 2005. doi:10.1007/S00224-005-1141-6.
4. Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008. doi:10.1137/060669474.
5. Marthe Bonamy, Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Panos Giannopoulos, Eun Jung Kim, Paweł Rzążewski, Florian Sikora, and Stéphan Thomassé. EPTAS and subexponential algorithm for maximum clique on disk and unit ball graphs. *J. ACM*, pages 1–38, 2021. doi:10.1145/3433160.
6. Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *LIPIcs*, pages 17:1–17:18, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.17.
7. Heinz Breu. *Algorithmic Aspects of Constrained Unit Disk Graphs*. PhD thesis, Department of Computer Science, University of British Columbia, 1996. doi:10.14288/1.0051600.
8. Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1):3–24, 1998. Special Issue on Geometric Representations of Graphs. doi:10.1016/S0925-7721(97)00014-X.
9. Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discret. Comput. Geom.*, 50(3):771–783, 2013. doi:10.1007/s00454-013-9538-5.
10. Paz Carmi, Matthew J Katz, and Pat Morin. Stabbing pairwise intersecting disks by four points. *Discret. Comput. Geom.*, pages 1751–1784, 2023. doi:10.1007/s00454-023-00567-0.
11. Timothy M Chan. Personal communication with the authors of [28], 2023. arXiv:2404.03751.
12. Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.
13. Li Chen, Rasmus Kyng, Yang Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *J. ACM*, 72(3):19:1–19:103, 2025. doi:10.1145/3728631.

14 Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, (STOC 2024)*, pages 83–94, 2024. doi:10.1145/3618260.3649725.

15 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discret. Math.*, 86(1):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

16 Ludwig W. Danzer. Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene. *Stud. Sci. Math. Hungar.*, 21(1-2):111–134, 1986.

17 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001. doi:10.1007/S00453-001-0016-8.

18 David Eppstein. Graph-theoretic solutions to computational geometry problems. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2009)*, volume 5911 of *LNCS*, pages 1–16, 2009. doi:10.1007/978-3-642-11409-0_1.

19 Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In *Proceedings of the 39th International Symposium on Computational Geometry, (SoCG 2023)*, volume 258 of *LIPIcs*, pages 30:1–30:17, 2023. doi:10.4230/LIPIcs.SOCG.2023.30.

20 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost NP-complete (preliminary version). In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, (FOCS 1991)*, pages 2–12, 1991. doi:10.1109/SFCS.1991.185341.

21 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and Algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/A13060146.

22 Aleksei V. Fishkin. Disk graphs: A short survey. In *Proceedings of the 1st International Workshop on Approximation and Online Algorithms, (WAOA 2003)*, volume 2909 of *LNCS*, pages 260–264, 2003. doi:10.1007/978-3-540-24592-6_23.

23 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *Proceedings of the 18th Annual European Symposium on Algorithms, (ESA 2010), Part I*, volume 6346 of *LNCS*, pages 243–254, 2010. doi:10.1007/978-3-642-15775-2_21.

24 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

25 Harry B. Hunt III, Madhav V. Marathe, Venkatesh Radhakrishnan, S. S. Ravi, Daniel J. Rosenkrantz, and Richard Edwin Stearns. A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs. In *Proceedings of the 2nd Annual European Symposium on Algorithms, (ESA 1994)*, volume 855 of *LNCS*, pages 424–435, 1994. doi:10.1007/BFB0049428.

26 Donggu Kang and James Payor. Flow rounding. *arXiv*, 2015. arXiv:1507.08139.

27 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

28 J. Mark Keil and Debajyoti Mondal. The maximum clique problem in a disk graph made easy. In *Proceedings of the 41st International Symposium on Computational Geometry, (SoCG 2025)*, volume 332 of *LIPIcs*, pages 63:1–63:16, 2025. doi:10.4230/LIPIcs.SOCG.2025.63.

29 J. Mark Keil, Debajyoti Mondal, Ehsan Moradi, and Yakov Nekrich. Finding a maximum clique in a grounded 1-bend string graph. *J. Graph Algorithms Appl.*, 26(1):553–575, 2022. doi:10.7155/JGAA.00608.

30 Paul Koebe. Kontaktprobleme der konformen abbildung. *Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Kl.*, 88:141–164, 1936.

31 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM J. Comput.*, 51(3):723–765, 2022. doi:10.1137/20M1388371.

32 Tim Nieberg, Johann L. Hurink, and Walter Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4):49:1–49:17, 2008. doi:10.1145/1383369.1383380.

33 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

34 Lajos Stachó. A solution of Gallai's problem on pinning down circles. *Mat. Lapok*, 32(1-3):19–47, 1981–84.

35 Anastasiia Tkachenko and Haitao Wang. Computing maximum cliques in unit disk graphs. In *Proceedings of the 37th Canadian Conference on Computational Geometry, (CCCG 2025)*, pages 283–291, 2025.

36 Haitao Wang. Unit-disk range searching and applications. *J. Comput. Geom.*, 14(1):343–394, 2023. doi:10.20382/JOCG.V14I1A13.

Dynamic MIS Revisited: Incremental, Fault Tolerant and Fully Dynamic

Manoj Gupta
Department of Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India

Shahbaz Khan[1]
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Madhu Surendra
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Abstract

Given a dynamic graph G, we aim to maintain a maximal independent set (MIS). This problem admits a deterministic solution in $O(m^{2/3})$ time [SOSA21], while randomized algorithms for oblivious adversary take $O(\text{poly}\log n)$ [FOCS19] time. Recently, Bernstein et al. [SODA26] proved a conditional lower bound of $\Omega(n^{1-o(1)})$ amortized update time even for incremental MIS with adaptive adversary. In this paper, we establish similar deterministic bounds through the lens of incremental and fault tolerant algorithms for MIS, and show the following:

1. **Incremental:** MIS can be maintained under edge insertions in $O(\sqrt{m})$ amortized update time.
2. **Fault Tolerant:** Using $O(m)$ preprocessing time, the MIS can be reported after k edge deletions in $O(k+\overline{n}^2)$ time, where $\overline{n}$ is the number of vertices on which deleted edges are incident.
3. **Fully Dynamic:** MIS can be maintained under fully dynamic edge updates (insertions and deletions) in $O(m^{2/3})$ amortized update time.

Our incremental result is thus polynomially optimal even on allowing randomization with an adaptive adversary. Our fully dynamic result matches the state-of-the-art [SOSA21], and uses our incremental and fault tolerant algorithms as a black box. Although our fully dynamic algorithm does not improve the existing bounds, it provides additional insights into the harder edge-insertion case. We believe these insights may potentially be useful for improving the bounds for fully dynamic MIS in the future to match the incremental MIS bounds.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Maximal Independent Set, MIS, Incremental, Fault Tolerant, Fully dynamic

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.21

Related Version The incremental MIS algorithm was previously reported [9] but not published [10]. *Previous Version*: https://arxiv.org/abs/1804.01823 [9]

Acknowledgements We would like to thank the anonymous reviewers for pointing out the matching lower bound [SODA26] for incremental MIS with adaptive adversaries, strengthening our result.

1 Introduction

Given a graph $G = (V, E)$ with n vertices and m edges, a set $M \subseteq V$ forms an *independent set* if no two vertices in M share an edge, i.e., $\forall x, y \in M, (x, y) \notin E$. A *maximal* independent set is one that cannot be extended by including any other vertex while preserving independence. Although computing a maximum cardinality independent set is NP-hard [8], a simple greedy algorithm computes a maximal independent set (MIS) in $O(m)$ time.

[1] Corresponding author

© Manoj Gupta, Shahbaz Khan, and Madhu Surendra;
licensed under Creative Commons License CC-BY 4.0

A graph is *incremental* if it gains edges over time, *decremental* if it loses edges, and *fully dynamic* if both insertions and deletions occur. In dynamic settings, one can trivially maintain a maximal independent set (MIS) in $O(m)$ update time by recomputing it from scratch after each update. Recent work has focused on improving this update time. Censor-Hillel et al. [5] presented the first non-trivial algorithm requiring $O(\Delta) = O(n)$ amortized update time, where Δ is the maximum degree of a vertex in the graph. Assadi, Onak, Schieber, and Solomon [1] improved it to an $O(m^{3/4})$ time algorithm to maintain an MIS. Du and Zhang [7] improved this update time to $\tilde{O}(m^{2/3})$[2] amortized, and independently Gupta and Khan [10] acheived $O(m^{2/3})$, all using deterministic algorithms. Du and Zhang [7] also proposed a randomized algorithm with $\tilde{O}(\sqrt{m})$ update time, which was later improved to $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m^{1/3})$ by Assadi et al. [2]. Finally, Chechik and Zhang [6] and Behnezhad et al. [3] independently achieved an $\tilde{O}(1)$ update time using randomized methods. All known algorithms with $o(m^c)$ update time are randomized assuming an oblivious adversary. Recently, Bernstein et al. [4] showed that even for randomized algorithms with an adaptive adversary, assuming the combinatorial BMM conjecture, there exists a lower bound of $O(n^{1-o(1)})$ amortized update time for incremental MIS, which naturally extend to deterministic and fully dynamic variants.

In this paper, we focus on deterministic algorithms for dynamically maintaining MIS. Despite not improving the state-of-the-art fully dynamic algorithm, our aim is to present further insights into the problem by using an alternate approach, potentially paving a way for improving the theoretical bounds in the future. Our results are as follows:

As a first step, we present a deterministic algorithm for maintaining an MIS in an *incremental* graph, where only edge insertions occur. Our algorithm achieves an amortized $O(\sqrt{m})$ update time, improving over the current state-of-the-art [10].

▶ **Theorem 1** (Incremental MIS). *Given a graph G with n vertices, its MIS can be maintained under the insertion of m edges in amortized $O(\sqrt{m})$ update time.*

▶ Remark 2. We also show that the upper bound of the incremental algorithm is tight in Appendix A. In the light of recent hardness result [4], our algorithm is polynomially optimal.

Next, we design a fault-tolerant algorithm for maintaining an MIS. In this setting, the graph $G(V, E)$ remains static, and we preprocess it to construct a data structure that can efficiently answer the following query:

FaultTolerant(F): Compute an MIS of $G(V, E \setminus F)$, where F is a set of edge failures.

We present a deterministic fault-tolerant algorithm with the following guarantee:

▶ **Theorem 3** (Fault Tolerant MIS). *Given a graph $G(V, E)$ with n vertices, it can be preprocessed in $O(m)$ time such that for any set $F \subseteq E$ of k edges, the MIS of $G(V, E \setminus F)$ can be reported in $O(k + \overline{n}^2)$ time, where $\overline{n}$ is the number of vertices incident to edges in F.*

▶ Remark 4. Since $\overline{n} = O(k)$, a simpler bound for the algorithm may be $O(k^2)$. However, we report the bound using $\overline{n}$ because k can be $O(m) = O(n^2)$, but the time is bounded by $O(n^2)$ not $O(k^2) = O(n^4)$. We further restrict it to $O(\overline{n}^2)$ so that it can be directly used in our result as a black box, simplifying its analysis.

Finally, we combine the incremental and fault-tolerant algorithms to get the following:

▶ **Theorem 5** (Fully Dynamic MIS). *Given a graph $G(V, E)$ with n vertices, its MIS can be maintained under fully dynamic edge updates in amortized $O(m^{2/3})$ update time, where m is the current number of edges in G.*

[2] $\tilde{O}$ hides polylogarithmic factors in n

Although our algorithm does not improve the state-of-the-art theoretical bound, the new approach may present new insights to potentially improve the state-of-the-art in the future.

2 Preliminaries

Given a graph $G(V, E)$ having n vertices and m edges, which is connected ($m \geqslant n-1$). We represent the graph G formed by deleting a set of edges $F \subseteq E$ from G as $G(V, E \setminus F)$. Also, we represent a subgraph of G induced by the vertices in V' as $G(V')$ which refers to $G(V', E')$ where $E' = E \cap \{V' \times V'\}$. We represent the degree of a vertex v by $\deg(v)$, i.e., the number of neighbours of a vertex v in G. We also represent the maximum degree of a vertex in G as Δ. A set $M \subseteq V$ is *independent* if no two vertices in M share an edge, i.e., $\forall x, y \in M, (x, y) \notin E$. A *maximal* independent $\mathcal{M}$ set is an independent set that cannot be extended by including any other vertex while preserving independence.

Fully dynamic algorithm with amortized $O(\Delta)$ update time

We now review the basic $O(\Delta)$ [5] algorithm for maintaining a maximal independent set $\mathcal{M}$. Some of its procedures and results would be used as a black box in our algorithm.

The algorithm maintains a $\mathtt{count}[u]$ for each vertex u, which contains the number of neighbours of u in $\mathcal{M}$. Assuming we start with the empty graph, we set $\mathtt{count}[u] = 0$ for every $u \in G$ and all the vertices are in $\mathcal{M}$. The algorithm maintains the following invariant

$$v \in \text{MIS}, \text{ iff } \mathtt{count}[v] = 0 \tag{1}$$

On insertion of an edge (u, v), InsertΔ (see Algorithm 1) checks if at least one of u or v is not in $\mathcal{M}$, in which case it simply updates its $\mathtt{count}$ if required. Else, if both are in $\mathcal{M}$, it removes one (say u) from $\mathcal{M}$, reducing the $\mathtt{count}$ of the neighbours of u. In case the count of any neighbour w becomes $\mathtt{zero}$, it is added to $\mathcal{M}$, and the $\mathtt{count}$ of the neighbours of w is increased. On deletion of an edge (u, v), DeleteΔ simply updates the $\mathtt{count}$ of u or v if required. In case the $\mathtt{count}$ becomes $\mathtt{zero}$, the corresponding vertex is added to $\mathcal{M}$, increasing the $\mathtt{count}$ of its neighbours.

By construction, whenever an edge is inserted or deleted, or a vertex enters or leaves $\mathcal{M}$ its count is updated, and it is added to the $\mathcal{M}$ according to Invariant 1. Thus, the correctness of the algorithm follows from the invariant.

We now analyze the running time of the algorithm. The analysis hinges on the fact that both in InsertΔ(u, v) and DeleteΔ(u, v), at most one vertex is removed from $\mathcal{M}$ while multiple vertices can be potentially added to $\mathcal{M}$.

Let $\phi_i(u) = \deg(u)$ if $u \notin \mathcal{M}$, else 0 after the i-th update step of the algorithm. We define $\phi_i = \sum_u \phi_i(u)$. After initialization, $\phi_o(u) = 0$ for each $u \in V$ as in an empty graph, all the vertices are in $\mathcal{M}$. Thus, $\phi_o = 0$. Now, if DeleteΔ(u, v) is called at the i-th update step. In the first step of this procedure, (u, v) is removed from the graph. If $u \notin \mathcal{M}$ (or $v \notin \mathcal{M}$), this decreases the potential of u (or v) by 1. In any case, the amortized cost of the first step is $O(1)$. Let us look at the main part of the algorithm when w is added to $\mathcal{M}$. In this case, w informs all its neighbors about its inclusion in $\mathcal{M}$, incurring a cost of $\deg(w)$. But its potential also decreases from $\deg(w)$ to 0. Thus, the amortized cost of processing w is 0. This implies that the amortized running time of DeleteΔ(u, v) is $O(1)$.

Now, if InsertΔ(u, v) is called at the i-th update step when an edge (u, v) is added to G. Addition of (u, v) in the graph, may increase the potential of u (or v) if $u \notin \mathcal{M}$ (or if $v \notin \mathcal{M}$) by 1. Thus, the amortized running time of the last step of InsertΔ(u, v) is $O(1)$. The main

Algorithm 1 Fully dynamic algorithm for MIS.

```
1  InsertΔ(u, v):
2  if u ∈ M or v ∈ M then
3      if u ∉ M then Exchange u and v;
4      if v ∉ M then
5          count[v] ← count[v] + 1
6      else
7          Remove u from M;
8          foreach neighbor w of u do
9              count[w] ← count[w] − 1;
10             if count[w] = 0 then
11                 Add w to M
12                 foreach neighbor x of w do
13                     count[x] ← count[x] + 1
14 Add (u, v) to the graph G;
15 DeleteΔ(u, v):
16 Remove (u, v) to the graph G;
17 if u ∈ M then
18     count[v] ← count[v] − 1
19 if v ∈ M then
20     count[u] ← count[u] − 1
21 foreach w ∈ {u, v} do
22     if count[w] = 0 then
23         Add w to M
24         foreach neighbor x of w do
25             count[x] ← count[x] + 1
```

part of the algorithm is when u is removed from M and many of its neighbours enter M. Let w be a neighbor of u that enters M. As discussed in the procedure DeleteΔ, the amortized cost of this operation is 0. Now, when u is removed from M, the potential of u increases to $\deg(u)$. Also, u incurs $\deg(u)$ work to update the count of its neighbours. Thus, the amortized time to process u is $O(\deg(u))$. Thus, the amortized time of InsertΔ(u, v) is $O(\deg(u))$ if u is removed from M and is $O(1)$ otherwise, proving the following result.

▶ **Lemma 6.** *For a given graph $G(V, E)$, having m edges and n vertices, the amortized update time of an algorithm is $O(\deg(u))$ if vertex u leaves the MIS M, and $O(1)$ otherwise.*

Since in both InsertΔ and DeleteΔ, at most one vertex (say u) leaves the MIS M in each update, the amortized update time is bounded by $O(\deg(u)) \leqslant O(\Delta)$. This proves the amortized $O(\Delta)$ update time of the algorithm.

3 Incremental MIS

We now consider the problem of maintaining an MIS in an incremental graph. We refer to a vertex v as *light* if $\deg(v) \leqslant \Delta_l$ and *heavy* otherwise, where Δ_l shall be computed later. Since the graph is incremental, all vertices start as light, and once a vertex becomes heavy, it remains heavy. Note that this distinction is required only for the analysis and not the algorithm. Similar to the $O(\Delta)$ algorithm, each vertex v maintains the count of its neighbours in the M as $\mathsf{count}[v]$.

Now, our algorithm matches the insert procedure InsertΔ exactly except for one crucial modification (see Algorithm 2). On insertion of an edge (u, v), if both the vertices are in M, instead of *arbitrarily* removing a vertex u from M, we perform a *degree biased removal*, i.e., we remove the vertex u having lower degree $\deg(u)$. Since the remaining procedure exactly matches InsertΔ, it correctness also follows from the correctness of $O(\Delta)$ algorithm.

Algorithm 2 INCREMENTAL(u, v).

```
1  if u ∈ M or v ∈ M then
2      if u ∉ M then Exchange u and v;
3      if v ∉ M then count[v] ← count[v] + 1 ;
4      else
5          if deg(u) > deg(v) then Exchange u and v;
6          Remove u from M;
7          foreach neighbor w of u do
8              count[w] ← count[w] − 1;
9              if count[w] = 0 then
10                 Add w to M
11                 foreach neighbor x of w do
12                     count[x] ← count[x] + 1
13 Add (u, v) to the graph G;
```

We now analyze the running time of our algorithm. Lemma 6 shows that the amortized update time of an update is $O(\deg(u))$ if a vertex u leaves M. Moreover, in each update, at most one vertex can leave M. Let the i^{th} edge update inserts the edge (u_i, v_i), such that u_i leave M (if any). We analyze the total time taken by a vertex over two *phases*: when it is *light* ($\deg(u_i) \leqslant \Delta_l$) and when it becomes *heavy* ($\deg(u_i) > \Delta_l$). If u_i is light, then we remove it from M in $O(\Delta_l)$ amortized time, since $\deg(u_i) \leqslant \Delta_l$. If u_i is heavy, then v_i must also be heavy since u_i was removed from M only because $\deg(v_i) \geqslant \deg(u_i) \geqslant \Delta_l$. Let the number of times a heavy vertex leaves M be n_h. Since a heavy u_i can leave M only when v_i is heavy, $n_h \leqslant O(\frac{m}{\Delta_l})$ as there are $O(\frac{m}{\Delta_l})$ heavy vertices in the final graph. Therefore, u_i can be removed from M at most $O(n_h) = O(\frac{m}{\Delta_l})$ times due to its heavy neighbors.

Combining both phases, we now compute the total amortized update time for m insertions.

$$\begin{aligned}
\sum_{i=1}^{m} O(\deg(u_i)) &= \sum_{i:u_i \text{ is light}} O(\deg(u_i)) + \sum_{i:u_i \text{is heavy}} O(\deg(u_i)) \\
&\leqslant \sum_{i=1}^{m} O(\Delta_l) + \textstyle\sum_{u \in V} n_h \times O(\deg(u)) \\
&\leqslant O(m\Delta_l) + O(mn_h)
\end{aligned}$$

Hence, the total amortized update time of our incremental algorithm is $O(m(\Delta_l + n_h))$, giving amortized $O(\Delta_l + n_h)$ update time, resulting in the following lemma:

▶ **Lemma 7.** *Maintaining the MIS in the incremental setting requires amortized* $O(\Delta_l + n_h)$ *update time, where* Δ_l *is the maximum degree of a light vertex, and* n_h *is the number of times a heavy vertex leaves* M*.*

Since $n_h = O(\frac{m}{\Delta_l})$, substituting $\Delta_l = \sqrt{m}$ gives us the following result.

▶ **Theorem 1** (Incremental MIS)**.** *Given a graph* G *with* n *vertices, its MIS can be maintained under the insertion of* m *edges in amortized* $O(\sqrt{m})$ *update time.*

▶ Remark 8. We show that the upper bounds for the $O(\Delta)$ algorithm and our incremental algorithm are tight in Appendix A.

4 Fault Tolerant MIS

We now describe our fault tolerant algorithm. It follows from the simple delete procedure *Delete*Δ of the $O(\Delta)$ algorithm with a careful analysis. Given that static graph $G(V, E)$, we preprocess it to build a data structure to answer the following query.

FaultTolerant(F): Compute an MIS of $G(V, E \setminus F)$, where F is a set of edge failures.

Our fault tolerant algorithm preprocesses the graph to compute the maximal independent set $\mathcal{M}$ of G and $\texttt{count}[v]$ for every $v \in V$. These can be computed in $O(m)$ time using the static algorithm. The pseudocode for the query algorithm is shown in Algorithm 3.

Let the end vertices of the edges in F be V_F. The deletion of edges in F can potentially reduce the $\texttt{count}[v]$ for a vertex $v \in V_F$ if the other end vertex is in $\mathcal{M}$. In case this count becomes zero, the vertex may be potentially added to $\mathcal{M}$. Let the set of such vertices be V_d. Since the vertices in V_d may not be independent, we cannot simply add all the vertices in V_d to $\mathcal{M}$. Instead, we add the MIS $\mathcal{M}$' of the subgraph $G(V_d)$ to $\mathcal{M}$. However, since we are dealing with the fault tolerant model, we do not modify the $\texttt{count}[\cdot]$ and $\mathcal{M}$ directly. We instead make a copy $c[v]$ of $\texttt{count}[v]$, which is modified, so that the changes need not be reversed after the query. Thus, our data structure can be directly used for the next query F'.

Algorithm 3 FaultTolerant(F): Return vertices added to $\mathcal{M}$ after deleting the edges F.

```
1 V_F ← Vertices incident on F;
2 foreach v ∈ V_F do c[v] ← count[v];
3 foreach (u, v) ∈ F do
4     if u ∈ M then c[v] ← c[v] − 1;
5     if v ∈M then c[u] ← c[u] − 1;
6 foreach v ∈ V_F do
7     if c[v] = 0 then Add v to V_d;
8 M'← MIS of G(V_d);
9 Return M ∪M'
```

Let $\overline{n}$ be the number of vertices on which the edges in F are incident, hence $|V_F| = \overline{n}$. Now, V_F can be computed in $O(k)$ time, $c[v]$ can be initialized in $O(\overline{n})$ time and updated in $O(k)$ time while processing each edge in F. Thereafter, V_d can be computed in $O(\overline{n})$ time where $V_d \subseteq V_F$ hence $|V_d| = O(\overline{n})$. The MIS on $G(V_d)$ can computed in $O(|V_d|^2) = O(\overline{n}^2)$ time. We thus get the following result for fault tolerant MIS.

▶ **Theorem 3** (Fault Tolerant MIS). *Given a graph $G(V, E)$ with n vertices, it can be preprocessed in $O(m)$ time such that for any set $F \subseteq E$ of k edges, the MIS of $G(V, E \setminus F)$ can be reported in $O(k + \overline{n}^2)$ time, where $\overline{n}$ is the number of vertices incident to edges in F.*

5 Fully dynamic MIS without rebuilding

We now describe our fully dynamic algorithm for MIS requiring $O(\min\{\Delta, m^{2/3}\})$ time which matches the state-of-the-art algorithm [10]. While the previous algorithm was based on $O(\Delta)$ algorithm [1] and the static algorithm, our algorithm is based on the incremental and fault tolerant algorithms described earlier.

At its core, our algorithm classifies a vertex v as *light* if $\deg(v) \leqslant m^{2/3}$, and *heavy* otherwise (similar to [10]). All insertions are performed using the incremental algorithm, while the deletions are performed based on the incident vertices. In case at least one of the incident vertices is *light*, we use the delete procedure DeleteΔ from the basic $O(\Delta)$ algorithm. However, if both incident vertices are heavy, we collect the deleted edges in F and use our fault tolerant algorithm to report the solution after every update. This essentially

handles the harder case of edge insertion between two heavy vertices, as handling their deletion limits such insertions similar to the incremental algorithm. If such an edge (deleted and stored in F) is inserted back into the graph, it is simply removed from F, and the fault tolerant algorithm updates $\mathcal{M}$ accordingly (see pseudocode in Algorithm 4).

Algorithm 4 Fully dynamic MIS using incremental and fault tolerant MIS algorithms.

```
1  Insert(u, v):
2  if (u, v) ∈ F then
3  |  Delete (u, v) from F;
4  else Incremental(u, v) ;
5  M" ← FaultTolerant(F);
6  Report M ∪M";
7  Delete(u, v):
8  if both u and v are heavy then
9  |  Add (u, v) in F;
10 else DeleteΔ(u, v) ;
11 M" ← FaultTolerant(F);
12 Report M ∪M";
```

The main intuition behind the time complexity is to limit the number of times a heavy vertex v leaves the MIS in the incremental algorithm. This is bounded by the number of edges inserted between v and a heavy vertex. In the fully dynamic setting, we avoid repeated insertions of such edges using the fault tolerant algorithm to process their deletions temporarily without affecting the main algorithm. Thus, given $\Delta_l = m^{2/3}$ and $n_h = m^{1/3}$ (as an edge to each of $m^{1/3}$ heavy vertex can be inserted exactly once), the edge insertions can be processed in amortized $O(m^{2/3})$ time (Lemma 7). Now, the edges collected in F are incident on both *heavy* vertices, resulting in $k = m^{2/3}$ and $\overline{n} = m^{1/3}$, allowing the fault tolerant algorithm to be processed in $O(m^{2/3})$ time (Theorem 3). Thus, the edge deletions can be processed in $O(m^{2/3})$ amortized time requiring $O(1)$ amortized if at least one incident vertex is light (Lemma 6, as no vertex leaves $\mathcal{M}$) and $O(m^{2/3})$ for the fault tolerant algorithm. This results in the following fully dynamic algorithm.

▶ **Theorem 5** (Fully Dynamic MIS). *Given a graph $G(V, E)$ with n vertices, its MIS can be maintained under fully dynamic edge updates in amortized $O(m^{2/3})$ update time, where m is the current number of edges in G.*

▶ Remark 9. The algorithm distinguishes the vertices as heavy or light based on a degree threshold $c = m^{2/3}$. However, the value of m keeps changing in a fully dynamic graph. We thus initialize the threshold $c_0 = m_0^{2/3}$, where m_0 is the initial number of edges. We keep the same threshold while $\frac{m_0}{2} \leqslant m \leqslant 2m_0$. In case m passes this range, we rebuild the MIS with $F = \emptyset$ and $m_0 = m$. The total cost of the algorithm $O(m \times m^{2/3})$ can be amortized among the $O(m_0)$ updates after which m exceeded the bounds, requiring amortized $O(m^{2/3})$ time.

▶ Remark 10. The fully dynamic setting can also potentially change the status of a vertex between light and heavy. In case a vertex v converts from light to heavy, no update is required as newly deleted edges on v will be added to F. However, in case a vertex converts from heavy to light, all its edges in F need to be deleted from the graph G, requiring potentially $O(m^{2/3})$ amortized time per deletion. We thus relax the criteria, converting a light vertex v to heavy if $\deg(v) > m^{2/3}$ but converting it to light when $\deg(v) < \frac{m^{2/3}}{2}$. This allows to charge $O(m^{2/3})$ for the deletion of its $O(m^{1/3})$ edges to heavy vertices during the $O(m^{2/3})$ updates when $\frac{m^{2/3}}{2} < \deg(v) < m^{2/3}$ resulting in amortized $O(m^{1/3})$ time per update.

▶ Remark 11. Though our proposed algorithm divides the vertices into light and heavy, similar to the state-of-the-art algorithm [10], its handling of heavy updates avoids rebuilding the MIS for heavy vertices using a simpler fault tolerant algorithm. Thus, our algorithm is expected to perform better in practice for edge updates between heavy vertices.

6 Conclusion

The dynamic MIS problem is one of the unique problems in which handling edge insertions is apparently harder than edge deletions. For most classical graph problems, such as connectivity, reachability, shortest paths, etc., handling edge deletions is harder than edge insertions. Hence, given a faster algorithm for incremental MIS requiring $O(\sqrt{m})$ amortized update time, we are hopeful to achieve the same bound for the fully dynamic case. Further, given the lower bound of $O(n^{1-o(1)})$ for incremental MIS, a polynomial improvement is not possible even for a randomized algorithm with an adaptive adversary.

In this paper, we highlighted an alternate way to handle insertions faster. We also presented a fully dynamic algorithm utilizing insights from the incremental and fault tolerant algorithm, although without improving the state-of-the-art. We believe that new insights may be drawn from our work, potentially resulting in a faster fully dynamic algorithm.

References

1 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 815–826, 2018. doi:10.1145/3188745.3188922.

2 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.

3 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00032.

4 Aaron Bernstein, Sayan Bhattacharya, Nick Fischer, Peter Kiss, and Thatchaphol Saranurak. Separations between oblivious and adaptive adversaries for natural dynamic graph problems. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 5669–5687. SIAM, 2026.

5 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 217–226, 2016. doi:10.1145/2933057.2933083.

6 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00031.

7 Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *arXiv preprint arXiv:1804.08908*, 2018. arXiv:1804.08908.

8 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

9 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.

10 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set, maximum flow and maximum matching. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 86–91. SIAM, 2021. doi:10.1137/1.9781611976496.10.

A Tightness of the Incremental Algorithm

We now present worst-case examples demonstrating the tightness of the analysis of $O(\Delta)$ algorithm and our incremental MIS algorithm. These essentially highlight the difference between arbitrary removal and degree biased removal of an end vertex on insertion of an edge between two vertices of the MIS.

A.1 Arbitrary removal

We start with an empty graph where all the vertices are in MIS. Let the vertices be divided into two sets $\mathcal{A} = \{a_1, ..., a_k\}$ and $\mathcal{B} = b_1, ..., b_t$ (refer to Figure 1), where $\mathcal{A}$ has $k = m/\Delta$ vertices and $\mathcal{B}$ has the remaining $t = n - m/\Delta = O(n)$ vertices.

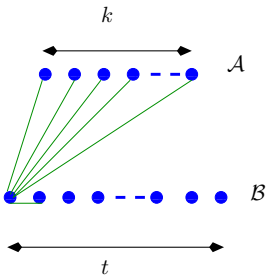


Figure 1 Worst case example for arbitrary removal.

We shall divide the insertion of edges into t phases, where in the j^{th} phase we add an edge from each vertex of $\mathcal{A}$ to b_j. And we always chose the vertex of $\mathcal{A}$ to be removed from MIS $\mathcal{M}$. At the end of the phase, all vertices of $\mathcal{A}$ are out of $\mathcal{M}$ and are connected to $b_j \in \mathcal{M}$. The phase ends with the addition of an edge between b_j and b_{j+1}, which removes b_j from $\mathcal{M}$ and hence all vertices of $\mathcal{A}$ are moved back to $\mathcal{M}$. Since each vertex is allowed only Δ neighbours, and each phase adds a neighbour to each vertex in $\mathcal{A}$, we stop after $t^* = \Delta$ phases.

Hence, after t^* phases, we have added all the edges between $\mathcal{A}$ and the first t^* vertices in $\mathcal{B}$, and among the first t^* adjacent vertices of $\mathcal{B}$. Overall we add $O(|\mathcal{A}| \times t^* + t^*)$ edges, which equals to $O(kt^*) = O(\frac{m}{\Delta} * \Delta) = O(m)$ edges.

Using Lemma 6, the total edges processed during the j^{th} phase is the sum of the degrees of vertices that were removed from $\mathcal{M}$, i.e., all the vertices in $\mathcal{A}$ and b_j. Now, in the j^{th} phase the degree of each vertex in $\mathcal{A}$ is $j - 1$, being connected to $b_1, ..., b_{j-1}$ and the degree of b_j is k. Hence, the total work in j^{th} phase is $|\mathcal{A}| \times (j - 1) + k = k \times j$. Thus, the total work done over all phases is

$$\sum_{j=1}^{t^*} \Omega(k \times j) = \Omega(k \times t^{*2}) = \Omega(\frac{m}{\Delta} \times \Delta^2) = \Omega(m\Delta)$$

Thus, we have the following bound for $O(\Delta)$ algorithm in incremental (and hence fully dynamic) setting.

▶ **Theorem 12.** *For each value of $n \leqslant m \leqslant \binom{n}{2}$ and $1 \leqslant \Delta \leqslant n$, there exists a sequence of m edge insertions where the degree of each vertex is bounded by Δ for which $O(\Delta)$ algorithm requires total $\Theta(m\Delta)$ time to maintain the MIS.*

A.2 Degree biased removal

In this example, we consider our incremental MIS algorithm, which chooses the end vertex with the lower degree to be removed from $\mathcal{M}$ when an edge is inserted between two vertices in $\mathcal{M}$. We essentially modify the previous example to make sure the vertices of $\mathcal{A}$ necessarily fall when connected to the vertices in $\mathcal{B}$.

Let the vertices to be divided into two sets $\mathcal{A} = \{a_1, ..., a_k\}$ and $\mathcal{B} = \{b_1, ..., b_t\}$ as before, and an additional set $\mathcal{C}$ of residual vertices to ensure that degrees of vertices in $\mathcal{B}$ are sufficiently high (refer to Figure 2), where $k = t = \sqrt{m}/4$. We connect each vertex b_i with some $\sqrt{m}+1$ vertices in $\mathcal{C}$. Additionally, we have a vertex b_0, connected to all the $O(n)$ vertices in $\mathcal{C}$. We initialize the MIS with all vertices of $\mathcal{A}$, $\mathcal{B}$, and b_0 in $\mathcal{M}$.

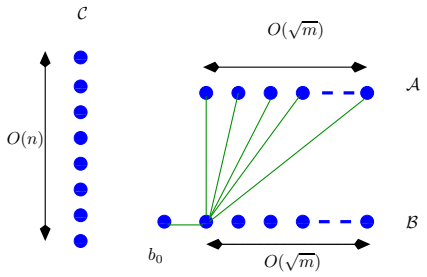


Figure 2 Tightness Example for degree biased removal.

Again, we divide the insertion of edges into t phases, where in the j^{th} phase we add an edge from each vertex of $\mathcal{A}$ to b_j. Since the maximum degree of a vertex in $\mathcal{A}$ is $|\mathcal{B}|$, and the degree of each b_j is $\sqrt{m}+1$, we always have the vertex of $\mathcal{A}$ to be removed from MIS $\mathcal{M}$. At the end of the phase, all vertices of $\mathcal{A}$ are out of $\mathcal{M}$ and are connected to $b_j \in \mathcal{M}$. The phase ends with the addition of (b_j, b_0), which removes b_j from $\mathcal{M}$ and hence all vertices of $\mathcal{A}$ are moved back to $\mathcal{M}$.

Hence, after t phases, we have added all the edges between $\mathcal{A}$ and $\mathcal{B}$, between each vertex of $\mathcal{B}$ and b_0. The initial graph already had each vertex of $\mathcal{B}$ connected to some $\sqrt{m}+1$ neighbours in $\mathcal{C}$ and b_0 connected to all neighbours of $\mathcal{C}$. Overall we add $O(|\mathcal{A}| \times |\mathcal{B}| + |\mathcal{B}| \times (\sqrt{m}+1) + |\mathcal{B}| + n)$ edges, which equals to $O(kt + t\sqrt{m} + n) = O(m+n)$ edges.

Again, using Lemma 6, the total edges processed during the j^{th} phase is the sum of the degrees of vertices that were removed from $\mathcal{M}$, i.e., all the vertices in $\mathcal{A}$ and b_j. Now, in the j^{th} phase the degree of each vertex in $\mathcal{A}$ is $j-1$, being connected to $b_1, ..., b_{j-1}$ and the degree of b_j is $\Omega(\sqrt{m})$. Hence, the total work done in j^{th} phase is $\Omega(|\mathcal{A}| \times j + \sqrt{m}) = \Omega(k \times j + \sqrt{m})$. Thus, the total work done over all phases is

$$\sum_{j=1}^{t} \Omega(k \times (j-1) + \sqrt{m}) = \Omega(k \times t^2 + k\sqrt{m}) = \Omega(\sqrt{m} \times \sqrt{m}^2 + m) = \Omega(m\sqrt{m})$$

Thus, we have the following bound for our incremental MIS algorithm.

▶ **Theorem 13.** *For each value of $n \leqslant m \leqslant \binom{n}{2}$, there exists a sequence of m edge insertions for which our incremental algorithm requires total $\Theta(m\sqrt{m})$ time to maintain the MIS.*

▶ Remark. This example is similar to the previous example with the exception that in this case, the degree of vertices in $\mathcal{B}$ should remain higher than the degree of a vertex in $\mathcal{A}$, i.e., $|\mathcal{B}|$ at the end of all the phases. Hence, $|\mathcal{B}| \leqslant \sqrt{m}$ else the total edges in G would be $\Omega(m)$.

Thus, the number of neighbours of $\mathcal{A}$ in $\mathcal{B}$ cannot be increased despite choosing any large value of Δ. As evident from the previous example in the absence of such a restriction, the amortized time can be raised to Δ, implying the significance of the degree biased removal in the incremental algorithm.

Search-Space Reduction for Boolean MinCSPs via Essential Constraints

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands

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

Abstract

For a fixed set $\mathcal{F}$ of Boolean constraint types, a MinCSP($\mathcal{F}$)-instance consists of a formula F that applies m constraints from $\mathcal{F}$ to a set of n Boolean variables. The goal is to remove a minimum subset of constraint applications from F to make the remaining formula satisfiable. Previous work characterized how the choice of $\mathcal{F}$ affects its polynomial-time solvability and approximability. We extend a recently introduced preprocessing framework for graph problems to the problem above. Rephrased in the context of CSPs, this framework defines a constraint application from a given formula F as *c-essential* if it is contained in all c-approximate solutions to F. Being able to efficiently detect these essential parts of a solution reduces the search space of any follow-up FPT algorithms parameterized by the solution size and yields an immediate asymptotic improvement to the runtime of such algorithms. In this work, we present a dichotomy theorem that distinguishes constraint sets $\mathcal{F}$ for which $c_{\mathcal{F}}$-essential constraint applications can be detected efficiently for some $c_{\mathcal{F}} \in \mathcal{O}(1)$, from those for which this task is intractable under established complexity-theoretic conjectures. Our results show that for any set $\mathcal{F}$ of *bijunctive* constraints, there is a polynomial-time algorithm that detects $\mathcal{O}(1)$-essential constraint applications. This contrasts the fact that constant-factor approximating a bijunctive MinCSP($\mathcal{F}$)-problem is intractable under the Unique Games Conjecture.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Constraint and logic programming

Keywords and phrases fixed-parameter tractability, constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.22

Related Version *Full Version*: https://arxiv.org/abs/2606.00770

1 Introduction

Constraint Satisfaction Problems (CSPs) form a broad class of problems that describe many well-known computational tasks and allow us to study them in a unified language [19]. A *constraint* of arity r over a domain D is a function $f\colon D^r \to \{0,1\}$, and a *constraint set* $\mathcal{F}$ is a set of such constraints over the same domain and possibly of different arity. In this work, we focus on Boolean CSPs, and assume the domain D to be $\{0,1\}$ from now on. For a fixed constraint set $\mathcal{F}$, a CSP instance contains a formula consisting of multiple applications of constraints from $\mathcal{F}$ over a set of variables $\mathbf{X}$. Such a *constraint application* is a pair consisting of a constraint $f \in \mathcal{F}$ and a sequence $\mathbf{X}'$, whose length equals the arity of f, of variables from $\mathbf{X}$ that f is applied to. A formula consisting of several constraint applications is called an *$\mathcal{F}$-formula* if all constraint applications come from a constraint in $\mathcal{F}$.

Though many variants of CSPs exist, the goal in such problems is typically related to the satisfiability of a given formula F. An *assignment* $s : \mathbf{X} \to \{0,1\}$ over variable set $\mathbf{X}$ is a function that maps every variable in this set to a value in the domain (in our setting the Boolean domain $\{0,1\}$). Then, an assignment s over variable set $\mathbf{X}$ is said to *satisfy* a constraint application $(f, \mathbf{X}')$ over variable set $\mathbf{X}' \subseteq \mathbf{X}$ if $f(s(\mathbf{X}')) = 1$. We say that an assignment s satisfies a formula F if the assignment satisfies all constraint applications in F. Finally, a formula is *satisfiable* if there exists at least one assignment that satisfies it.

© Bart M. P. Jansen and Ruben F. A. Verhaegh;
licensed under Creative Commons License CC-BY 4.0

Perhaps the most well-known class of CSPs is the one where, for a given $\mathcal{F}$-formula F, the task is to determine whether or not F is satisfiable. Varying the pool of allowed constraints $\mathcal{F}$ impacts the difficulty of the problem and, in 1978, Schaefer gave a dichotomy theorem that characterizes for which constraint sets $\mathcal{F}$ the resulting problem is polynomial-time solvable and for which this is NP-hard [21]. Since then, many other variants of CSPs have been studied. We continue this line of research for the following class of optimization problems.

Weighted MinCSP($\mathcal{F}$)	
Input:	An $\mathcal{F}$-formula F in which each constraint application has a positive integer weight encoded in unary.
Task:	Find a minimum-weight set X of constraint applications in F s.t. $F - X$ is satisfiable.

For an instance F, we denote the weight of an optimal solution by $\mathrm{opt}(F)$. Additionally, we define the unweighted variant MinCSP($\mathcal{F}$) of the above problem as the restriction in which the weights in the input are all 1. Then, the resulting task is equivalent to finding a minimum-*size* set X for which $F - X$ is satisfiable.

This problem is NP-hard for many constraint classes $\mathcal{F}$. As such, e.g. approximation algorithms and fixed-parameter tractable (FPT) algorithms parameterized by the solution size t (i.e. algorithms that decide whether $\mathrm{opt}(F) \leq t$ in $g(t) \cdot \mathrm{poly}(n)$ time on formulas of size n, for some computable function g) have been studied. This has resulted in dichotomy theorems for the problem's constant-factor approximability in polynomial time [13], its solvability in FPT time [15], and its constant-factor approximability in FPT time [5].

We study a class of preprocessing algorithms for this problem. Various preprocessing algorithms are known to be very useful in practice as observed in SAT solvers for example [1], motivating a study of the extent to which polynomial-time preprocessing can aid in solving Weighted MinCSP($\mathcal{F}$). Preprocessing with the goal of reducing to an instance whose size is polynomially bounded in $\mathrm{opt}(F)$ has been studied for several problems through the notion of kernelization [7]. These studies include, for example, the Min 2CNF-Deletion problem (a.k.a. Almost 2-SAT [18]). Here, we take a different perspective on preprocessing with the goal of investigating how preprocessing can help for inputs whose solutions are large, that is, of size comparable to the total input size.

Essential constraint applications. We extend a recent framework by Bumpus et al. that introduces the notion of c-essential objects [6]. Although originally defined within the context of vertex selection problems on graphs, we can rephrase their original definition to apply to MinCSP($\mathcal{F}$) problems. Formally, for a given constant $c \in \mathbb{R}_{\geq 1}$ and an $\mathcal{F}$-formula F, we call a constraint application in this formula *c-essential* if it is contained in all c-approximate solutions to the (Weighted) MinCSP($\mathcal{F}$) instance F.

One could expect such c-essential objects to somehow stand out in the solution space, since we cannot even form a c-approximation without them. This might make it feasible to identify such objects during preprocessing, which would be useful for problems like MinCSP($\mathcal{F}$) in which the goal is to arrive at a certain structure by deleting a minimum number of objects. All essential objects are in particular part of all optimal solutions, so that, after finding and reducing them from the input, it suffices to solve an instance that looks for a strictly smaller solution. Since many FPT algorithms have a running time that scales exponentially with the solution size, each element of the solution we find during preprocessing decreases the running time by a multiplicative factor. As such, we are interested in preprocessing that finds essential objects, if there are any. This algorithmic task is formalized in the following problem statement, defined for a fixed constraint set $\mathcal{F}$ and constant $c \in \mathbb{R}_{\geq 1}$.

c-essential Detection for Weighted MinCSP($\mathcal{F}$)

Input: An $\mathcal{F}$-formula F with associated weights, and an integer k

Task: Find a subset S of constraint applications in F such that:

(G1) if $\text{opt}(F) \leq k$, then there is an optimal solution to F containing all of S, and

(G2) if $\text{opt}(F) = k$, then S contains all c-essential constraint applications.

Again we define the unweighted variant of the problem by requiring all weights in the input to be 1. Note that this problem is, in two ways, easier than the task of simply returning a set containing exactly the c-essential constraint applications in an input. First, the output set is allowed to contain more constraint applications, as long as they belong to an optimal solution. Secondly, the algorithm only needs to give a meaningful output if the integer k in its input provides a correct guess or upper bound on the optimal solution of the input. Nonetheless, being able to solve this detection task suffices to speed up FPT algorithms parameterized by solution size, in a black-box fashion. In particular, if for a constraint set $\mathcal{F}$ and a constant $c \in \mathbb{R}_{\geq 1}$, (1) there is a polynomial-time algorithm for c-essential Detection for MinCSP($\mathcal{F}$), and (2) there is an algorithm that outputs a solution to MinCSP($\mathcal{F}$) of size at most p, if one exists, in $g(p) \cdot \text{poly}(n)$ time for some computable function g, then there is an algorithm that computes an optimal solution of MinCSP($\mathcal{F}$) in $g(p-q) \cdot \text{poly}(n)$ time, where q is the number of c-essential constraint applications in the input and $p = \text{opt}(F)$. Bumpus et al. have proven this guarantee in the setting of graph problems, but their proof is easily seen to be valid for MinCSP($\mathcal{F}$) as well [6, Theorem 5.1].

Using this framework, several positive results and hardness results have been achieved for graph problems such as Vertex Multicut and Directed Feedback Vertex Set [6, 11]. Here, we extend the framework to achieve new results for CSPs.

Our results and other dichotomies. We present a dichotomy theorem that characterizes for which constraint sets $\mathcal{F}$ there is a constant c such that c-essential Detection for MinCSP($\mathcal{F}$) is polynomial-time solvable. Specifically, we show that this is true for constraint sets $\mathcal{F}$ that are 0-valid, 1-valid, IHS-B, or bijunctive (these terms are defined in Section 2). The dichotomy reveals that all other constraint sets $\mathcal{F}$ yield a MinCSP($\mathcal{F}$) problem for which c-essential detection is hard for every $c \in \mathbb{R}_{\geq 1}$, under established hardness assumptions.

▶ **Theorem 1.1.** *Let $\mathcal{F}$ be a finite Boolean constraint set.*

1. *If $\mathcal{F}$ is 0-valid, 1-valid, or 2-monotone, then there is a polynomial-time algorithm for c-essential Detection for MinCSP($\mathcal{F}$) for any $c \in \mathbb{R}_{\geq 1}$, since even Weighted MinCSP($\mathcal{F}$) is polynomial-time solvable in this case [13].*
2. *Otherwise, if $\mathcal{F}$ is IHS-B or bijunctive, then there is a constant $c_{\mathcal{F}}$ such that $c_{\mathcal{F}}$-essential Detection for Weighted MinCSP($\mathcal{F}$) is solvable in polynomial time.*
3. *Otherwise, if $\mathcal{F}$ is affine, then c-essential Detection for MinCSP($\mathcal{F}$) is NP-hard for every constant $c \in \mathbb{R}_{\geq 1}$ under the Unique Games Conjecture.*
4. *Otherwise, if $\mathcal{F}$ is weakly positive or weakly negative, then c-essential Detection for MinCSP($\mathcal{F}$) is NP-hard for every constant $c \in \mathbb{R}_{\geq 1}$.*
5. *Otherwise, c-essential Detection for MinCSP($\mathcal{F}$) is NP-hard for every constant $c \in \mathbb{R}_{\geq 1}$, even on formulas F with $\text{opt}(F) \leq 1$.*

As mentioned, other dichotomies exist that characterize the efficient solvability and approximability of MinCSP($\mathcal{F}$). Table 1 shows a comparison between some of these that closely relate to our work. It focuses on the boundary between positive and hardness results. The table includes, in left-to-right order, results on constant-factor approximating the problem

Table 1 A comparison of dichotomies for results on solving or approximating MinCSP($\mathcal{F}$). *Solving the problem on IHS-B or bijunctive constraint sets is FPT if an underlying graph (respectively the arrow graph or Gaifman graph) of the constraint set is $2K_2$-free. It is W[1]-hard otherwise [15].

Restriction on $\mathcal{F}$	Poly-time approx. [13]	FPT exact [15]	FPT approx. [5]	Essential detection
0-valid, 1-valid, 2-monotone	*easy*	*easy*	*easy*	*easy*
IHS-B	*easy*	*sometimes easy**	*easy*	*easy*
Bijunctive	hard	*sometimes easy**	*easy*	*easy*
None of the above	hard	hard	hard	hard

in polynomial time [13], exactly solving the problem in FPT time, parameterized by solution size [15], and constant-factor approximating the problem in FPT time [5]. A summary of our results from Theorem 1.1 is included in the final column. In a given row, the table indicates whether the specified tasks for constraint sets from that category – and not from any category from a previous row – are tractable under a suitable hardness assumption.

We point out that c-essential detection appears to be a strictly easier task than approximating the problem in polynomial-time or exactly solving the problem in FPT time. It is in particular interesting to see that all cases that admit an exact FPT algorithm also admit a polynomial-time c-essential detection algorithm. Based on previous work on c-essential objects mentioned earlier, this implies that all these FPT algorithms for MinCSP($\mathcal{F}$) can be updated to yield an improved runtime dependence on the complexity parameter. We also note that our results show the same boundary of hardness as the existing dichotomy on constant-factor approximating the problem in FPT time.

Finally, we mention some of the existing work on preprocessing, specifically kernelization and compression, for MinCSP and related problems. Unlike kernelization, compression allows problem instances to be transformed to an instance of a possibly different problem. Jansen and Włodarczyk have characterized the best-possible compression size achievable for MinCSP($\mathcal{F}$) in terms of the number of variables [12]. Related CSPs include determining whether a given $\mathcal{F}$-formula can be satisfied by an assignment that sets at least, or at most, a given number k of variables to 1. Kratsch and Wahlström characterized for which constraint sets the minimization problem admits a kernel of size polynomial in k [17]. Later, such a characterization was also given for the maximization problem [16].

Techniques. We proceed by giving a brief overview of the techniques we use to obtain our results. Our algorithmically most interesting positive result is the polynomial-time c-essential detection algorithm on bijunctive constraint sets. To achieve it, we use the insight that bijunctive formulas admit a representation in graph form. Then, for each constraint application in the input formula, we solve a sequence of separation problems on this graph to determine whether to put it in the output set of our c-essential detection algorithm or not.

Next, for each of our hardness results, we provide a two-step approach. To show, for a given category of constraint sets, that c-essential detection is hard for MinCSP($\mathcal{F}$), we start by formulating a canonical constraint set from that category and showing that the problem is hard for that set. Then, we show that the corresponding problem for all other constraint sets in that category is as hard as the problem for the canonical constraint set.

To achieve this second step, we show that an *essential implementation* of the canonical set $\mathcal{F}_{\text{can}}$ for a category can be made via any constraint set $\mathcal{F}$ in that category. That is, for every constraint application over $\mathcal{F}_{\text{can}}$, there is a conjunction of constraint applications over $\mathcal{F}$

that is equivalent to it, while satisfying an additional property as defined in Definition 4.1. The latter ensures that for every constant c, a reduction can be made from MinCSP($\mathcal{F}_{\text{can}}$) to Weighted MinCSP($\mathcal{F}$) that preserves the entire space of c-approximate solutions.

We remark that our definition of essential implementations is strictly stronger than the very similar notion of strong and perfect implementations by Khanna et al. [13]. An implementation from their framework also yields a reduction between CSP instances over different constraint sets. They showed that this reduction preserves the constant-factor approximability of the involved problems, but, in contrast to our result, it need not preserve the exact approximation ratio. We were surprised to see that most of the implementations we encountered in related work (also beyond strong and perfect implementations) happen to fit our definition of an essential implementation as well. This reveals even stronger links between problems that have already been shown to be reducible to one another.

Organization. The rest of the paper is organized as follows. In Section 2, we define some fundamental concepts and notation. Section 3 contains our positive results, proving Case 2 of Theorem 1.1. Section 4 contains our hardness results, starting with an introduction to essential implementations. Then, in Sections 4.1–4.3, we prove Cases 3, 4, and 5 of Theorem 1.1 respectively. We conclude with open questions in Section 5. The proofs of statements marked (★) are deferred to the full version.

2 Preliminaries

A constraint is a Boolean function $f\colon \{0,1\}^p \to \{0,1\}$ for some integer p, which is its *arity*. We assume all individual Boolean constraints to be satisfiable: unsatisfiable constraint applications in a formula F must be part of every solution to F and could thus be filtered in a simple preprocessing step. We call two constraints *equivalent* if they have the same arity and are satisfied by the same set of assignments.

Next, we define some standard Boolean constraints that are used throughout the paper. We write True or False for the arity-1 constraint that is satisfied if and only if its input variable is respectively 1 or 0. For non-negative integers p and $q \leq p$, we denote the arity-p constraint that takes the logical OR over p literals, of which q are negated, by $\text{OR}_{p,q}$. Hence, $\text{OR}_{p,q}$ denotes the constraint $(\neg x_1 \vee \ldots \vee \neg x_q \vee x_{q+1} \vee \ldots \vee x_p)$. We use shorthands OR_p for $\text{OR}_{p,0}$, and NOR_p for $\text{OR}_{p,p}$. We denote the constraint $(x_1 \oplus \ldots \oplus x_p = 1)$ by XOR_p and the constraint $(x_1 \oplus \ldots \oplus x_p = 0)$ by XNOR_p. We use shorthands $[\neq]$ for XOR_2 and $[=]$ for XNOR_2. We denote the constraint of arity p that is satisfied if and only if its p input variables are not all equal by NAE_p.

Now, we can define different classes of constraints. First, we call a constraint 0-*valid* or 1-*valid* if it is satisfied by the all-0 or the all-1 assignment respectively. We call a constraint *IHS-B*$^+$ if it can be expressed as a conjunction of OR_p constraints for integers p, the constraint $\text{OR}_{2,1}$, and the arity-1 constraint False. Likewise, we call constraints *IHS-B*$^-$ if they can be expressed as conjunction of NOR_p constraints for integers p, the constraint $\text{OR}_{2,1}$, and the arity-1 constraint True. We call a constraint *bijunctive* if it can be expressed in 2CNF (i.e.: as conjunction of OR_2, $\text{OR}_{2,1}$, and NOR_2 constraints). As subset of the bijunctive constraints, 2-monotone constraints are defined as those that can be expressed in the form $(a_1 \wedge \ldots \wedge a_p) \vee (\neg b_1 \wedge \ldots \wedge \neg b_q)$ for some $p, q \geq 0$. We call a constraint *affine* if it can be expressed as conjunction of linear constraints mod 2 (i.e.: as conjunction of XOR_p and XNOR_q constraints for integers $p \geq 1$ and integers $q \geq 1$). We call a constraint *weakly positive* (resp. *weakly negative*) if it can be expressed in CNF with all clauses containing at most one negated (resp. positive) variable.

We call a constraint set $\mathcal{F}$ 0-valid / 1-valid / bijunctive / 2-monotone / affine / weakly positive / weakly negative if every constraint $f \in \mathcal{F}$ satisfies the corresponding property. We call $\mathcal{F}$ IHS-B if all its constraints are IHS-B$^+$ or all its constraints are IHS-B$^-$ and we call a single constraint f IHS-B if it is either IHS-B$^+$ or IHS-B$^-$.

Finally, we note that we can use a reduction that transforms integer-weighted formulas into unweighted ones by duplicating constraint applications as many times as their weight, to show the following.

▶ **Lemma 2.1 (★).** *Let $\mathcal{F}$ be a constraint set and $c \geq 1$. If MinCSP($\mathcal{F}$) admits a polynomial-time c-essential detection algorithm, then so does Weighted MinCSP($\mathcal{F}$) if all weights are integers that are polynomial in the number of constraint applications.*

3 Positive results

To achieve our positive results, we establish conditions on a constraint set $\mathcal{F}$ that imply the existence of a c-essential detection algorithm for MinCSP($\mathcal{F}$). The statement below admits a similar proof to Theorem 4.1 in the earlier work by Bumpus et al. on essential vertices [6].

▶ **Lemma 3.1 (★).** *Let $\mathcal{F}$ be a constraint set and let $c \in \mathbb{R}_{\geq 1}$ be a constant. Suppose there is a polynomial-time algorithm that takes as input an $\mathcal{F}$-formula F and a constraint application C in it, and outputs a set of constraint applications P_C such that:*

1. *$C \notin P_C$; and*
2. *if X_C is a smallest set of constraints from $F - C$ such that $F - X_C$ is satisfiable, then $|P_C| \leq c \cdot |X_C|$; and*
3. *for every set X such that $F - X$ is satisfiable, $F - ((X \setminus \{C\}) \cup P_C)$ is also satisfiable,*

then, there is a polynomial-time $(c+1)$-essential detection algorithm for MinCSP($\mathcal{F}$).

In the proof of this statement, we show that such a set P_C is strictly larger than $c \cdot \mathrm{opt}(F)$ if C is a c-essential constraint application, and that C does not belong to every optimal solution if $|P_C| \leq c \cdot \mathrm{opt}(F)$. As such, for a given integer k, we can construct a c-essential detection algorithm by computing the set P_C for every constraint application C in F and returning the constraint applications for which $|P_C| > c \cdot k$.

In the full version of this paper, we show that a polynomial-time c-approximation algorithm for Weighted MinCSP($\mathcal{F}$) can be used to construct an algorithm that satisfies all three properties from Lemma 3.1. Khanna et al. have shown that such an algorithm exists when $\mathcal{F}$ is IHS-B [13, Theorem 2.13], so we conclude the following.

▶ **Corollary 3.2.** *Let $\mathcal{F}$ be an IHS-B constraint set. Then, there is a constant $c_{\mathcal{F}}$ for which MinCSP($\mathcal{F}$) admits a polynomial-time $c_{\mathcal{F}}$-essential detection algorithm.*

For bijunctive constraint sets $\mathcal{F}$, we can also use Lemma 3.1 to prove that a c-essential detection algorithm exists for MinCSP($\mathcal{F}$). This does require some more work and to do so, we make use of the well-known fact [3, 20] that formulas in 2CNF can be represented in graph-form as follows.

▶ **Definition 3.3.** *Let F be a 2CNF formula over variable set $\mathbf{X}$, or more generally, an $\mathcal{F}$-formula for some bijunctive constraint set $\mathcal{F}$. We construct a directed (multi)graph G_F as follows. For every variable $x \in \mathbf{X}$, we add x and $\neg x$ to the vertex set of G_F. For every clause $(\ell_1 \vee \ell_2)$ in the 2CNF representation of F with literals ℓ_1 and ℓ_2, we add the arcs $(\neg\ell_1, \ell_2)$ and $(\neg\ell_2, \ell_1)$ to G_F. We call the resulting graph the* implication graph *of F.*

We highlight that clauses may appear in multiple constraint applications of F and that this is reflected in the implication graph of F by allowing duplicate arcs in it. Furthermore, in our remaining arguments, we assume that we keep track of which clause in a formula corresponds to which arc in the implication graph. We exploit some of the structural properties of implication graphs to prove the following result.

▶ **Lemma 3.4 (★).** *Suppose that $\mathcal{F}$ is bijunctive so that every constraint $f \in \mathcal{F}$ can be written as conjunction of at most d disjunctions of two literals, for some constant d. Then, there is a polynomial-time $(2d^2+1)$-essential detection algorithm for MinCSP($\mathcal{F}$).*

Proof sketch. First note that we may assume w.l.o.g. that input formulas F are already given to us in suitable 2CNF representation when dealing with MinCSP($\mathcal{F}$) instances. Otherwise, we could, for every fixed bijunctive $\mathcal{F}$, provide a polynomial-time transformation using a hard-coded bijunctive representation of each constraint in $\mathcal{F}$. Thus, let F be an $\mathcal{F}$-formula represented in 2CNF in which every constraint is written as conjunction of at most d clauses. Let G_F be its implication graph, and let C be a constraint application in F that is expressed as $(a_1 \vee b_1) \wedge \ldots \wedge (a_\ell \vee b_\ell)$, where $a_1, \ldots a_\ell, b_1, \ldots, b_\ell$ are literals for some $\ell \leq d$.

To prove the lemma, we give a polynomial-time algorithm that computes a set P_C from F and C, after which we show that this set meets the three preconditions described in Lemma 3.1. To ensure that our algorithm is well-defined, we prove the claim below.

▷ Claim 3.5 (★). Let F' be any 2CNF formula that contains C and let $G_{F'}$ be the implication graph of F'. If there is an $i \in [\ell]$ such that $G_{F'}$ contains both an $(a_i, \neg a_i)$-path and a $(b_i, \neg b_i)$-path, then F' is unsatisfiable.

Proof sketch. We observe that an assignment s that satisfies F' and sets an arbitrary literal r to 1 must set all other literals reachable from r in $G_{F'}$ to 1 as well. Thus, if literals $\neg a_i$ and $\neg b_i$ are reachable from a_i and b_i respectively, any satisfying assignment to F' must set a_i and b_i to 0. Such an assignment cannot satisfy the clause C in F' that requires $(a_i \vee b_i)$. ◁

Now, P_C can be computed as follows.

- Construct the implication graph G_F of F.
- For all $i \in [\ell]$, compute a smallest subset of arcs in G_{F-f} that breaks all $(a_i, \neg a_i)$-paths in G_F and compute a smallest subset of arcs in G_{F-f} that breaks all $(b_i, \neg b_i)$-paths in G_F. If only one of these two sets exists, let T_i be this set, or, if both exist, let T_i be a minimum-size set among the two. (Note that at least one of these cuts must exist. If not, the implication graph G_C of the singular constraint application C would have to contain both an $(a_i, \neg a_i)$-path and a $(b_i, \neg b_i)$-path, which, by Claim 3.5, would imply C to be unsatisfiable. This would contradict our assumption that all individual constraints we consider are satisfiable.)
- Let T be $\bigcup_{i \in [\ell]} T_i$ and let P_C be the set of constraint applications in F with at least one of their corresponding implication arcs in T.

First, we observe that the construction above can be executed in polynomial time. The bottleneck is the second step in which a linear number of cut problems need to be solved. These are variants of the standard min-cut problem on directed graphs, and are easily seen to be solvable efficiently by a simple modification of a polynomial-time mincut algorithm, such as the Ford-Fulkerson algorithm [8].

It remains to prove that P_C satisfies the three properties specified in Lemma 3.1. By construction, P_C does not contain C, so it satisfies the first property.

For the other two properties, we start by noting that T is a set of arcs that, for every $i \in [\ell]$, breaks either all $(a_i, \neg a_i)$-paths or all $(b_i, \neg b_i)$-paths in G_F without including any arcs that correspond to a clause of C. We call such a set a *C-respecting arc set*. (Note that such sets

may include duplicates of arcs that correspond to a clause of C, but not those that specifically correspond to C.) Likewise, we call a set of constraint applications from F a *C-respecting constraint application set* if the union of their respective arcs in G_F is a C-respecting arc set. As such, P_C is a C-respecting constraint application set. We use this notation to show that P_C satisfies the second property specified in Lemma 3.1.

▷ Claim 3.6 (★). Let X_C be a smallest set of constraints from $F - C$ such that $F - X_C$ is satisfiable. Then, $|P_C| \leq 2d^2 \cdot |X_C|$.

Proof sketch. Since $F - X_C$ contains C, it follows from Claim 3.5 that X_C is a C-respecting constraint application set. By definition, the union T^* of all arcs in G_F that correspond to a clause in X_C is a C-respecting arc set in G_F. Since C contains at most d clauses that each introduce 2 arcs in G_F, we find that $|T^*| \leq 2d \cdot |X_C|$. Since every T_i breaks its respective paths optimally, we find that $\max_{i \in [\ell]} |T_i| \leq |T^*|$, so that the union T of all $\ell \leq d$ sets T_i has $|T| \leq d \cdot |T^*|$. With P_C being the set of constraint applications corresponding to the arcs in T, we find that $|P_C| \leq |T|$. Combining all inequalities we obtain $|P_C| \leq 2d^2 \cdot |X_C|$. ◁

Now, we show that P_C also satisfies the third property listed in Lemma 3.1.

▷ Claim 3.7 (★). For every set X such that $F - X$ is satisfiable, $F^* := F - ((X \setminus \{C\}) \cup P_C)$ is also satisfiable.

Proof sketch. Given an assignment s that satisfies $F - X$, we explain how to transform it into a satisfying assignment for F^*. Pick an arbitrary clause $(a_i \vee b_i)$ from C that is not yet satisfied by s and determine whether G_{F^*} contains no $(a_i, \neg a_i)$-path or that it contains no $(b_i, \neg b_i)$-paths. Since P_C is C-respecting, at least one of these statements is true and we assume w.l.o.g. that the former is true. Then, we modify s by setting a_i and all literals reachable from a_i in G_{F^*} to 1. We repeat this for as long as C contains unsatisfied clauses.

To see that s is a valid assignment after a single iteration, suppose for contradiction that s was modified to set both x and $\neg x$ to 1. Then, x and $\neg x$ are both reachable from a_i. However, the structure of implication graphs ensures that, for every arc (u, v), the arc $(\neg v, \neg u)$ also exists. Thus, $\neg a_i$ is reachable from x, which in turn is reachable from a_i, contradicting that G_{F^*} contains no $(a, \neg a_i)$-path.

Thus, every iteration ends with a valid assignment. Moreover, by setting a_i to 1, s satisfies C. Moreover, no previously satisfied clause $(x \vee y)$ becomes unsatisfied, by setting both $\neg x$ and $\neg y$ to 1: if, e.g., $\neg x$ is reachable from a_i, then so is y via the arc $(\neg x, y)$. Thus, setting $\neg x$ to 1 means that y will also be set to 1. Therefore, the described procedure strictly increases the number of satisfied clauses in each iteration and it stops when all clauses from C are satisfied, resulting in a satisfying assignment for G_{F^*}. ◁

This proves that the algorithm described above computes a set P_C in polynomial time that satisfies all three preconditions for Lemma 3.1. In particular, our algorithm satisfies these constraints with $c = 2d^2$. By Lemma 3.1, there is a polynomial-time $(2d^2 + 1)$-detection algorithm for MinCSP($\mathcal{F}$). ◀

We recall that FPT algorithms, parameterized by solution size, are known for MinCSP($\mathcal{F}$) on bijunctive constraint sets $\mathcal{F}$ if and only if the so-called Gaifman graph of $\mathcal{F}$ is $2K_2$-free [15]. So, by a result analogous to one from the original work [6, Theorem 5.1] on c-essential detection, Lemma 3.4 effectively shows that we can reduce the search-space of these FPT algorithms. In particular, it shows that their exponential runtime dependence on the total size of an optimal solution can be improved to depend only on the number of constraint applications in an optimal solution that are *not* c-essential.

4 Hardness results

Our hardness results all follow the same overarching structure. For a given category of constraint sets, we first give a canonical set of constraints $\mathcal{F}$ from that category and prove for the corresponding MinCSP($\mathcal{F}$) problem that c-essential detection is hard for all constants $c \in \mathbb{R}_{\geq 1}$. Then, we show that there is a reduction from this problem to every other MinCSP problem for constraint sets in the same category. This reduction is constructive. To achieve this, we show that every constraint set from a given category can be used to express the constraints from the canonical set. Under the right restrictions, this immediately shows the reducibility from the canonical set to the other sets in the category. These restrictions are given in the notion of *essential implementations* as defined below.

▶ **Definition 4.1.** *Let $f : \{0,1\}^p \to \{0,1\}$ be a constraint over p variables. A collection $\mathcal{C}$ of constraint applications $C_1, \ldots, C_m$ over variables $\mathbf{X} = \{x_1, \ldots, x_p\}$ and dummy variables $\mathbf{Y} = \{y_1, \ldots, y_q\}$ is an* essential implementation *of the constraint application $(f, \mathbf{X})$ if:*

(I1) *an assignment to $\mathbf{X}$ satisfies $(f, \mathbf{X})$ if and only if there is an extension of this assignment to $\mathbf{X} \cup \mathbf{Y}$ that satisfies all constraint applications $C_1, \ldots, C_m$; and*

(I2) *for every assignment to $\mathbf{X}$, there is an extension to $\mathbf{X} \cup \mathbf{Y}$ that satisfies $C_2, \ldots, C_m$. (Note the exclusion of C_1, which we call the* head *of the implementation.)*

We say that a constraint set $\mathcal{F}$ *essentially implements* a constraint f if f admits an essential implementation via constraint applications that are each from $\mathcal{F}$. We denote this as $\mathcal{F} \xrightarrow{\text{ess.}} f$. We say that a constraint set $\mathcal{F}_1$ essentially implements a constraint set $\mathcal{F}_2$ if $\mathcal{F}_1$ essentially implements each of the constraints from $\mathcal{F}_2$. We denote this as $\mathcal{F}_1 \xrightarrow{\text{ess.}} \mathcal{F}_2$. In both arrow notations, if $\mathcal{F}_1$ is a singleton set, we may replace it by just its singleton element.

This notion of essential implementations is stricter than, e.g., the notion of pp-definitions, which are similarly defined but without requiring Property **(I2)**. Such definitions facilitate reductions between formulas of different constraint sets that preserve the satisfiability of the formulas, as for example used by Schaefer [21]. Even more similar is the notion of α-implementations as defined by Khanna et al. [13], especially those that are both "strict" and "perfect". These implementations only differ from our Definition 4.1 by allowing different assignments to $\mathbf{X}$ that do not satisfy f to extend to an assignment on $\mathbf{X} \cup \mathbf{Y}$ that leaves a different (but still singular) constraint application C_i unsatisfied. This uncertainty in which constraint application remains unsatisfied leads to changes in the structure of the solution space that makes these implementations unsuitable for our purposes. However, the authors use their definition of α-implementations to construct reductions that preserve the constant-factor approximability of MinCSP($\mathcal{F}$).

Our definition of essential implementations allows for a reduction that also preserves the exact approximation-ratio for constant factor-approximable instances. In fact, it even preserves the entire space of c-approximate solutions in a predictable and useful manner. This is needed to prove the following result.

▶ **Lemma 4.2 (★).** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be constraint sets such that $\mathcal{F}_1 \xrightarrow{\text{ess.}} \mathcal{F}_2$, and let $c \in \mathbb{R}_{\geq 1}$ be a constant. If MinCSP($\mathcal{F}_1$) admits a polynomial-time c-essential detection algorithm, then MinCSP($\mathcal{F}_2$) also admits a polynomial-time c-essential detection algorithm.*

Proof sketch. We prove the statement by showing how to reduce an $\mathcal{F}_2$-formula to a weighted $\mathcal{F}_1$-formula in such a way that a polynomial-time c-essential detection algorithm on the $\mathcal{F}_1$-formula can be used to detect c-essential constraint applications in the $\mathcal{F}_2$-formula. By Lemma 2.1, this suffices to prove the lemma.

Let F_2 be an $\mathcal{F}_2$-formula with constraint applications $C_1, \ldots, C_\ell$ over variables $\mathbf{X} = \{x_1, \ldots, x_n\}$. To transform F_2 into a weighted $\mathcal{F}_1$-formula F_1, we replace every C_i in F_2 by an essential implementation $\mathcal{C}_i$ of constraint applications from $\mathcal{F}_1$ over variables $\mathbf{X} \cup \mathbf{Y}_i$. We denote the head of $\mathcal{C}_i$ by C_i^*. We give all constraint applications a weight of $\lceil c \cdot \ell \rceil + 1$, except for the heads $C_1^*, \ldots, C_\ell^*$ that each receive a weight of 1. We define $\mathbf{Y} := \bigcup_{i \in [\ell]} \mathbf{Y}_i$.

The first step towards proving the correctness of this reduction is to observe that non-head constraint applications in F_1 do not appear in any c-approximate solution. After all, it follows from Property **(I2)** that the set of heads $H := \{C_1^*, \ldots, C_\ell^*\}$, with total weight ℓ, makes $F - H$ satisfiable, whereas every other constraint has weight $\lceil c \cdot \ell \rceil + 1$. Now, the following claim effectively shows that the set H spans the exact same space of c-approximate solutions to F_1 as the set of original constraint applications $C_1, \ldots, C_\ell$ does for F_2.

▷ Claim 4.3. Let S be a set of constraint applications in F_2 and let $S_H \subseteq H$ be the corresponding set of heads in F_1. Then, $F_2 - S$ is satisfiable if and only if $F_1 - S_H$ is satisfiable.

Proof. For one direction, let s be an assignment on $\mathbf{X}$ that satisfies $F_2 - S$. For every $i \in [\ell]$ such that $C_i \in S$, assignment s can be extended to an assignment on $\mathbf{X} \cup \mathbf{Y}_i$ that satisfies all constraint applications in $\mathcal{C} \setminus \{C_i^*\}$, due to Property **(I2)**. For every $i \in [\ell]$ such that $C_i \notin S$, since s satisfies C_i, assignment s can be extended to an assignment on $\mathbf{X} \cup \mathbf{Y}_i$ that satisfies all constraint applications in $\mathcal{C}_i$, due to Property **(I1)**. This covers all constraint applications in $F_1 - S_H$, showing that an extension of s to $\mathbf{X} \cup \mathbf{Y}$ exists that satisfies $F_1 - S_H$.

For the other direction, observe that any assignment s' on $\mathbf{X} \cup \mathbf{Y}$ that satisfies $F_1 - S_H$, satisfies the constraint applications in $\mathcal{C}_i$ for every i such that $C_i \notin S$. Thus, by Property **(I1)**, the restriction of s' to $\mathbf{X}$ satisfies $F_2 - S$. ◁

Since every head has weight 1, it follows that $\mathrm{opt}(F_1) = \mathrm{opt}(F_2)$. Now we show how to use a polynomial-time c-essential detection algorithm $\mathcal{A}$ for Weighted MinCSP($\mathcal{F}_1$) to construct a polynomial-time c-essential detection algorithm for MinCSP($\mathcal{F}_2$).

Given an $\mathcal{F}_2$-formula F_2 and an integer k, we start by transforming it into a weighted $\mathcal{F}_1$-formula F_1 using the reduction above; the weights are polynomial. Then, we apply $\mathcal{A}$ to F_1 and let D_1 denote its output. If D_1 contains non-head constraint applications of F_1, we return the empty set. Otherwise, we return the set of constraint applications C_i for which the corresponding head C_i^* is in D_1. In either case, let D_2 be the output of our algorithm.

It remains to show that the above is a c-essential detection algorithm for F_2. First, suppose that $\mathrm{opt}(F_1) = \mathrm{opt}(F_2) \leq k$. Since D_1 satisfies Property **(G1)** it is a subset of some optimal solution S_1 to F_1. S_1, and by extension D_1 must consist entirely of heads. Then, the set S_2 of constraint applications in F_2 for which the corresponding head is in S_1 must be a superset of D_2. By Claim 4.3, S_2 is an optimal solution to F_2, showing that D_2 satisfies Property **(G1)**.

Next, suppose that $\mathrm{opt}(F_1) = \mathrm{opt}(F_2) = k$. Since D_1 satisfies Property **(G2)**, it contains all c-essential constraint applications from F_1. Since the set of heads was shown to preserve the entire space of c-approximate solutions, a constraint application C_i in F_2 is c-essential if and only if its corresponding head C_i^* is c-essential in F_1. Since D_2 contains the set of heads corresponding to D_1, it contains all c-essential constraint applications in F_2. ◀

Next, we prove that the notion of essential implementations is a transitive relation.

▶ **Lemma 4.4.** *Let $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$ be constraint sets. If $\mathcal{F}_1 \xrightarrow{\text{ess.}} \mathcal{F}_2$ and $\mathcal{F}_2 \xrightarrow{\text{ess.}} \mathcal{F}_3$, then $\mathcal{F}_1 \xrightarrow{\text{ess.}} \mathcal{F}_3$.*

Proof sketch. Let $(f_3, \mathbf{X})$ be an arbitrary constraint application from $\mathcal{F}_3$ over variables $\mathbf{X} = \{x_1, \ldots, x_p\}$. To prove the lemma, we show that $\mathcal{F}_1 \xrightarrow{\text{ess.}} f_3$.

Since $\mathcal{F}_2 \xrightarrow{\text{ess.}} \mathcal{F}_3$, there is an essential implementation of $(f_3, \mathbf{X})$ consisting of constraint applications $C_1, \ldots, C_m$ from $\mathcal{F}_2$ over variables $\mathbf{X}$ and shared dummy variables $\mathbf{Y}$. Let C_1 be the head of this implementation. Since $\mathcal{F}_1 \xrightarrow{\text{ess.}} \mathcal{F}_2$, there is, for each $i \in [m]$, an essential implementation $\mathcal{C}_i$ of C_i consisting of constraint applications from $\mathcal{F}_1$ over variables $\mathbf{X} \cup \mathbf{Y}$ and dummy variables $\mathbf{Z}_i$. Let $C_i^* \in \mathcal{C}_i$ denote the head of $\mathcal{C}_i$. Note that the constraint applications in one $\mathcal{C}_i$ share one set of dummy variables $\mathbf{Z}_i$, but that this set differs between different implementations $\mathcal{C}_i$.

We show that the concatenation of all constraint applications in $\mathcal{C}_i, \ldots, \mathcal{C}_m$ is an essential implementation of f_3 with head C_1^*. It is easily verified that this satisfies Property **(I1)**, so we continue by showing the more interesting fact that it satisfies Property **(I2)**.

Consider an arbitrary assignment s on $\mathbf{X}$. By Property **(I2)** of implementation $C_1, \ldots, C_m$, it extends to an assignment s' on $\mathbf{X} \cup \mathbf{Y}$ that satisfies $C_2, \ldots, C_m$. By Property **(I1)** of the implementations $\mathcal{C}_2, \ldots, \mathcal{C}_m$, s' extends to an assignment on $\mathbf{X} \cup \mathbf{Y} \cup (\mathbf{Z} \setminus \mathbf{Z}_1)$ that satisfies all constraint applications in $\mathcal{C}_2, \ldots, \mathcal{C}_m$. By Property **(I2)** of the implementation $\mathcal{C}_1$, every assignment on $\mathbf{X} \cup \mathbf{Y}$ – including s' – extends to an assignment on $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}_1$ that satisfies $\mathcal{C}_1 \setminus \{C_1^*\}$. Thus, s extends to an assignment s' that in turn extends to an assignment on $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$ that satisfies $\mathcal{C} \setminus \{C_1^*\}$, proving that $\mathcal{C}$ satisfies Property **(I2)**. ◀

Next, in Sections 4.1–4.3, we prove Cases 3, 4, and 5 from Theorem 1.1. Due to space limitations, the proofs in Sections 4.1 and 4.3 are deferred to the full version. The proofs in Section 4.2 also showcase the general proving strategies used in the other two sections, but we remark that each case of the dichotomy represents a significant effort in tailoring these strategies to their specific setting. Although we consider the introduction and use of essential implementations to be one of our main contributions, this also means that our proofs show only two examples of specific essential implementations.

4.1 Affine constraint sets

We pick $\{\text{XOR}_4, \text{XNOR}_4\}$ as canonical constraint set for the category of affine constraint sets that are neither 0-valid, 1-valid nor bijunctive. First, we prove that c-essential Detection for MinCSP$(\{\text{XOR}_4, \text{XNOR}_4\})$ is NP-hard for every $c \in \mathbb{R}_{\geq 1}$ under the Unique Games Conjecture (UGC). As starting point for our hardness reduction, we use the fact that distinguishing between two regimes of solution sizes for gap instances of MinCSP$(\{[=], [\neq]\})$ is NP-hard under the UGC [14]. We give a reduction that transforms an $\{[=], [\neq]\}$-formula F_1 into an $\{\text{XOR}_4, \text{XNOR}_4\}$-formula F_2 in such a way that we can use the output of a c-essential detection algorithm on F_2 to determine whether F_1 admits a small or large optimal solution. For example, each arity-2 constraint over $[\neq]$ turns into an arity-4 constraint over XOR_4 by inserting two global variables z_1, z_2 to which the constraint is additionally applied. Their contributions cancel whenever $z_1 = z_2$ and we add a medium-weight constraint enforcing this behavior. This proves c-essential detection to be hard for MinCSP$(\{\text{XOR}_4, \text{XNOR}_4\})$ and the details of this reduction are given in the full version.

Then, if $\mathcal{F}$ is any affine constraint set that is neither 0-valid, 1-valid nor bijunctive, we show that $\mathcal{F} \xrightarrow{\text{ess.}} \{\text{XOR}_4, \text{XNOR}_4\}$. To start, we use an existing result that shows that $\mathcal{F}$ can be written as conjunction of XOR and XNOR constraints that are each implemented by $\mathcal{F}$ [13, Lemma 4.18]. It is easily seen that the provided implementations in this work are even essential. Then, since $\mathcal{F}$ is not bijunctive, it implements at least one XOR or XNOR constraint over 3 or more variables. We also note that XNOR_q is 0-valid for all integers q and that XOR_p

and XNOR_q are 1-valid for odd values of p and even values of q. Combining these insights with several sequences of new implementations, we can show that $\mathcal{F} \xrightarrow{\text{ess.}} \{\mathrm{XOR}_4, \mathrm{XNOR}_4\}$, the details of which are given in the full version.

The two results above combine to prove Case 3 of Theorem 1.1.

4.2 Weakly positive and weakly negative constraint sets

In this section, we prove Case 4 of Theorem 1.1. Throughout this section, we assume to be working with weakly positive constraint sets, rather than weakly negative ones. Proofs for weakly negative constraint sets follow symmetrically. We pick $\mathcal{F}_{\text{can}} := \{\mathrm{OR}_{2,1}, \mathrm{OR}_{3,1}, \textsc{True}, \textsc{False}\}$ as canonical constraint set and we start by showing that c-essential Detection for $\mathrm{MinCSP}(\mathcal{F}_{\text{can}})$ is NP-hard in Section 4.2.1. Then, in Section 4.2.2, we show for any weakly positive constraint set $\mathcal{F}$ that is neither 0-valid, 1-valid, bijunctive, nor IHS-B, that $\mathcal{F} \xrightarrow{\text{ess.}} \mathcal{F}_{\text{can}}$. (Note that these conditions imply $\mathcal{F}$ is not affine either.)

4.2.1 Hardness of c-essential detection for the canonical problem

We give a reduction from the Set Cover optimization problem. In this problem, a universe U is given together with a collection $\mathcal{S}$ of subsets of U, and the goal is to determine the size of a smallest subset $Y \subseteq \mathcal{S}$ such that every $u \in U$ is contained in at least one set in Y. We call such a set Y (of any size) a *set cover* of $(U, \mathcal{S})$, and we denote the size of a smallest set cover of $(U, \mathcal{S})$ by $\mathrm{opt}(U, \mathcal{S})$. The following hardness result is known.

▶ **Lemma 4.5** ([2, Theorem 22.31]). *For every sufficiently small constant $\varepsilon > 0$, the following holds. Unless P=NP, there is no polynomial-time algorithm that takes an integer $t \geq 1$ and a Set Cover instance $(U, \mathcal{S})$ as input and distinguishes between the following two cases:*

(i) $\mathrm{opt}(U, \mathcal{S}) \leq t$;

(ii) $\mathrm{opt}(U, \mathcal{S}) > \frac{t}{4\sqrt{\varepsilon}}$.

We use this result to prove the statement below.

▶ **Lemma 4.6.** *Unless P=NP, there is no constant $c \in \mathbb{R}_{\geq 1}$ for which a polynomial-time c-essential detection algorithm exists for MinCSP($\{\mathrm{OR}_{2,1}, \mathrm{OR}_{3,1}, \textsc{True}, \textsc{False}\}$).*

Proof. Let $\mathrm{HORN}_p := \left(\bigcup_{i \in \{2 \ldots p\}} \mathrm{OR}_{i,1}\right) \cup \{\textsc{True}, \textsc{False}\}$, so that the lemma states hardness of c-essential detection for $\mathrm{MinCSP}(\mathrm{HORN}_3)$. We start by showing that the existence of a polynomial-time c-essential detection algorithm for Weighted $\mathrm{MinCSP}(\mathrm{HORN}_\infty)$ implies that P=NP. We assume that such an algorithm exists for an arbitrary value of $c \in \mathbb{R}_{\geq 1}$ and argue that it can be used to solve the distinguishing task from Lemma 4.5 in polynomial time. Afterwards, we explain how to extend this result to HORN_3.

Now, let $(U, \mathcal{S})$ be a Set Cover instance, and let $t \geq 1$ be an arbitrary integer. We let m denote the maximum size of a set in $\mathcal{S}$, and show how to transform $(U, \mathcal{S})$ into a weighted HORN_{m+1}-formula F.

We start by introducing two variables z and w and adding constraint applications $\textsc{True}(w)$ with weight $c(t+1)+1$ and $\textsc{False}(z)$ with weight $t+1$ to F. Then, for every set $S_j \in \mathcal{S}$, we introduce a variable y_j and we add a constraint application $(\neg y_j \vee z)$ to F with weight 1.

Next, for every $u_i \in U$, we introduce a variable x_i and we add $(x_i \vee \neg w)$ to F with weight $c(t+1)+1$. Additionally, if we let $S_{j_1}, \ldots, S_{j_\ell}$ be the collection of sets that u_i appears in, we add $(\neg x_i \vee y_{j_1} \vee \ldots \vee y_{j_\ell})$ to F with weight $c(t+1)+1$.

This concludes the reduction. Observe that the resulting formula F is a HORN$_{m+1}$-formula whose size is polynomial in the total size of the original Set Cover instance. We continue by proving the correctness of our reduction, starting with the following claim.

▷ Claim 4.7. A collection $\mathcal{C} := S_{j_1}, \ldots, S_{j_r}$ is a set cover of $(U, \mathcal{S})$ if and only if the collection of constraint applications $X_\mathcal{C} := \{(\neg y_{j_1} \vee z), \ldots, (\neg y_{j_r} \vee z)\}$ is such that $F - X_\mathcal{C}$ is satisfiable.

Proof. For one direction, suppose that $\mathcal{C}$ is a set cover of $(U, \mathcal{S})$. Next, consider the assignment s to the variables of F that sets w to 1, z to 0, all variables x_i to 1, the variables y_j for which $S_j \in \mathcal{C}$ to 1, and the variables y_j for which $S_j \notin \mathcal{C}$ to 0.

Clearly, True(w) and False(z) are satisfied by this assignment. All constraint applications of the form $(\neg y_j \vee z)$ in $F - X_\mathcal{C}$ are also satisfied because all variables y_j for which $S_j \notin \mathcal{C}$ are set to 0. Finally, note that, because $\mathcal{C}$ is a set cover of U, every constraint application of the form $(\neg x_i \vee y_{j_1} \vee \ldots \vee y_{j_\ell})$ contains at least one y_j for which $S_j \in \mathcal{C}$. All such variables are set to 1, so all constraint applications of this form in $F - X_\mathcal{C}$ are also satisfied by s.

For the other direction, suppose that $F - X_\mathcal{C}$ is satisfiable and let s be a satisfying assignment. To satisfy True(w) and False(z), we must have $s(w) = 1$ and $s(z) = 0$. Then, to satisfy all constraint applications of the form $(x_i \vee \neg w)$, we must have $s(x_i) = 1$ for all variables x_i. Likewise, to satisfy all constraint applications in $F - X_\mathcal{C}$ of the form $(\neg y_j \vee z)$, we must have $s(y_j) = 0$ for all variables y_j with $S_j \notin \mathcal{C}$.

Next, consider an arbitrary constraint application of the form $(\neg x_i \vee y_{j_1} \vee \ldots \vee y_{j_\ell})$. Since s satisfies this constraint application but also has $s(x_i) = 1$ it must set at least one of the variables $y_{j_1}, \ldots, y_{j_\ell}$ to 1. However, since s sets all variables y_j for which $S_j \notin \mathcal{C}$ to 0, at least one of the variables $y_{j_1}, \ldots, y_{j_\ell}$ must correspond to a set from $\mathcal{S}$ that is in $\mathcal{C}$. Thus, we obtain that for every variable x_i, the constraint application $(\neg x_i \vee y_{j_1} \vee \ldots \vee y_{j_\ell})$ contains at least one variable y_j for which $S_j \in \mathcal{C}$. This means that $\mathcal{C}$ is a set cover of $(U, \mathcal{S})$. ◁

Now, we define $\varepsilon := \frac{1}{65c^2}$ and use the claim above to show that a c-essential detection algorithm for MinCSP(HORN$_p$) on input F can be used to determine whether the original Set Cover input $(U, \mathcal{S})$ satisfies $\text{opt}(U, \mathcal{S}) \leq t$ or $\text{opt}(U, \mathcal{S}) > \frac{t}{4\sqrt{\varepsilon}}$. To this end, let S be the result of a c-essential detection algorithm for MinCSP(HORN$_p$) on input F with $k = t + 1$. To recognize the first case, consider the following claim.

▷ Claim 4.8. If $\text{opt}(U, \mathcal{S}) \leq t$, then S does not contain the constraint False(z).

Proof. Let $\mathcal{C} \subseteq \mathcal{S}$ be a set cover of $(U, \mathcal{S})$ of size at most t. Then, by Claim 4.7, the corresponding set $X_\mathcal{C}$ is such that $F - X_\mathcal{C}$ is satisfiable. Since all constraint applications in $X_\mathcal{C}$ have weight 1, this constraint set is a solution to F of weight t. Thus, $\text{opt}(F) \leq t < k$. As such, S must be a subset of an optimal solution to F to satisfy Property **(G1)**. Since an optimal solution of F has weight $\leq t$ and the constraint application False(z) has weight $t + 1$, False(z) is not in any optimal solution and by extension also not in S. ◁

We continue by showing how the set S differs for the other case.

▷ Claim 4.9. If $\text{opt}(U, \mathcal{S}) > \frac{t}{4\sqrt{\varepsilon}}$, then S does contain the constraint False(z).

Proof. First, we argue that the singleton set $\{\text{False}(z)\}$ of weight $t + 1$ is a solution to F. To see this, note that every constraint application in $F - \{\text{False}(z)\}$ contains at least one positive literal so that the all-one assignment satisfies it. We continue by showing that solutions to F that do not contain False(z) have a weight of more than $c \cdot (t + 1)$.

Let X be an arbitrary such solution and suppose for contradiction that it has a weight of $\leq c \cdot (t+1)$. Recall that the only constraint applications that do not individually exceed this weight are $\textsc{False}(z)$ and the weight-1 constraint applications of the type $(\neg y_i \vee z)$. As such, X consists solely of such weight-1 constraints.

However, by Claim 4.7, a solution that only consists of these weight-1 constraint applications corresponds to a set cover of $(U, \mathcal{S})$ of equal size. Therefore, X must be at least of size $\text{opt}(U, \mathcal{S})$, which, by assumption, is larger than $\frac{t}{4\sqrt{\varepsilon}}$. We defined $\varepsilon = \frac{1}{65c^2}$, so by rewriting the inequality $\varepsilon < \frac{1}{64c^2}$, we obtain that $\frac{1}{4\sqrt{\varepsilon}} > 2c$, which in turn yields that $\frac{t}{4\sqrt{\varepsilon}} > 2ct$. Since $2t \geq t+1$ for all $t \geq 1$, we get that $|X| \geq \text{opt}(U, \mathcal{S}) > \frac{t}{4\sqrt{\varepsilon}} > c \cdot (t+1)$. This contradicts our assumption that X has weight at most $c \cdot (t+1)$.

We conclude that all solutions to F that do not contain $\textsc{False}(z)$ are more than c times as heavy as $\textsc{False}(z)$ itself. This means that $\{\textsc{False}(z)\}$ is an optimal solution of weight $(t+1)$ and even that $\textsc{False}(z)$ is c-essential. As $k = t+1 = \text{opt}(F)$, S must contain all c-essential constraint applications to satisfy Property **(G1)**. Hence, S contains $\textsc{False}(z)$. ◁

Now, the two claims above indicate how a polynomial-time c-essential detection algorithm for F can be used to distinguish between $\text{opt}(U, \mathcal{S}) \leq t$ and $\text{opt}(U, \mathcal{S}) > \frac{t}{4\sqrt{\varepsilon}}$ in polynomial time as well: after running such an algorithm, it suffices to check whether the output contains $\textsc{False}(z)$. By Lemma 4.5, this distinction is NP-hard to make. As $c \in \mathbb{R}_{\geq 1}$ was chosen arbitrarily, this shows that c-essential detection for Weighted MinCSP(HORN_∞) is NP-hard for every $c \in \mathbb{R}_{\geq 1}$. By Lemma 2.1, the same holds for the unweighted variant of the problem.

It remains to show that the same is true for MinCSP(HORN_3). To this end, we show how to transform the HORN_{m+1}-formula F into a HORN_3-formula whose size is polynomial in the size of F. We note that, for any integer $p \geq 2$, $\{\text{OR}_{2,1}, \text{OR}_{3,1}\} \xrightarrow{\text{ess.}} \text{OR}_{p,1}$. This can be seen by inductively applying the following claim.

▷ Claim 4.10. Let $p \geq 4$. Then $\{\text{OR}_{p-1,1}, \text{OR}_{3,1}\} \xrightarrow{\text{ess.}} \text{OR}_{p,1}$.

Proof. We show that $(\neg x_1 \vee x_2 \vee \ldots \vee x_{p-2} \vee v) \wedge (\neg v \vee x_{p-1} \vee x_p)$, with dummy variable v and the first clause as its head, is an essential implementation of $(\neg x_1 \vee x_2 \vee \ldots \vee x_p)$. It is easily seen that this implementation satisfies Property **(I1)** by verifying that $(\neg x_1 \vee x_2 \vee \ldots \vee x_{p-2} \vee v) \wedge (\neg v \vee x_{p-1} \vee x_p)$ is satisfied if and only if $(\neg x_1 \vee x_2 \vee \ldots \vee x_p)$ is satisfied.

Next, note that every assignment to the variables $x_1, \ldots, x_p$ extends to a satisfying assignment for $(\neg v \vee x_{p-1} \vee x_p)$ by setting v to 0. Hence, Property **(I2)** is satisfied. ◁

Now, a single essential implementation of $\text{OR}_{p,1}$ can be made using $\mathcal{O}(p)$ applications of $\text{OR}_{2,1}$ and $\text{OR}_{3,1}$, since the construction in Claim 4.10 shows that we can build an $\text{OR}_{p,1}$ application from one (constant-size) $\text{OR}_{3,1}$ application and only one $\text{OR}_{p-1,1}$ application.

To transform F into a HORN_3-formula, we replace every HORN_p application in it with $p > 3$ by an essential implementation of $\text{OR}_{2,1}$ and $\text{OR}_{3,1}$ applications in which the head receives weight 1 and the other applications receive weight $c(t+1)+1$. Then, the same logic as in Lemma 4.2 shows that a polynomial-time c-essential detection algorithm for the resulting HORN_3-formula can be used to perform c-essential detection for the original formula F in polynomial time. This, in turn, was shown to imply P=NP. ◀

4.2.2 Essential implementation of the canonical constraint set

This section is dedicated to proving the following result.

▶ **Lemma 4.11.** *Let $\mathcal{F}$ be a constraint set that is neither 0-valid, nor 1-valid nor IHS-B. If $\mathcal{F}$ is weakly positive, then $\mathcal{F} \xrightarrow{\text{ess.}} \{\text{OR}_{2,1}, \text{OR}_{3,1}, \textsc{True}, \textsc{False}\}$.*

Proof sketch. Using a characterization of weakly positive constraint sets [13, Lemma 4.20], we can prove that $\mathcal{F} \xrightarrow{\text{ess.}} \{\textsc{True}, \textsc{False}\}$. Then, we can follow a sequence of implementations by Khanna et al. almost directly to show that $\mathcal{F} \cup \{\textsc{True}, \textsc{False}\} \xrightarrow{\text{ess.}} \{\text{OR}_{2,1}, \text{OR}_{3,1}\}$. Almost all implementations in the proof of Lemma 7.18 in their work happen to be essential as well [13]. The only exception is that their provided implementation $(\neg x \vee y) \wedge (\neg y \vee x)$ of $(x = y)$ is not essential. Note that it does not satisfy Property **(I2)**. Using a more complicated implementation, we can however still show that $\text{OR}_{2,1} \xrightarrow{\text{ess.}} [=]$. In particular, we note that $(\neg v_1 \vee v_2) \wedge (\neg x \vee v_1) \wedge (\neg y \vee v_1) \wedge (\neg v_2 \vee x) \wedge (\neg v_2 \vee y)$ is an essential implementation of $(x = y)$ with dummy variables v_1 and v_2 and the first clause as head. Using a simple case distinction, one can easily check that it satisfies Property **(I1)**. Moreover, any assignment to x and y can be extended to satisfy the last four clauses by setting v_1 to 1 and v_2 to 0, showing that the implementation satisfies Property **(I2)**. ◀

Now, Lemmas 4.6 and 4.11 combine to prove Case 4 of Theorem 1.1.

4.3 All other constraint sets

For constraint sets $\mathcal{F}$ that do not belong to any of the categories specified in the first four cases of Theorem 1.1, we pick $\{\text{NAE}_5, [\neq], [=]\}$ as the canonical constraint set. First, we prove that c-essential Detection for MinCSP$(\{\text{NAE}_5, [\neq], [=]\})$ is NP-hard for every $c \in \mathbb{R}_{\geq 1}$, even on formulas F with $\text{opt}(F) = 1$. To achieve this, we give a reduction that transforms a 3-SAT instance F_1 into a weighted $\{\text{NAE}_5, [\neq], [=]\}$-formula F_2.

We replace every 3CNF clause in F_1 by a large-weight NAE_5-constraint application in F_2 by supplementing it with two global dummy variables w_1 and w_2. We use large-weight $[\neq]$ constraints to model negated variables in the original 3CNF clauses. Then, including a weight-1 constraint application that requires that $(w_1 = w_2)$ ensures that F_1 is satisfiable if and only if F_2 is satisfiable. Moreover, if F_1 is not satisfiable, we can make F_2 satisfiable by removing the constraint application $(w_1 = w_2)$. Then, the remaining NAE_5-constraint applications in F_2 can indeed be satisfied by setting w_1 and w_2 unequal. Since all other constraint applications have a large weight, this makes $(w_1 = w_2)$ essential and it means that we can determine whether F_1 is satisfiable from the output of a c-essential detection algorithm on F_2. The details of this are given in the full version.

Secondly, if $\mathcal{F}$ is any constraint set that is neither 0-valid, 1-valid, IHS-B, bijunctive, affine, weakly positive, nor weakly negative, we show that $\mathcal{F} \xrightarrow{\text{ess.}} \{\text{NAE}_5, [\neq], [=]\}$. To this end, we extend a reduction by Schaefer, which is split into two cases, depending on whether or not all constraints in $\mathcal{F}$ are complementive [21]. (A constraint is complementive if the complement of a satisfying assignment is also satisfying.) If not, we can – with a few minor modifications and extensions – follow Schaefer's sequence of implementations rather directly as it consists mostly of implementations that are already essential. For complementive constraint sets however, we provide a new sequence of implementations to ensure they are essential. The details of this are given in the full version and, together with the hardness of c-essential Detection for MinCSP$(\{\text{NAE}_5, [\neq], [=]\})$, this proves Case 5 of Theorem 1.1.

5 Conclusion and Discussion

We extended the framework of search-space reduction via essential objects beyond the setting of graph problems for which it was originally defined. Investigating the task of detecting c-essential constraint applications for MinCSP$(\mathcal{F})$ has resulted in a dichotomy

that characterizes for which constraint sets $\mathcal{F}$ this is polynomial-time solvable. We note the similarity between our dichotomy theorem and the dichotomy theorem from Bonnet et al. [5] regarding the constant-factor approximability of MinCSP($\mathcal{F}$) in FPT time. Both dichotomies yield positive results for exactly the same classes of constraint sets. We leave it to future work to investigate whether this is a coincidence or a consequence of a deeper connection between the two tasks.

Other directions for follow-up research include extending the results of this dichotomy to CSPs beyond the Boolean domain. In fact, successfully applying the idea of c-essentiality to a problem domain different from graph theory has solidified the framework as a robust method by which meaningful and non-trivial new results can be attained across problem domains. As such, one could also attempt to find results using it in even different areas such as scheduling or computational geometry.

While our dichotomy theorem characterizes when c-essential detection is possible for *some* constant c, it does not shed any light on what the smallest value of c is for which this task is tractable, nor on the relation between this value and the choice of constraint set $\mathcal{F}$. We leave this to future work. A concrete question in this direction is to determine the smallest constant $c_{\mathcal{F}}$ for which $c_{\mathcal{F}}$-essential Detection for MinCSP($\mathcal{F}$) is possible for bijunctive constraint languages $\mathcal{F}$. Our approach in Lemma 3.4 of solving several cut problems in the implication graph gives a guarantee of $c_{\mathcal{F}} = (2d^2 + 1)$ whenever each constraint in $\mathcal{F}$ can be expressed as a 2CNF-formula with d clauses, but we do not expect this to be best-possible. Perhaps a discrete relaxation of the cut problem, like those studied in earlier work on Almost 2-SAT [9, 10], can provide a better bound?

A final open question concerns the theoretical basis for the impossibility of $c_{\mathcal{F}}$-essential Detection for affine constraint languages $\mathcal{F}$. We currently rely on the Unique Games Conjecture to rule out $c_{\mathcal{F}}$-detection algorithms for this family, via known lower bounds for MinCSP($\{[=], [\neq]\}$) [14]. Can the same lower bound be established relative to the standard assumption P $\neq$ NP, for example via known hardness of approximation results for Nearest Codeword [4]?

References

1 Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. Technical Report 16-44, ZIB, Takustr.7, 14195 Berlin, 2016. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-60370.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090.

3 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

4 Vijay Bhattiprolu, Venkatesan Guruswami, and Xuandi Ren. PCP-free APX-hardness of nearest codeword and minimum distance. *Electron. Colloquium Comput. Complex.*, TR25, 2025. URL: https://eccc.weizmann.ac.il/report/2025/029.

5 Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of Boolean MinCSPs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, Aarhus, Denmark, August 22-24, 2016*, volume 57 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.18.

6 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. *SIAM J. Discret. Math.*, 38(3):2392–2415, 2024. doi:10.1137/23M1589347.

7 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

8 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

9 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, lp-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. doi:10.1137/140962838.

10 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/All CSPs, half-integral A-path packing, and linear-time FPT algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 462–473. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00051.

11 Bart M. P. Jansen and Ruben F. A. Verhaegh. Search-space reduction via essential vertices revisited: Vertex multicut and cograph deletion. *J. Comput. Syst. Sci.*, 156:103730, 2026. doi:10.1016/J.JCSS.2025.103730.

12 Bart M. P. Jansen and Michal Wlodarczyk. Optimal polynomial-time compression for Boolean max CSP. *ACM Trans. Comput. Theory*, 16(1):4:1–4:20, 2024. doi:10.1145/3624704.

13 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000. doi:10.1137/S0097539799349948.

14 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

15 Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: Complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints. *SIAM J. Comput.*, 54(4):1065–1137, 2025. doi:10.1137/23M1553698.

16 Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized complexity and kernelizability of max ones and exact ones problems. *ACM Trans. Comput. Theory*, 8(1):1:1–1:28, 2016. doi:10.1145/2858787.

17 Stefan Kratsch and Magnus Wahlström. Preprocessing of min ones problems: A dichotomy. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 653–665. Springer, 2010. doi:10.1007/978-3-642-14165-2_55.

18 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

19 Andrei Krokhin and Stanislav Zivny. *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.

20 M. S. Ramanujan and Saket Saurabh. Linear-time parameterized algorithms via skew-symmetric multicuts. *ACM Trans. Algorithms*, 13(4):46:1–46:25, 2017. doi:10.1145/3128600.

21 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

Indexing Range Maximum-Sum Segment Queries with Offsets

Seungbum Jo
Chungnam National University, Daejeon, Republic of Korea

Dominik Köppl
University of Yamanashi, Kofu, Japan

Abstract

Given an array of n real numbers, the maximum segment sum (MSS) problem is to find a contiguous subarray that has the largest sum. While the MSS problem can be solved optimally with Kadane's algorithm in $O(n)$ time, the study of its indexing version spawned new extensions such as (a) retrieving the MSS after subtracting a query offset parameter for all array entries or (b) retrieving the MSS for arbitrary query ranges. We here study the combination of both problems (a) and (b), which requires retrieving the MSS for arbitrary query ranges after subtracting a query offset parameter for all array entries. For that, we present an index whose query time is only slower than the best known for (a) by a factor of $O(\log n)$. In detail, our index uses $O(n \log n)$ space, supports queries in $O(\log^2 n)$ time, and can be constructed in $O(n \log^3 n)$ time. As side results, we study our combined problem in the context of run-length compressed input, and also deduce a solution for (a) that works in run-length compressed space and time. Finally we give supportive lower bounds for our query problem, showing that there is only a polylogarithmic gap of improvement left.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases maximum segment sum, data structure, range query

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.23

Funding *Seungbum Jo*: Seungbum Jo was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-202523963814).
Dominik Köppl: This work was supported by JSPS KAKENHI Grant Number 25K21150.

1 Introduction

Given an array X of real numbers, the maximum-sum segment (MSS) problem is to find an interval $[i, j]$ within X that maximizes the sum of its elements, which we call an MSS of the sequence. This problem found applications, among others, in image processing [11], pattern recognition [19], and biological sequence analysis [26].

The textbook algorithm for solving the MSS problem is Kadane's algorithm, cf. [10, 4.4 Maximum Subarray Sum]. Since the various applications often require adaptations to specific problem variations, it is not surprising that the MSS problem has been studied in various settings, cf. [6] for an encyclopedic survey of MSS and its variations. Chen and Chao [7] considered an indexing problem, where it is allowed to preprocess X and then answer queries on intervals of X. For that task, they devised a linear-time constructible data structure that supports constant-time queries of an MSS for any query interval. They also showed that this problem is linearly equivalent to the range-minimum query (RMQ) problem, meaning that a data structure with construction time $t_c(n)$ and query time $t_q(n)$ solving the MSS problem can be transformed into a data structure solving RMQs with the time complexities $t_c(cn)$ and $t_q(cn)$ for some constant c, respectively. Gawrychowski and Nicholson [12] gave a $\Theta(n)$-bits data structure that also achieves constant query times in the setting that the answer is only the interval and not the maximum sum itself. They also show that $1.89113n - \Theta(\log n)$ bits are necessary for supporting this query type.

© Seungbum Jo and Dominik Köppl;
licensed under Creative Commons License CC-BY 4.0

A maximal local MSS is a local MSS that is not a segment of any local MSS other than it, where a local MSS is a segment that has itself as its only MSS. Ruzzo and Tompa [21] gave an algorithm that finds all distinct maximal local MSSs in $O(n)$ time, and Sakai [22] designed a linear-time constructible data structure supporting constant-time queries of the maximal local MSS of any given segment that contains any given position.

Bengtsson and Chen [3] showed that an arbitrarily given number of non-overlapping segments that maximize the sum of all their elements can be found in linear time. Yu et al. [28] considered the MSS problem where each element of the input sequence is uncertain within a specific interval and proposed a linear-time algorithm for this problem. The density of a segment is defined as the mean of all elements in the segment. Cheng et al. [8] considered the MSS problem with the condition that the density of the segment to be found is between given lower and upper bounds. Bae [1] and Liu and Chao [17] considering returning k non-overlapping segments that maximize the sum of all their elements.

Another variant is the maximum-density segment query [13, 15, 16] that asks to find a segment of length between a given lower and upper bound that maximizes the density. Chung and Lu [9] showed that this query is solvable in linear time.

Another natural extension of the one-dimensional maximum segment sum problem is the maximum subarray problem, where the input is a two-dimensional array with length n in each dimension, and the goal is to find a contiguous rectangular subarray that has the largest sum. This problem has been addressed by Bentley [4], who gave an $O(n^3)$-time algorithm for the problem. Subsequently, the time complexity has been improved to $O(n^3\sqrt{\log\log n/\log n})$ by Takaoka [24] and further to $O(n^3 \log\log n/\log n)$ by Tamaki and Tokuyama [25]. They showed that the problem can be reducible to $(\min,+)$-matrix multiplication. Using this result, the current fastest algorithm runs in $n^3/2^{\Omega(\log n)^{1/2}}$ time by Chan and Williams [5], which also applies to higher-dimensional arrays. In addition, Fukuda et al. [11] introduced the maximum convex sum problem, which is a variant of the maximum subarray problem where the retrieved subarray must be convex (i.e., the intersection of the subarray with any row or column is connected). Here, Bae and Takaoka [2] gave an $O(n)$-time parallel algorithm solving the maximum convex sum problem for n^2 processors.

Our Contribution. Among the aforementioned variations, we focus on two indexing versions of the MSS problem, where we are allowed to preprocess the input array and then answer queries (a) on intervals of the input array, or (b) after subtracting a query offset parameter from all array entries. In particular, we are interested in the combinations of both problems (a) and (b), where we want to answer queries on intervals of the input array after subtracting a query offset parameter from all array entries. We present an indexing data structure solving this combined problem in Theorem 8. The main tool is to answer this problem for the special case that the output interval has to be a prefix or a suffix of the input array, and to reuse known results for the problem (b) to solve the general case on $O(\log n)$ segments. The results are summarized in Table 1. Subsequently, we focus on the special case of run-length compressed input, where we present a data structure in Corollary 15 for the combined problem that uses space linear in the size of the compressed input and answers queries in time polylogarithmic in the size of the compressed input. We can improve the query time for the plain array or in the run-length compressed setting if we allow only integer offsets (Corollary 11 and Remark 16). Finally, we conclude with lower bounds for our combined problem, which show that our data structures are optimal up to polylogarithmic factors (Corollary 19).

Table 1 Summary of best known complexities for the MSS query and its variants.

Query type	Space	Query time	Reference
MSS	$O(1)$	$O(n)$	Kadane's algorithm
OMSS	$O(n)$	$O(\log n)$	Sakai [23]
R-OMSS without offset	$O(n)$	$O(1)$	Chen et al. [7]
Suffix R-OMSS	$O(n)$	$O(\log n)$	Theorem 5
R-OMSS	$O(n \log n)$	$O(\log^2 n)$	Theorem 8

	1	2	3	4	5	6	7	8	9	10	11	12
X_0	4	-9	15	-22	7	-31	18	-5	27	-14	33	-26
C_0	4	-5	10	-12	-5	-36	-18	-23	4	-10	23	-3

Query type	Query Range	Offset	Answer Range	Answer
MSS	-	-	$[7, 11]$	59
SMSS	-	-	$[7, 12]$	33
OMSS	-	10	$[11, 11]$	23
Suffix R-OMSS	$[1, 5]$	1	$[5, 5]$	6
R-OMSS	$[1, 5]$	1	$[3, 3]$	14

Figure 1 Examples for the MSS query and its variants.

2 Preliminaries and Query Definitions

Our computation model is the RAM model. Let $\mathbb{R}$ denote the set of all real numbers. We write $[i, j] = \{i, i+1, i+2, \ldots, j\}$ for two integers $i \leq j$, and use the same notation $[i, j]$ for intervals of real numbers in case that $i, j \in \mathbb{R}$. We assume that all real numbers can be stored in a single word and that basic arithmetic operations on these numbers can be performed in constant time. (We also assume that there are no errors in computational precision due to rounding or overflow.)

Let $X_0[1..n]$ be the input array of real numbers. X_0 is subject to the following queries, for which we present examples in Figure 1. While the queries are initially stated to return the value of a specific subarray of X_0, it is beneficial to also return the interval of the subarray that gives the answer, which we call the *answer range*. Since we here mostly focus on data structures using $\Omega(n)$ space, it suffices to focus on finding answer ranges – we will shortly later see that the actual value can be obtained from an answer range with an $O(n)$-space array of prefix sums. We state our base problem as follows.

Maximum-Segment-Sum (MSS)
Input: array of real numbers $X_0[1..n]$.
Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[1,n]} \sum_{i=\ell}^{r} X_0[i]$.

We also study the special case that the answer range of the MSS problem is constrained to be a suffix of X_0.

Suffix Maximum-Segment-Sum (SMSS)
Input: array of real numbers $X_0[1..n]$
Output: value and integer ℓ achieving $\max_{\ell\in[1,n]} \sum_{i=\ell}^{n} X_0[i]$.

Sakai [23] introduced the following indexing problem that generalizes the MSS problem by allowing a query offset parameter α to be subtracted from all array entries before answering the query.

Offset Maximum-Segment-Sum (OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$.
Query Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[1,n]} \sum_{i=\ell}^{r}(X_0[i] - \alpha)$.

Defining the array $X_\alpha[1..n]$ with $X_\alpha[i] := X_0[i] - \alpha$, the OMSS query on X_0 with offset α is the MSS query on X_α. Hence, OMSS is a generalization of MSS, since the answer to an MSS query on X_0 is the same as the answer to an OMSS query on X_α with offset $\alpha = 0$. Sakai [23] presented an approach answering an OMSS query as follows.

▶ **Lemma 1** ([23]). *Given an array $X_0[1..n]$ of n real numbers, we can compute a list L of at most n (offset, interval)-pairs in $O(n \log^2 n)$ time using $O(n)$ space such that the solution to an OMSS query for a given offset α is the interval of the predecessor offset of α in L. By sorting L descendingly by the offsets, a binary search for the predecessor of α in L can return an answer range of an OMSS query in $O(\log n)$ time.*

The *predecessor* of α in L is the largest offset in L that is smaller than or equal to α. Three observations are in order regarding Lemma 1.

First, while the list of Lemma 1 does not explicitly return the MSS for a query offset α, we can compute the sum $\sum_{i=a}^{b}(X_0[i] - \alpha)$ for an entry $(\alpha, [a..b])$ in L in constant time if we have the array of prefix lengths $C_0[1..n]$ of X_0, where $C_0[i] := \sum_{k=1}^{i} X_0[k]$ for each $i \in [1, n]$. To see that, define the prefix sum of X_α for each $i \in [1, n]$ as $C_\alpha[i] := \sum_{k=1}^{i} X_\alpha[k] = C_0[i] - \alpha k$, and the sum of the subarray $X_\alpha[i..j]$ as $S_\alpha(i, j) = \sum_{k=i}^{j} X_\alpha[k] = C_0[j] - C_0[i-1] - \alpha(j-i+1)$ for any i and j with $1 \leq i \leq j \leq n$. Then $\sum_{i=a}^{b}(X_0[i] - \alpha) = S_\alpha(a, b)$.

Second, the upper bound $|L| \leq n$ is tight since L can reach the size n for the input $X_0 = [1, 2, 3, \ldots, n]$, where the OMSS query for $\alpha = k + \epsilon$ gives the range $[k + 1, n]$ for every $\epsilon \in (0, 1)$ and $k \in [0, n]$.

Finally, there can be multiple answer ranges for a query, in particular if the interval covers trailing zeros (after applying the offset). While implicit, the intervals of Lemma 1 do not contain any zero-sum suffix nor prefix.

Now we present our main problem of interest, which combines the OMSS and MSS queries on subarrays of X_0.

Range Offset Maximum-Segment-Sum (R-OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$ and a range $[a, b] \subseteq [1, n]$.
Query Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[a,b]} \sum_{i=\ell}^{r}(X_0[i] - \alpha)$.

Using the notation of $S_\alpha(i, j)$, the answer to an R-OMSS query on X_0 with query range $[a, b]$ and offset α can be computed as $\max_{[\ell,r]\subseteq[a,b]} S_\alpha(\ell, r)$. The R-OMSS is a generalization of the OMSS problem (with the query range $[1, n]$) and the range MSS problem introduced by Chen et al. [7], which asks for the MSS of a query range $[a, b]$ without the offset parameter (i.e., $\alpha = 0$). Like for MSS and its restriction SMSS, we can also define the suffix and prefix versions of R-OMSS queries, where the answer range is constrained to be a suffix or a prefix of the query range, respectively.

Suffix Range Offset Maximum-Segment-Sum (Suffix R-OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$ and a range $[a, b] \subseteq [1, n]$.
Query Output: value and integer ℓ achieving $\max_{\ell\subseteq[a,b]} \sum_{i=\ell}^{b}(X_0[i] - \alpha)$.

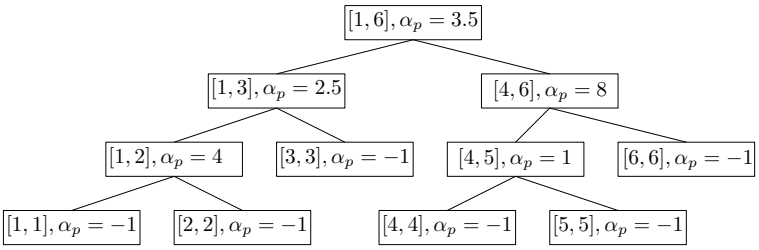


Figure 2 SBST on $X_0 = [4, 1, 6, 1, 8, 0]$. Each node is labeled with its corresponding canonical range and threshold offset.

A symmetric definition can be given for *prefix R-OMSS* queries, where the answer range is constrained to be a prefix of the query range.

3 Data structure for R-OMSS Queries

Our main goal of this paper is to devise data structures for answering R-OMSS queries on X_0, where a non-negative offset $\alpha \geq 0$ and a query range $[a, b]$ are given at query time. There are two naive approaches prioritizing either (1) space or (2) time.

(1) We can apply Kadane's algorithm on $X_\alpha[a..b]$. Since any value in X_α can be computed in $O(1)$ time from X_0, Kadane's algorithm answers R-OMSS in $O(n)$ time using $O(1)$ space additional to X_0.

(2) Alternatively, we can maintain the data structure of Lemma 1 for all $\Theta(n^2)$ possible query ranges. This results in an $O(n^3)$-space data structure with $O(\log n)$ query time.

In what follows, we present two non-trivial data structures for R-OMSS queries that finds its complexities between the solutions (1) and (2). We start with an $O(n)$-space data structure answering suffix R-OMSS queries on X_0 in $O(\log^2 n)$ time. By symmetry, the same data structure results can be obtained for prefix R-OMSS queries. Next, in Section 3.2, we consider a data structure for (general) R-OMSS queries. By combining the data structure of Lemma 1 with the suffix R-OMSS data structures, we obtain an $O(n \log n)$-space data structure that can answer an R-OMSS query in $O(\log^2 n)$ time.

3.1 Data Structure for Suffix R-OMSS Queries

We start with devising a data structure for range suffix OMSS queries. The main idea is to maintain the canonical ranges (defined below) of X_0 using a *suffix binary search tree (SBST)*. Each node in the tree stores a *threshold offset*. By comparing α with the offsets stored along a root-to-leaf path in the tree, as in a standard binary search tree, we can track the endpoint of the range corresponding to the query answer when the query range is canonical. Finally, we partition the query range $[a, b]$ into $O(\log n)$ canonical ranges and answer the query using a dynamic programming approach.

The canonical ranges of X_0 are the elements of a recursive binary partition of $[1, n]$ into $O(n)$ ranges, which can be represented by a binary tree. The *canonical ranges* of X_0 are defined as follows:

(1) $[1, n]$ is a canonical range, and

(2) for every canonical range $[g, h]$ with $g \leq h$, both $[g, g + \lfloor (h - g)/2 \rfloor]$ and $[g + \lfloor (h - g)/2 \rfloor + 1, h]$ are also canonical ranges.

Under this definition, there are $O(n)$ canonical ranges in total. We then define the SBST on $X_0[g, h]$ as follows:

- The root of the SBST on $X_0[g, h]$ is the canonical range $[g, h]$.
- If $g < h$, then the SBSTs on $X_0[g, g + \lfloor (h - g)/2 \rfloor]$ and $X_0[g + \lfloor (h - g)/2 \rfloor + 1, h]$ are the left and right child subtrees of the root, respectively. Otherwise, g=h; a leaf is added with canonical range [g, g].

See Figure 2 for an example. (The reader familiar with segment trees may notice that the SBST is a specialization of a segment tree indexing all integers in $[1, n]$.)

Let T denote the SBST on $X_0[1, n]$. From the definition, each canonical range of X_0 is a node in T, and vice versa. The leaves of T are all canonical ranges of size 1, i.e., $[i, i] = \{i\}$ for each $i \in [1, n]$. By construction, T is a binary tree with $O(n)$ nodes and height $O(\log n)$. Finally, we augment each node p of T with a *threshold offset* α_p. Before defining the threshold offsets, we introduce the following key property for answering SMSS queries.

▶ **Proposition 2.** *For any positive α and β with $\alpha < \beta$, the answer range of the SMSS query on X_β is a subrange of the answer range of the SMSS query on X_α.*

Proof. Let $[x, n]$ and $[x', n]$ be the answer ranges of the SMSS query on X_α and X_β, respectively. Suppose that $x' < x$. Since $S_\beta(x', n) > S_\beta(x, n)$, it follows that the sum $S_\beta(x', x - 1) > 0$, which implies $S_\alpha(x', x - 1) = S_\beta(x', x - 1) + (\beta - \alpha)(x - x') > 0$. This leads to a contradiction since $S_\alpha(x', n) = S_\alpha(x, n) + S_\alpha(x', x - 1) > S_\alpha(x, n)$. ◀

Let $[x_p, y_p]$ be the range corresponding to the node p, and let $[x_\alpha, y_p] \subseteq [x_p, y_p]$ be the answer range of the SMSS problem on $X_\alpha[x_p, y_p]$ for some variable α. We then define the threshold offset α_p at node p as the maximum value of α such that $x_\alpha \in [x_p, x_p + \lfloor (y_p - x_p)/2 \rfloor]$, that is, x_α lies in the range corresponding to the left child of p. If no such α exists, or if p is a leaf node, we define $\alpha_p = -1$. By Proposition 2, the value α_p is well defined as a single constant.

The data structure for suffix R-OMSS queries is composed of (1) the array C_0, and (2) the SBST T on X_0. Each node of the SBST stores its threshold offset and its canonical range. (1) and (2) together use $O(n)$ space in total.

We now describe how to answer a query using this data structure. We first consider the case where the query range is a canonical range $[x_p, y_p]$ corresponding to a node p in T. In this case, it suffices to find the position x such that $[x, y_p]$ is the answer range. To find x, we start from the node p and compare α with α_p. As in a standard binary search tree, we recursively perform the same comparison: we proceed to the left child if $\alpha \leq \alpha_p$, and to the right child otherwise, until we reach a leaf node corresponding to the canonical range $[x', x']$. Then $x' = x$ from the following proposition.

▶ **Proposition 3.** *Let p be an internal node in T with the corresponding range $[x_p, y_p]$, and let $z_p = x_p + \lfloor (y_p - x_p)/2 \rfloor$. If $\alpha \leq \alpha_p$ (resp. $\alpha > \alpha_p$), then the leftmost endpoints of the answer ranges of the SMSS queries on $X_\alpha[x_p, y_p]$ and $X_\alpha[x_p, z_p]$ (resp. on $X_\alpha[x_p, y_p]$ and $X_\alpha[z_p + 1, y_p]$) are the same.*

Proof. Here we consider only the case that $\alpha \leq \alpha_p$; the case $\alpha > \alpha_p$ can be proved analogously. Let x and x' be the leftmost endpoints of the answer ranges of the SMSS queries on $X_\alpha[x_p, y_p]$ and $X_\alpha[x_p, z_p]$. We now claim that $x = x'$. To prove the claim, suppose that $x < x'$ (the case $x > x'$ can be handled analogously). Since $S_\alpha(x, y_p) > S_\alpha(x', y_p)$, it follows that $S_\alpha(x, x' - 1) > 0$. Consequently, $S_\alpha(x, z_p) = S_\alpha(x', z_p) + S_\alpha(x, x' - 1) > S_\alpha(x', z_p)$, which contradicts the definition of x'. ◀

Since the height of T is $O(\log n)$, we can answer a suffix R-OMSS query on X_0 in $O(\log n)$ time when the query range is canonical.

Finally, we consider how to answer a suffix R-OMSS query on X_0 for a general query range $[a, b]$ using the data structure. We first partition $[a, b]$ into $c = O(\log n)$ distinct canonical ranges $[a_1, b_1], \ldots, [a_c, b_c]$, ordered from left to right (i.e., $a_1 = a$, $b_c = b$, and $b_k + 1 = a_{k+1}$ for all $k \in [c-1]$). This partition can be computed by starting from the root of T and traversing the tree downwards in two directions for finding the leaves of the canonical ranges $[a, a]$ and $[b, b]$, respectively. We then collect $O(\log n)$ canonical ranges corresponding to children on the two root-to-leaf paths. At each level of T, there are at most two nodes whose canonical ranges $[x, y]$ satisfy $[x, y] \cap [a, b] \neq \emptyset$ but $[x, y] \not\subseteq [a, b]$; only these nodes require recursion to the next level, while any other intersecting node is fully contained in $[a, b]$ and can be reported as a canonical range immediately. Therefore, we can compute these canonical ranges in $O(\log n)$ time.

If $c = 1$ (i.e., $[a, b]$ is a canonical range), we can answer the R-OMSS query in $O(\log n)$ time using T. Otherwise, we apply a dynamic programming approach as follows. Define an array $A[1..c]$ of size c, where for each $k \in [1, c]$, $A[k]$ stores the answer range of the SMSS query on $X_\alpha[a_1, b_k]$. Then $A[c]$ stores the answer of the suffix R-OMSS query.

To compute $A[c]$, we first handle the base case by computing $A[1]$ in $O(\log n)$ time using T. Next, for $k > 1$, we compute $A[k]$ from $A[k-1]$ as follows. To compute $A[k]$, it suffices to determine the answer range $[x, b_k]$ of the SMSS query on $X_\alpha[a_1, b_k]$. Now we introduce the following lemma, which is used to compute $A[k]$.

▶ **Lemma 4.** *Let $[x, b_k]$ and $[x', b_{k-1}]$ be the answer ranges of the SMSS queries on $X_\alpha[a_1, b_k]$ and $X_\alpha[a_1, b_{k-1}]$, respectively. Then x is either (1) x' or (2) z, where $[z, b_k]$ is the answer range of the SMSS query on $X_\alpha[a_k, b_k]$.*

Proof. If $x < x'$ or $x' < x \leq b_{k-1}$, then $S_\alpha(x, b_k) = S_\alpha(x, b_{k-1}) + S_\alpha(a_k, b_k) > S_\alpha(x', b_{k-1}) + S_\alpha(a_k, b_k)$, which contradicts the fact that $[x', b_{k-1}]$ is the answer range of $X_\alpha[a_1, b_{k-1}]$.

Similarly, if $a_k \leq x < z$ or $x > z$, then $S_\alpha(x, b_k) > S_\alpha(z, b_k)$, which contradicts the fact that $[z, b_k]$ is the answer range of $X_\alpha[a_k, b_k]$. ◀

The answer ranges of the SMSS queries on $X_\alpha[a_1, b_{k-1}]$ and $X_\alpha[a_k, b_k]$ can be obtained from $A[k-1]$ and from T, respectively, in $O(\log n)$ time. Thus, by Lemma 4, given $A[k-1]$, the value $A[k]$ can be computed in $O(\log n)$ time, which implies that $A[c]$ can be computed in $O(\log^2 n)$ time.

Construction time. As C_0 can be constructed in $O(n)$ time, it suffices to analyze the construction time of T. To construct T, for each node p whose corresponding range is $[x_p, y_p]$ with $\ell_p = y_p - x_p + 1$, we compute its threshold offset α_p as follows. Consider the point set $P = \{(t, S_0(y_p - t + 1, y_p)) \mid 1 \leq t \leq \ell_p\}$, that is, each point corresponds to a suffix of $[x_p, y_p]$, where the x-coordinate t is the suffix length and the y-coordinate is its sum. Among these points, we keep only those that can be optimal for some $\alpha > 0$. Equivalently, we keep the vertices of the upper convex hull of P (since we maximize $y - \alpha x$). The convex hull can be constructed in $O(\ell_p)$ time, since the points in P are already sorted by their first coordinates [14].

For any suffix represented by a point $(x, y) \in P$, we have $S_\alpha(y_p - x + 1, y_p) = y - \alpha x$. Thus, given an offset α, answering the SMSS query on $X_0[x_p, y_p]$ reduces to finding the point $(x, y) \in P$ that maximizes $y - \alpha x$. By point–line duality, each point (x, y) corresponds to the line $L_{(x,y)}(w) = y - wx$, and $\max_{(x,y)\in P}(y - \alpha x)$ equals the value of the upper envelope at $w = \alpha$, which can be computed in $O(\ell_p)$ time from the convex hull of P. The upper

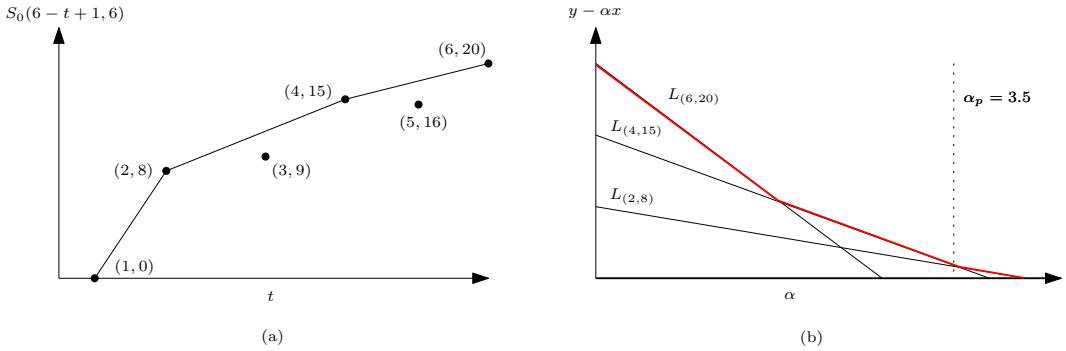


Figure 3 Example illustrating the computation of the threshold offset at the root of T in Figure 2. (a) The point set P with its convex hull. (b) The dual lines corresponding to the convex hull of P. The red segments represent the upper envelope of these lines.

envelope is piecewise linear and changes only at intersection points of consecutive envelope lines. Therefore, α_p is the breakpoint on the envelope where the optimal suffix switches from a suffix whose starting position lies in the left half of $[x_p, y_p]$ to one whose starting position lies in the right half. Since the envelope breakpoints are sorted by slope, α_p can be found by binary search over the hull in $O(\log \ell_p)$ time (using Proposition 2). See Figure 3 for an example.

Thus, the nodes at each level of T can be constructed in $O(n)$ time, implying that the entire tree T can be constructed in $O(n \log n)$ time overall.

We summarize the results of this section in the following theorem.

▶ **Theorem 5.** *Given an array X_0 of size n, there exists an $O(n)$-space data structure that can answer a prefix/suffix R-OMSS query on X_0 in $O(\log^2 n)$ time for any positive offset α. When the query range is a canonical range of X_0, the data structure can answer a prefix/suffix R-OMSS query in $O(\log n)$ time. The data structure can be constructed in $O(n \log n)$ time.*

3.2 Data Structure for General R-OMSS Queries

Building on Section 3.1, we now generalize our findings and devise a data structure for answering (general) R-OMSS queries on X_0 for any positive offset $\alpha > 0$ with the query range $[a, b]$. The data structure is composed of the following three parts:

(1) the array C_0,

(2) the data structures of Theorem 5 on X_0, which can answer a prefix and suffix R-OMSS query on X_0 in $O(\log n)$ time when the query range is canonical, and

(3) an $O(|h-g|)$-space data structure of Lemma 1 constructed for each canonical range $[g, h]$, which enables answering the OMSS query on $X_0[g, h]$ in $O(\log n)$ time.

Since the total length of all canonical ranges of X_0 is $O(n \log n)$, (3) uses $O(n \log n)$ space in total. As both (1) and (2) use $O(n)$ space, the overall data structure requires $O(n \log n)$ space.

To answer an R-OMSS query on X_0 with query range $[a, b]$, we first partition $[a, b]$ into $c = O(\log n)$ distinct canonical ranges $[a_1, b_1], \ldots, [a_c, b_c]$, ordered from left to right, in $O(\log n)$ time. If $c = 1$, we answer the query directly in $O(\log n)$ time using the corresponding OMSS data structure of Lemma 1. Otherwise, we apply a dynamic programming approach similar to that in Section 3.1, as follows. We define an array $A[1..c]$, where for each $k \in [1, c]$, $A[k]$ stores the answer range of the R-OMSS query on X_0 with query range $[a_1, b_k]$, under the condition that the rightmost endpoint of the answer range is in $[a_k, b_k]$.

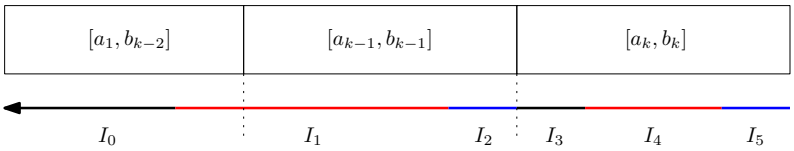


Figure 4 Subranges that partition the range $[a_1, b_k]$.

Then the answer to the R-OMSS query on X_0 with range $[a, b]$ is the maximum value in A, since the rightmost endpoint of the answer range may lie in any of the canonical ranges.

The base case $k = 1$ can be computed in $O(\log n)$ time using the corresponding OMSS data structure of Lemma 1. For $k > 1$, we compute $A[k]$ from $A[k-1]$, we partition the range $[a_1, b_k]$ into five subranges as follows (see Figure 4 for a schematic sketch):

- I_0: The range $[a_1, b_{k-1}] \setminus A[k-1]$
- I_1: The answer range of the R-OMSS query on X_0 with query range $[a_1, b_{k-1}]$ under the condition that the rightmost endpoint of the answer range is in $[a_{k-1}, b_{k-1}]$, i.e., the answer range corresponding to $A[k-1]$.
- I_2: The range $[a_{k-1}, b_{k-1}] \setminus (I_0 \cup I_1)$.
- I_3: The maximal prefix of $X_0[a_k, b_k]$ that does not intersect with I_4.
- I_4: The answer range of the R-OMSS query on X_0 with query range $[a_k, b_k]$.
- I_5: The range $[a_k, b_k] \setminus (I_3 \cup I_4)$.

I_1 (and hence I_0 and I_2) can be obtained from $A[k-1]$ and I_4 (and hence I_3 and I_5) can be computed in $O(\log n)$ time using the data structure of Lemma 1 on the canonical range $[a_k, b_k]$. Thus, all the subranges above can be identified in $O(\log n)$ time. Now we introduce the following lemma, which we use to compute $A[k]$.

▶ **Lemma 6.** *Let I be the answer range of the R-OMSS query on X_0 with query range $[a_1, b_k]$, under the condition that the rightmost endpoint of the answer range falls within $[a_k, b_k]$. Then I must be one of the following five ranges:*

1. I_4,
2. $I_1 \cup I_2 \cup I'_3$,
3. $I_1 \cup I_2 \cup I_3 \cup I_4$,
4. $I'_2 \cup I'_3$, *or*
5. $I'_2 \cup I_3 \cup I_4$.

Here, I'_2 denotes the answer range of the suffix R-OMSS query on X_0 with offset α and query range $[a_{k-1}, b_{k-1}]$, and I'_3 denotes the answer range of the prefix R-OMSS query on X_0 with offset α and query range $[a_k, b_k]$.

Proof. Observe that the sum of elements in X_α over any suffix of I_0 is negative; otherwise, I_1 would include such a suffix of I_0. Similarly, the sum of elements in X_α over any prefix of I_5 is negative. Hence, I does not contain any position from either I_0 or I_5.

Also, if $I'_2 \cap I_1 \neq \emptyset$, then $I_1 \cup I_2$ is the answer range of the SMSS problem on $X_\alpha[a_1, b_{k-1}]$. Otherwise, suppose that $I' \neq I_1 \cup I_2$ is the answer range of the SMSS problem on $X_\alpha[a_1, b_{k-1}]$. By the definition of I'_2, we have $I_2 \subseteq I'_2 \subseteq I'$. Then the sum of X_α over the range $I' \setminus I_2$, which is neither I_1 nor empty (since $I'_2 \cap I_1 \neq \emptyset$), is greater than the sum of X_α over I_1. This contradicts the definition of I_1 as the answer range on $[a_1, b_{k-1}]$. Therefore, when I'_2 contains positions in $[a_1, b_{k-1}]$, I'_2 must be either $I_1 \cup I_2$ or satisfy $I'_2 \subseteq I_2$. By a similar argument for I'_3, we obtain that $I'_3 = I_3 \cup I_4$ or $I'_3 \subseteq I_3$.

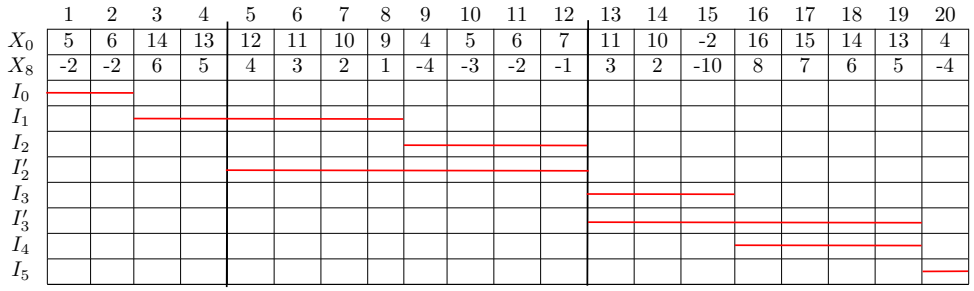


Figure 5 Supplement figure for Example 7. Canonical ranges are represented by thick vertical bars, and the intervals I_j by red horizontal lines in their respective rows.

From the above observations, we consider the following three cases:

(A) $I \subseteq [a_k, b_k]$,

(B) $I \cap I_1 \neq \emptyset$, and

(C) $I \cap I_1 = \emptyset$ but $I \cap I_2 \neq \emptyset$.

For Case (A), we have $I = I_4$ by the definition of I_4. For Case (B), I_1 must be completely contained in I by the observation above. Thus, in this case, I is either $I_1 \cup I_2 \cup I'_3$ or $I_1 \cup I_2 \cup I_3 \cup I_4$ (the latter corresponds to the case where $I'_3 = I_3 \cup I_4$). For Case (C), since $I'_2 \subseteq I_2$ by the observation above, I is either $I'_2 \cup I'_3$ or $I'_2 \cup I_3 \cup I_4$ (again, the latter corresponds to the case where $I'_3 = I_3 \cup I_4$). ◀

▶ **Example 7.** Figure 5 illustrates how Lemma 6 operates. In this example, the range $[1, 20]$ is partitioned into three canonical ranges $[1, 4]$, $[5, 12]$, and $[13, 20]$ (here the notion of canonical ranges is defined differently from that in the main text for convenience).

To answer the query with offset $\alpha = 8$ on the query range $[1, 20]$, consider the step of computing $A[3]$ from $A[2]$. In this case, both $I'_2 \cap I_1$ and $I'_3 \cap I_4$ are non-empty. Therefore, $A[3]$ is the maximum between the sum of X_8 over I_4 and the sum over $I_1 \cup I_2 \cup I_3 \cup I_4$, which are 26 and 32, respectively. Hence, $A[3] = 32$.

Query time. I'_2 can be computed in $O(\log n)$ time using the suffix R-OMSS data structure on X_0 on the canonical query range $[a_{k-1}, b_{k-1}]$ with I_1. Similarly, I'_3 can be computed in $O(\log n)$ time using the prefix R-OMSS data structure on X_0 on the canonical query range $[a_k, b_k]$. Therefore, by Lemma 6, we can compute $A[k]$ from $A[k-1]$ in $O(\log n)$ time, which implies that all values in A can be computed in $O(\log^2 n)$ time. Finally, we can find the maximum value in A in $c = O(\log n)$ time by a linear scan.

Construction time. The construction time of the data structure in Lemma 1 is $O(\ell \log^2 \ell)$, where ℓ denotes the input size. Since the total length of all canonical ranges of X_0 is $O(n \log n)$, this yields a total construction time of $O(n \log^3 n)$ [23]. In addition, constructing the data structure of Theorem 5 requires $O(n \log n)$ time. Hence, the overall construction time of the data structure is $O(n \log^3 n)$.

We summarize the results of this section in the following theorem.

▶ **Theorem 8.** *Given an array X_0 of size n, there exists an $O(n \log n)$-space data structure that can answer an R-OMSS query on X_0 in $O(\log^2 n)$ time for any positive offset α. The construction time of the data structure is $O(n \log^3 n)$.*

4 Special Types of Inputs

Based on the results of the previous section, we analyze and devise specializations of our data structures for R-OMSS queries on X_0 under special input types. Specifically, we consider the following three scenarios: (1) the offset α is restricted to positive integers, (2) X_0 is an array over an alphabet of size σ, and (3) X_0 is given in run-length encoded (RLE) form.

Integer Offsets. When the offset α is restricted to positive integers, the data structure of Theorem 8 can be improved by the following modifications. First, we introduce a lemma that states the data structure of Lemma 1 can be improved under this restriction.

▶ **Lemma 9.** *When the offset α is restricted to positive integers, there exists an $O(n)$-space data structure that can answer an OMSS query on X_0 in $O(\log\log n)$ time.*

Proof. We modify the data structure of Lemma 1 as follows. For each (offset, interval) pair (α, I) in L (the list defined in the statement of Lemma 1), we replace it with the integer pair $(\lceil\alpha\rceil, I)$. If there exist two distinct pairs (α_1, I_1) and (α_2, I_2) in L such that $\alpha_1 < \alpha_2$ and $\lceil\alpha_1\rceil = \lceil\alpha_2\rceil$, we discard the pair (α_1, I_1) and maintain only $(\lceil\alpha_2\rceil, I_2)$ in L. We then maintain L using a y-fast trie [27], which supports predecessor searches on L in $O(\log\log n)$ time using $O(n)$ space. ◀

Next, for each canonical range $[g, h]$ of X_0, let $T_{[g,h]}$ be the rooted subtree of the SBST T on X_0 (defined in Section 3.1) whose root corresponds to $[g, h]$. We define a list $L_{[g,h]}$ of (offset, interval) pairs as follows: $(\beta, I) \in L_{[g,h]}$ if and only if β is a threshold offset stored in the node of $T_{[g,h]}$, and I is the answer range of the suffix R-OMSS query on X_0 with query range $[g, h]$ and offset β. The list $L_{[g,h]}$ has size $O(|h - g|)$, since the number of distinct offsets equals the number of nodes in $T_{[g,h]}$. Moreover, by the algorithm for answering suffix R-OMSS queries on X_0 using T, for any offset α, the answer range of the suffix R-OMSS query on X_0 with the query range $[g, h]$ is exactly the interval I such that $(\beta, I) \in L_{[g,h]}$, where β is the predecessor of α in $L_{[g,h]}$.

Therefore, $L_{[g,h]}$ plays the same role as the data structure of Lemma 1 for the suffix R-OMSS query on X_0 when the query range is restricted to $[g, h]$. By maintaining such lists for all canonical ranges of X_0, together with the modifications used in the proof of Lemma 9, we obtain the following lemma, since the total length of all canonical ranges of X_0 is $O(n \log n)$.

▶ **Lemma 10.** *When the offset α is restricted to positive integers and the query range is restricted to a canonical range of X_0, there exists an $O(n \log n)$-space data structure that can answer an suffix R-OMSS query on X_0 in $O(\log\log n)$ time.*

Finally, by combining the data structures of Lemma 9 and Lemma 10, together with C, we obtain the following corollary.

▶ **Corollary 11.** *When the offset α is restricted to positive integers, there exists an $O(n \log n)$-space data structure that can answer an R-OMSS query on X_0 in $O(\log n \log\log n)$ time.*

Bounded Alphabet Size. When X_0 is an array over an alphabet $\Sigma \subset \mathbb{R}$ of size σ, we show that the data structure in Lemma 1 requires $\Omega(\sqrt{n})$ space, even when X_0 is a binary array. Recall that this data structure partitions the domain of offsets into the ranges $[0, \alpha_1)$, $[\alpha_1, \alpha_2)$, $[\alpha_2, \alpha_3), \ldots, [\alpha_{k-1}, \alpha_k), [\alpha_k, \infty)$ and assigns each such range $[\alpha_i, \alpha_{i+1})$ an answer range $[a_i, b_i]$ such that $[a_i, b_i]$ is the answer range for each offset $\alpha \in [\alpha_i, \alpha_{i+1})$ (defining

the border cases $\alpha_0 = 0$ and $\alpha_{k+1} = \infty$). Therefore, the question arises how large k must be. We say that two answer ranges $[a, b]$ and $[a', b']$ for the OMSS queries on X_0 with two different offsets α and α', respectively, are *non-compatible* if $S_\alpha(a, b) > S_\alpha(a', b')$ or $S_{\alpha'}(a, b) < S_{\alpha'}(a, b)$. Distinct answer ranges are compatible if and only if their respective subarrays in X_0 are permutations of each other. For the analysis, we say that two offsets α and α' are *non-compatible* if the offsets α and α' have no answer range in common, i.e., there is no answer range of α that is compatible with any answer range of α' (and vice versa).

▶ **Theorem 12.** *There exists a binary array X_0 of size n with $\Theta(\sqrt{n})$ non-compatible offsets.*

Proof. Consider the input $X_0 = 110^1 10^2 10^3 10^4 \cdots 0^k$ for $k = \Theta(\sqrt{n})$. Then, for each $k' \in [k-1]$, when the offset α lies in the interval $(1/(k'+1), 1/k')$, the answer range of the OMSS query on X_0 is the minimal prefix range that covers $1 + \lfloor 1/k' \rfloor$ ones. Since for each such k' there is exactly one answer range, and all these answer ranges are non-compatible, the claim follows. ◀

For the bounded-alphabet setting, improving the $\Omega(\sqrt{n})$ space lower bound for the data structure of Lemma 1, or designing an indexing data structure for OMSS queries that uses $o(n)$ space, remains open. In Section 5, we will discuss the space usage of the data structure of Theorem 5 and Theorem 8 in the bounded-alphabet case.

Run-Length Encoded (RLE) Input. If the input array $X_0 = X_0[j_1]^{k_1} X_0[j_2]^{k_2} \cdots X_0[j_z]^{k_z}$ for $j_1 = 1 < j_2 < \cdots < j_z = n$ and $k_1, \ldots, k_z \geq 1$ of size n has z distinct runs, we can reduce the space usage of the data structures in Lemma 1, Theorem 5, and Theorem 8 to $O(z)$ space by indexing the RLE form of X_0. The *RLE form* of X_0 is a compact representation of X_0 that encodes each run $X_0[j_i]^{k_i}$ of X_0 by a pair of value $X_0[j_i]$ and the length of the run k_i. Hence, X_0 can be represented in $O(z)$ space with this RLE form.

We make use of the RLE form of X_0 as follows. As common components of our data structure, we maintain the following arrays:

1. $X' := [X_0[j_1], X_0[j_2], \ldots, X_0[j_z]]$ (i.e., X_0 in RLE form),
2. D of size z, where $D[i] = \sum_{t=1}^{i} k_t$, and
3. C' of size z, where $C'[i] = \sum_{t=1}^{i} k_t \cdot X'[t]$ for all $i \in [1, z]$.

These arrays use $O(z)$ space in total. We stipulate that $D[0] = 0$.

Given a position $i \in [1, n]$ of X_0 with $i \in [D[i'-1]+1, D[i']]$ for some $i' \in [1, z]$, i.e., i is in the i'-th run X_0, we can compute i' from i in $O(\log z)$ time using a binary search on D. Then, we have $X_0[i] = X'[i']$ and $C_0[i] = C'[i'-1] + (i - D[i'-1]) \cdot X'[i']$. Conversely, given a position $i' \in [1, z]$ of X', i' corresponds to the i'-th run that starts at position $D[i'-1]+1$ and ends at position $D[i']$ in X_0. We now present a lemma showing that the data structure of Lemma 1 can be improved under this representation.

▶ **Lemma 13.** *There exists a data structure of size $O(z)$ that supports OMSS queries on X_0 in $O(\log z)$ time.*

Proof. We construct the data structure of Lemma 1 on X' using $O(z)$ space, which supports OMSS queries on X' in $O(\log z)$ time. To answer an OMSS query on X with offset α, we first compute the OMSS on X' with the same offset in $O(\log z)$ time. Let $[i', j']$ be the answer range of the OMSS on X'. We claim that $[i, j]$ is the answer range on X, where i (resp. j) is the leftmost (resp. rightmost) position in X corresponding to i' (resp. j').

To prove the claim, it suffices to show that the endpoints of the answer range on X_0 must lie on run boundaries. Assume the contrary, i.e., one side j of the answer range partially covers a run. By definition, this run covering position $X_0[j]$ corresponds to a positive value

in $X_\alpha[j]$, otherwise the range could be shortened to exclude all values of this run and thus increase the sum. However, since $X_\alpha[j]$ is positive, extending the range to include the entire run further increases the sum, and thus contradicts the assumption.

Since the claim holds and the corresponding positions i and j can be computed in $O(\log z)$ time, we can answer an OMSS query on X_0 in $O(\log z)$ time overall. ◀

From the claim used in the proof of Lemma 13, we can also obtain data structures in Theorem 5 and Theorem 8 parameterized by z, as summarized in the following corollaries.

▶ **Corollary 14.** *There exists a data structure of size $O(z)$ that supports suffix R-OMSS queries on X_0 in $O(\log^2 z)$ time.*

Proof. We construct the data structure of Theorem 5 on X' using $O(z)$ space, which supports OMSS queries on X' in $O(\log^2 z)$ time. To answer a suffix R-OMSS query on X_0 with query range $[a, b]$ and offset α, we first compute the OMSS on X' with the same offset and query range $[a', b']$ in $O(\log^2 z)$ time, where a' and b' are the positions in X' corresponding to the runs of X_0 containing the positions a and b, respectively.

Let $[i', j']$ be the answer range of the OMSS on X', and i and j be the leftmost and the rightmost positions in X_0 corresponding to i' and j', respectively. Then, by the claim in the proof of Lemma 13, the answer range of the query on X_0 is $[\max(a, i), b]$. ◀

▶ **Corollary 15.** *There exists a data structure of size $O(z \lg z)$ that supports R-OMSS queries on X_0 in $O(\log^2 z)$ time.*

Proof. The proof is analogous to that of Corollary 14, exchanging the data structure of Theorem 5 with that of Theorem 8. The only difference is that we return $[\max(a, i), \min(b, j)]$ as the answer range, where j is the rightmost position in X_0 corresponding to j' (all notations are the same as in the proof of Corollary 14). ◀

▶ Remark 16. As all values in D are integers, we can locate any position in X and its corresponding position in X' (and vice versa) in $O(\log \log z)$ time by maintaining a y-fast trie on D. Consequently, by applying the results of Lemma 9, Lemma 10, and Corollary 11, the query times of the data structures in Lemma 13, Corollary 14, and Corollary 15 can be improved to $O(\log \log z)$, $O(\log z \log \log z)$, and $O(\log z \log \log z)$ time, respectively, when the offset α is restricted to positive integers.

5 Lower Bounds

Finally, we present lower bound results for data structures that answer R-OMSS queries on X_0. We first show a reduction from the predecessor query to the OMSS query, and subsequently show that any encoding data structure that answers suffix R-OMSS queries requires at least $\Omega(n \log n)$ bits.

Time-Space Trade-Offs Lower Bound. In the following theorem, we show there exists a reduction from the predecessor query to OMSS query.

▶ **Theorem 17.** *For any set S of n values from a universe $U \subset \mathbb{R}$, given a value $\alpha \in U$, the predecessor of α in S can be determined by a OMSS query and an access on an array of size n.*

Proof. Suppose we want to answer a predecessor query for a value $\alpha \in U$ in S. For that, we have constructed, in a precomputation step, an OMSS data structure on X_0, where X_0 stores the values of S in ascendingly sorted order, and compute the OMSS query with offset α. If the answer range for α is $[n, n]$, then the predecessor of α is either $X_0[n]$ (if $X_0[n] \leq \alpha$) or $X_0[n-1]$ (if $X_0[n] > \alpha$). If the answer range is $[1, n]$, then there is no predecessor of α in S. Otherwise, the answer range is $[i, n]$ with $i \in [2, n-1]$. Then the predecessor of α is $X_0[i-1]$ since (a) $X_\alpha[k] < 0$ for all $k \leq i-2$, (b) $X_\alpha[i-1] \leq 0$, and (c) $X_\alpha[k] > 0$ for all $k \geq i$. We here assume that the answer range of the OMSS query is the shortest one (omitting trailing zero entries), which can be ensured by a tie-breaking rule in the definition of the OMSS query. ◀

From the predecessor lower bound of Pătraşcu and Thorup [18], the data structure of Lemma 1 achieves an optimal tradeoff between space and query time. Moreover, the data structures of Theorem 5 and Theorem 8 also provide optimal space–query time tradeoffs up to a $\log n$ factor in the query time, since they can answer OMSS queries.

Space Lower Bounds in the Encoding Model. One may also study the R-OMSS query in the variant where only the answer range in X_0 is returned. In this setting, the problem is considered under the *encoding model*: after preprocessing X_0, we maintain only the information necessary to answer queries, without storing X_0 explicitly. For many range queries, encoding data structures use less space than explicitly storing X_0 (see [20] for details). For example, there exists an $O(n)$-bit data structure for answering range MSS queries under a word-RAM model with a word size of $\Theta(\log n)$ bits [12], which uses asymptotically less space than the $\Omega(n \log n)$ bits required to store X_0 under the same model. We now prove the following theorem, which shows that for the suffix R-OMSS queries (and hence for R-OMSS queries), any encoding data structure must use asymptotically as much space as storing the input. This implies that the data structure of Theorem 5 uses asymptotically optimal space.

▶ **Theorem 18.** *Any permutation X_0 of $[1, n]$ can be reconstructed from an encoding that supports suffix R-OMSS queries. Consequently, any encoding data structure that answers suffix R-OMSS queries requires at least $\Omega(n \log n)$ bits.*

Proof. We first identify the position of 1 as follows. Let $\alpha_1 = 0$ and $\alpha_2 = 1 + \epsilon$, where $0 < \epsilon < 1$ is a fixed constant. For each $a \in [1, n-1]$, we perform R-OMSS queries on the range $[a, a+1]$ with offsets α_1 and α_2. From the definition of the offsets, $X_0[i] = 1$ if and only if the answer ranges for the query range $[i, i+1]$ are $[i, i+1]$ under α_1 and $[i+1, i+1]$ under α_2. Since X_0 is a permutation, there exists exactly one such position i unless $X_0[n] = 1$. To check whether $X_0[n] = 1$, we perform two queries with query range $[n-1, n]$: one query with offset α_1 and expected answer range $[n-1, n]$, and another query with offset α_2 and expected answer range $[n-1, n-1]$.

After locating the position i of 1, we split the permutation into the two subarrays $X_0[1, i-1]$ and $X_0[i+1, n]$. We then find the position of 2 in the same manner, which must lie in exactly one of these two subarrays. Repeating this procedure recursively determines the position of every value up to n, and hence reconstructs X_0. ◀

As a corollary of Theorem 18 we can also obtain the space lower bound for suffix R-OMSS queries (and hence for R-OMSS queries) in the encoding model with bounded alphabet size.

▶ **Corollary 19.** *Any encoding data structure that supports suffix R-OMSS queries on an array of size n over an alphabet of size σ requires at least $\Omega(n \log \sigma)$ bits.*

Proof. Let $\mathcal{X}$ be the set of all arrays of size n obtained by concatenating n/σ permutations of $[1, \sigma]$ (we assume that n is a multiple of σ, which does not affect the asymptotic lower bound). Then $|\mathcal{X}| = (\sigma!)^{n/\sigma}$, and hence $\log|\mathcal{X}| = \Omega(n \log \sigma)$. Moreover, any array in $\mathcal{X}$ can be reconstructed from suffix R-OMSS queries, since each permutation block can be reconstructed by Theorem 18. This proves the corollary. ◀

Corollary 19 implies that even in the bounded-alphabet setting, any encoding data structure for suffix R-OMSS queries (and hence for R-OMSS queries) must use asymptotically as much space as required to store the input.

6 Open Problems

There are many gaps left open regarding time and space complexities. It is puzzling to us whether we can construct the data structure of Lemma 1 in optimal $O(n)$ time. This would improve the construction time of our R-OMSS data structures in Theorem 8. Here, we additionally want to improve the space to $O(n)$, which would be optimal by Theorem 18. Improving the query time to $O(\log n)$ is also an interesting open problem, which would be optimal by Theorem 17 for general offsets. In case that the number of distinct values in X_0 is bounded, we wonder whether we can improve lower and upper bounds on the number of non-compatible offsets.

For future directions, we want to study the variations of the MSS problem described in the introduction as query problems with offset and/or query range parameters.

References

1 Sung Eun Bae. *Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem*. PhD thesis, University of Canterbury, 2007.

2 Sung Eun Bae, Tong-Wook Shinn, and Tadao Takaoka. Efficient algorithms for the maximum sum problems. *Algorithms*, 10(1), 2017. doi:10.3390/a10010005.

3 Fredrik Bengtsson and Jingsen Chen. Computing Maximum-Scoring segments optimally. Research Report 2007:03, Luleå University of Technology, Department of Computer Science and Electrical Engineering, 2007.

4 Jon Louis Bentley. Perspective on performance. *Commun. ACM*, 27(11):1087–1092, 1984. doi:10.1145/1968.381154.

5 Timothy M. Chan and R. Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

6 Kun-Mao Chao. Maximum-sum segments. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1242–1244. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_226.

7 Kuan-Yu Chen and Kun-Mao Chao. On the range maximum-sum segment query problem. *Discrete Applied Mathematics*, 155(16):2043–2052, 2007. doi:10.1016/J.DAM.2007.05.018.

8 Chih-Huai Cheng, Hsiao-Fei Liu, and Kun-Mao Chao. Optimal algorithms for the average-constrained maximum-sum segment problem. *Inf. Process. Lett.*, 109(3):171–174, 2009. doi:10.1016/J.IPL.2008.09.024.

9 Kai-Min Chung and Hsueh-I Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2004. doi:10.1137/S0097539704440430.

10 Christoph Dürr and Jill-Jênn Vie. *Competitive Programming in Python: 128 Algorithms to Develop your Coding Skills*. Cambridge University Press, 2020. URL: https://books.google.de/books?id=JA0FEAAAQBAJ.

11 Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.*, 26(2):179–213, 2001. doi:10.1145/383891.383893.

12 Paweł Gawrychowski and Patrick K Nicholson. Encodings of range maximum-sum segment queries and applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 196–206. Springer, 2015.

13 Michael H. Goldwasser, Ming-Yang Kao, and Hsueh-I Lu. Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In *Proc. WABI*, volume 2452 of *LNCS*, pages 157–171, 2002. doi:10.1007/3-540-45784-4_12.

14 Ronald L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

15 Sung Kwon Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *Information Processing Letters*, 86(6):339–342, 2003. doi:10.1016/S0020-0190(03)00225-4.

16 Yaw-Ling Lin, Tao Jiang, and Kun-Mao Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.*, 65(3):570–586, 2002. doi:10.1016/S0022-0000(02)00010-7.

17 Hsiao-Fei Liu and Kun-Mao Chao. Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence. *Theor. Comput. Sci.*, 407(1-3):349–358, 2008. doi:10.1016/J.TCS.2008.06.052.

18 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 232–240, 2006.

19 Kalyan S. Perumalla and Narsingh Deo. Parallel algorithms for maximum subsequence and maximum subarray. *Parallel Process. Lett.*, 5:367–373, 1995. doi:10.1142/S0129626495000345.

20 Rajeev Raman. Encoding data structures. In *International Workshop on Algorithms and Computation*, pages 1–7. Springer, 2015. doi:10.1007/978-3-319-15612-5_1.

21 Walter L. Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proc. ISMB*, pages 234–241, 1999. URL: http://www.aaai.org/Library/ISMB/1999/ismb99-027.php.

22 Yoshifumi Sakai. A maximal local maximum-sum segment data structure. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101-A(9):1541–1542, 2018. doi:10.1587/TRANSFUN.E101.A.1541.

23 Yoshifumi Sakai. A Data Structure for the Maximum-Sum Segment Problem with Offsets. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, volume 296 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.26.

24 Tadao Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science*, 61:191–200, 2002. CATS'02, Computing: the Australasian Theory Symposium. doi:10.1016/S1571-0661(04)00313-5.

25 Hisao Tamaki and Takeshi Tokuyama. Algorithms for the maximum subarray problem based on matrix multiplication. *Interdisciplinary Information Sciences*, 6(2):99–104, 2000. doi:10.4036/iis.2000.99.

26 Lusheng Wang and Ying Xu. SEGID: identifying interesting segments in (multiple) sequence alignments. *Bioinform.*, 19(2):297–298, 2003. doi:10.1093/BIOINFORMATICS/19.2.297.

27 Dan E Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983.

28 Hung-I Yu, Tien-Ching Lin, and D. T. Lee. Finding maximum sum segments in sequences with uncertainty. *Theor. Comput. Sci.*, 850:221–235, 2021. doi:10.1016/J.TCS.2020.11.005.

Cutwidth Versus BFS-Width with Applications to Graph Reconstruction from Distance Queries

Chirag Kaudan
Oregon State University, Corvallis, OR, USA

Amir Nayyeri
Oregon State University, Corvallis, OR, USA

Abstract

Eppstein, Goodrich, and Liu [ESA 2025] introduced a new graph parameter, called BFS-width, and gave polylogarithmic bounds on it for bounded bandwidth graphs. Their bounds naturally imply several applications, e.g. in graph reconstruction via shortest path distance queries, graph drawing, and matrix reordering.

We study this parameter for a broader class of graphs, namely bounded cutwidth graphs. We prove a sublinear upper bound on the BFS-width of bounded cutwidth graphs and show that our bounds are asymptotically tight. Our upper bound implies the first deterministic algorithm for reconstructing a bounded cutwidth graph with a subquadratic number of shortest path distance queries.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Graph algorithms, graph theory, cutwidth, pathwidth, BFS-width

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.24

Funding The authors were supported by NSF grants CCF-1941086 and CCF-2311180.

1 Introduction

Recently, Eppstein, Goodrich, and Liu [13] introduced the parameter **BFS-width** of a graph, defined as the maximum cardinality of a layer over all breadth-first search trees of the graph. More formally, for an undirected graph $G = (V, E)$, and any vertex $v \in V$, the $\text{bfsw}(G, v)$ is the maximum number of vertices that are at the same distance from v, i.e. $\max_{i \geq 0} |L_i(v)|$, where $L_i(v)$ is the set of all vertices at distance i of v. In turn, the BFS-width of the graph is defined as $\text{bfsw}(G) = \max_{v \in V}(\text{bfsw}(G, v))$.

BFS-width is a natural parameter to study, as the width of a breadth-first search tree plays a central role in many graph algorithms, for example in the computation of small separators. Eppstein, Goodrich, and Liu study BFS-width in relation to bandwidth, a parameter whose exact computation is NP-hard [14, 15], and show polylogarithmic upper and lower bounds on the BFS-width of bounded bandwidth graphs. Consequently, simple BFS-based algorithms perform well on bounded-bandwidth graphs for a variety of tasks. Specifically, Eppstein, Goodrich, and Liu show applications in analysis of the Cuthill-McKee algorithm, graph drawing, and graph reconstruction from shortest-path distance queries.

Motivated both by this theoretical perspective and by the analysis of simple BFS-based algorithms, we investigate whether there exist families of graphs beyond bounded bandwidth that admit sublinear BFS-width.

© Chirag Kaudan and Amir Nayyeri;
licensed under Creative Commons License CC-BY 4.0

Overview of Results

Cutwidth vs. BFS Width

The next natural family to consider is bounded cutwidth graphs. Our primary result shows that such graphs have sublinear BFS-width, although the width may still be polynomially large. Specifically, we prove that

$$\mathrm{bfsw}(G) = O\Big((\mathrm{cw}(G))^2 \cdot n^{1-\epsilon(\mathrm{cw}(G))}\Big),$$

where $\epsilon(x) = 2^{-(2x+1)}$, and $\mathrm{cw}(G)$ denotes the cutwidth of G (see Theorem 7).

Moreover, we show a nearly matching lower bound by constructing an infinite family of trees such that for every tree T in the family with n vertices,

$$\mathrm{bfsw}(T) = \Omega\Big(\frac{1}{\mathrm{cw}(T)} \cdot n^{1-\epsilon\left(\frac{\mathrm{cw}(T)}{2}-1\right)}\Big).$$

(see Theorem 4)

Our upper bound can be stated more generally in terms of pathwidth and maximum degree. Specifically, we prove that

$$\mathrm{bfsw}(G) = O\big(\Delta(G)^2 \cdot n^{1-\epsilon(\mathrm{pw}(G))}\big),$$

where $\mathrm{pw}(G)$ and $\Delta(G)$ denote the pathwidth and maximum degree of G, respectively. Since bounded degree together with bounded pathwidth implies bounded cutwidth (and vice versa up to constant factors), this formulation decouples the dependence on $\Delta(G)$ and $\mathrm{pw}(G)$ and is therefore more general. Our lower bound can similarly be expressed in terms of pathwidth.

We note that a bound on the maximum degree is necessary to obtain sublinear BFS-width. For example, a star has linear BFS-width. In particular, bounded pathwidth graphs with unbounded maximum degree may have linear BFS-width. Furthermore, even among bounded-degree trees, linear BFS-width is possible: a binary tree may have more than half of its vertices in its last layer, and therefore linear BFS-width. Thus, the family of bounded-degree trees does not, by itself, guarantee sublinear BFS-width. Hence, it appears that extending beyond bounded cutwidth – e.g., to bounded pathwidth graphs or to bounded-degree graphs of bounded treewidth – breaks the guarantee of sublinear BFS-width.

Finally, since $\mathrm{bw}(G) \leq 2\,\mathrm{bfsw}(G)$, as shown by Eppstein, Goodrich, and Liu, our result implies a sublinear bound on the bandwidth of bounded-cutwidth graphs. In particular,

$$\mathrm{bw}(G) = O\Big((\mathrm{cw}(G))^2 \cdot n^{1-\epsilon(\mathrm{cw}(G))}\Big).$$

Graph Reconstruction

We apply our results to the hidden graph reconstruction problem, which asks to reconstruct a graph which has an invisible edge set using queries to an oracle that returns some information about the graph e.g. a shortest path between a pair of vertices. In this paper, the query model we consider is the shortest path distance model, which returns the shortest path *length* between the queried pair of vertices. We assume that graphs are connected for the graph reconstruction problem with shortest path distance queries, as there are disconnected graphs which require $\Omega(n^2)$ queries to reconstruct.

A simple, natural algorithm for graph reconstruction with shortest path distance queries first constructs breadth-first search layers and then simply queries all possible edges, which we know must lie between vertices in the same or consecutive BFS layers. That is, the

algorithm simply queries all pairs of vertices that lie in BFS layers at most one apart. Of course, if the BFS-width of an n vertex graph is $\Omega(n)$, then this algorithm must make $\Omega(n^2)$ queries to reconstruct it. Conversely, if the graph has sublinear BFS-width then we obtain a subquadratic query reconstruction algorithm, improving on the trivial $\Theta(n^2)$ query algorithm which simply queries all pairs of vertices. Eppstein, Goodrich, and Liu show that this algorithm based on BFS layers makes nearly linear queries to reconstruct any bounded bandwidth graph.

We show that this algorithm makes a subquadratic number of queries for bounded cutwidth graphs, specifically, $O(\mathrm{cw}(G)^2 n^{2-\frac{1}{2^{2\,\mathrm{cw}(G)+1}}})$ queries for any graph G on n vertices and cutwidth $\mathrm{cw}(G)$ (see Theorem 10). Our result is the first subquadratic deterministic algorithm to reconstruct bounded cutwidth graphs to the best of our knowledge. A randomized algorithm with $\widetilde{O}(n^{3/2})$ queries to reconstruct bounded degree graphs was known before [18, 21].

2 Background

In this section, we first briefly review the basic definitions and notation used throughout the paper. We then provide a more detailed overview of graph width parameters, as they are the central subject of study in the paper.

2.1 Basic Definitions and Notation

In this paper, all graphs are unweighted. For a graph $G = (V, E)$, we use $|V(G)|$ to refer to the cardinality of the vertex set of G. We refer to this quantity $|V(G)|$ as the **order** of G and the quantity $|E(G)|$ as the **size** of G. For a vertex $v \in V(G)$, we use $N_G(v)$ to denote the set of all neighbors of v in the graph G and $N_G[v] = N_G(v) \cup \{v\}$, the open and closed neighborhood of v respectively. Analogously, for a graph $G = (V, E)$ and a set $S \subseteq V(G)$, we let $N_G[S] = \{u \in V(G) \mid \exists a \in S, d_G(a, u) \leq 1\}$ and $N_G(S) = N_G[S] \setminus S$ be the closed neighborhood and open neighborhood of S in G respectively where $d_G(x, y)$ is the shortest path distance between vertices x and y in G. For a graph G, we let $\Delta(G)$ denote the maximum degree of G. For a subset $S \subseteq V(G)$ of a graph G, we denote the subgraph of G induced by S with $G[S]$.

For any non-trivial tree T, we define $\ell(T)$ as the number of leaves in T. For the trivial tree T_1 on one vertex, we explicitly define $\ell(T_1) = 1$, despite T_1 having no leaves. We define a uniform depth tree as a rooted tree in which every leaf has the same distance from the root.

For any vector $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and any real $p \geq 1$, we use $\|x\|_p$ to denote the p-norm of x: $\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$.

We refer to the binary tree of height k with $2^{k+1} - 1$ vertices as a **full binary tree** of height k.

2.2 Preliminaries on Width Parameters and Strahler Numbers

We formally define the graph width parameters bandwidth, pathwidth, and cutwidth, show some of the well-known relationships between them, and define the Strahler number, a well-studied graph parameter for directed rooted trees (arborescences) which we will use to develop our results.

A **linear layout**, or **layout**, of an undirected graph G with $n = |V(G)|$ is a bijection $\phi : V(G) \to [n] = \{1, 2, \ldots, n\}$. The length of an edge $uv \in E(G)$ in a layout ϕ of G is $|\phi(u) - \phi(v)|$. The bandwidth of ϕ, denoted $\mathrm{bw}(\phi)$, is the maximum length of any edge $uv \in E(G)$. The **bandwidth** $\mathrm{bw}(G)$ of graph G is the minimum bandwidth over all bijections $\varphi : V(G) \to [n]$. The notion of bandwidth is equivalent to finding the length of the longest edge on a linear layout of the graph on a number line that minimizes the longest edge length.

Given an undirected graph $G = (V, E)$, a layout ϕ of G, and an integer i, the **edge cut of ϕ at position i**, denoted $\theta(i, \phi, G)$, is defined $\theta(i, \phi, G) = |\{uv \in E(G) \mid u \in \{x \in V(G) \mid \phi(x) \leq i\} \wedge v \in \{x \in V(G) \mid \phi(x) > i\}\}|$. That is, if we imagine a layout ϕ of G as an ordering of the vertices on a number line, the edge cut of ϕ at position i is the number of edges of G crossing the vertical line between vertex i and $i + 1$. The maximum edge cut of ϕ, denoted $\Theta(\phi, G)$, is the maximum value of $\theta(i, \phi, G)$ over all i. The **cutwidth** $\mathrm{cw}(G)$ of graph G is the minimum value of $\Theta(\phi, G)$ over all bijections $\phi : V(G) \to [n]$. We will use the informal terms "left" and "right" of an edge cut of layout ϕ of G at position i to refer to the sets $\{x \in V(G) \mid \phi(x) \leq i\}$ and $\{x \in V(G) \mid \phi(x) > i\}$ respectively, viewing the layout as embedding the graph on a number line. We specifically note that both of the graph classes bounded cutwidth and bounded pathwidth are monotone graph classes i.e. they are closed under taking *any* subgraph (in particular, not merely induced subgraphs).

A **path decomposition** of $G = (V, E)$ is a sequence of subsets of $V(G)$, called bags, such that each edge of G is contained in a bag and each vertex appears in some contiguous subsequence of the bags. The width of a path decomposition $(V_i)_{i \geq 1}$ of G is $\max_i |V_i| - 1$ and the **pathwidth** of undirected graph G, denoted $\mathrm{pw}(G)$, is the minimum width over any path decomposition of G.

The Strahler number, also called the Horton-Strahler number, of a rooted, directed tree T is given by the following recursive definition.

▶ **Definition 1.** *Let T be a rooted, directed tree. The **Strahler number of a vertex** $x \in V(T)$, denoted $\nu(x)$, is defined recursively below.*

$$\nu(x) = \begin{cases} 1, & \text{if } x \text{ is a leaf,} \\ 1 + k, & \text{if } x \text{ has at least 2 distinct children, each with Strahler number } k = \max\limits_{x_i \in \mathrm{Ch}(x)} \{\nu(x_i)\} \\ \max\limits_{x_i \in \mathrm{Ch}(x)} \{\nu(x_i)\}, & \text{otherwise.} \end{cases}$$

*where $\mathrm{Ch}(v)$ denotes the set of children of a vertex in T. Note that the Strahler number of any vertex must be a positive integer. **The Strahler number of a rooted, directed tree** T is the Strahler number of its root, denoted $\nu(T)$.*

For technical convenience, we also define a modified notion of the Strahler number called the t-shifted Strahler number which retains the recursive definition of the Strahler number but assigns t to each leaf instead of 1 as the base case, for some positive integer t.

▶ **Definition 2.** *Let T be a rooted, directed tree. The **t-shifted Strahler number of a vertex** $x \in V(T)$, denoted $\nu_t(x)$, is defined recursively below.*

$$\nu_t(x) = \begin{cases} t, & \text{if } x \text{ is a leaf,} \\ 1 + k, & \text{if } x \text{ has at least 2 distinct children, each with Strahler number } k = \max\limits_{x_i \in \mathrm{Ch}(x)} \{\nu_t(x_i)\} \\ \max\limits_{x_i \in \mathrm{Ch}(x)} \{\nu_t(x_i)\}, & \text{otherwise.} \end{cases}$$

*where $\mathrm{Ch}(v)$ denotes the set of children of a vertex in T. Note that the t-shifted Strahler number of any vertex must be an integer at least t. **The t-shifted Strahler number of a rooted, directed tree** T is the t-shifted Strahler number of its root, denoted $\nu_t(T)$.*

For any directed, rooted tree T, the relationship $\nu_t(T) = \nu(T) + t - 1$ follows immediately from the definitions of Strahler number and t-shifted Strahler number, hence the name.

We briefly discuss some of the many known relationships between these graph parameters. Firstly, note that bounded bandwidth implies bounded maximum degree and also implies bounded pathwidth, but bounded pathwidth does not imply bounded degree e.g. the star graph $K_{1,n-1}$ on n vertices has pathwidth 1. That bounded bandwidth implies bounded pathwidth can be seen via the following alternate definition of pathwidth: The pathwidth of G can also be seen as a layout parameter, called the vertex separation number, defined nearly identically to cutwidth but defined in terms of number of vertices with neighbors across a cut rather than number of edges crossing a cut [8, 12].

This alternate definition of pathwidth as the vertex separation number yields the inequality $\mathrm{pw}(G) \leq \mathrm{cw}(G) \leq \Delta\,\mathrm{pw}(G)$ for any graph G with maximum degree Δ. Also, $\mathrm{cw}(G) \leq \Delta(G)\,\mathrm{bw}(G)$ for any graph G [10].

Furthermore, bounded BFS-width implies bounded bandwidth [13]. This can be seen by taking the linear ordering of a graph G induced by the breadth-first search tree that realizes the BFS-width of G and noting that edges of G must lie between vertices in the same or consecutive BFS layers.

In this paper, we need to work with different width parameters of trees as well as their Strahler number. Technically, the Strahler number is defined for directed rooted trees, while the width parameters are defined for undirected trees. Whenever the root is clear from the context, we abuse notation by denoting both of these directed and undirected trees by T. Specifically, we use $\mathrm{pw}(T)$, $\mathrm{cw}(T)$, and $\Delta(T)$ for a directed rooted tree T, where we mean the pathwidth, cutwidth and the maximum degree of the underlying undirected tree in order.

In this case, the inequality $\mathrm{pw}(T) \leq \nu(T) \leq 2\,\mathrm{pw}(T) + 2$ holds [20]. The following lemma summarizes some of the most important inequalities for our purposes that were discussed above.

▶ **Lemma 3.** *For a directed rooted tree T with maximum degree Δ, we have*

$$\frac{\mathrm{cw}(T)}{\Delta(T)} \leq \mathrm{pw}(T) \leq \nu(T) \leq 2\,\mathrm{pw}(T) + 2 \leq 2\,\mathrm{cw}(T) + 2$$

3 Lower Bound Construction

In this section, we prove the following theorem.

▶ **Theorem 4.** *For any $\nu \geq 2$, there exists an infinite family of (uniform depth) trees $\mathcal{C}_\nu$ of Strahler number ν and cutwidth and pathwidth at most ν such that for any $T \in \mathcal{C}_\nu$ we have*

$$\mathrm{bfsw}(T) \geq \frac{1}{\nu - 1} \cdot |V(T)|^{1-1/2^{\nu-1}} .$$

We explicitly construct the infinite sequences of trees $\mathcal{C}_\nu = (C_{\nu,p})_{p \geq \nu}$ (one sequence per each value of $\nu \in \{2, 3, \ldots\}$) with Strahler number $\nu \geq 2$. First, we consider the case $\nu = 2$.

Base Case ($p = 2$). The tree $C_{2,2}$ is a rooted full binary tree of height 1, i.e. a root with two leaves as children.

Recursive case ($p > 2$). The tree $C_{2,p}$ is the rooted binary tree defined by its left and right subtrees: (1) its left subtree is $C_{2,p-1}$, and (2) its right subtree is a rooted path of length $p - 2$, i.e. $(u_1, u_2, \ldots, u_{p-1})$ with root u_1. Let $C_{2,p-1}$ be as defined recursively with

root r_{p-1}. One can also view the construction $C_{2,p}$ as introducing p vertices to the graph $C_{2,p-1}$ that form a path of length $p-1$ (as an induced subgraph), of which one endpoint is a leaf of $C_{2,p}$ and the other endpoint is the root of $C_{2,p}$.

Figure 1 Left: The recursive construction of $C_{2,p}$: $C_{2,2}$, $C_{2,3}$ and $C_{2,4}$ in order. Right: Layout $L_{2,4}$ of $C_{2,4}$, which has cutwidth 2.

▶ **Lemma 5.** *For any $p \geq 2$, $C_{2,p}$ has Strahler number 2, cutwidth at most 2, p leaves, and $p(p+1)/2$ vertices.*

Proof. We use induction on p. The base case is a tree with three vertices, two leaves, and Strahler number two, which satisfies the statement of the lemma.

Inductively, $C_{2,p}$ is a tree with $\ell(C_{2,p-1})+1$ leaves, and $|V(C_{2,p-1})|+p$ vertices. Solving the recurrence, one obtains $\ell(C_{2,p}) = p$ and $|V(C_{2,p})| = p(p+1)/2$, respectively. In addition, $\nu(C_{2,p-1}) = 2$ (that is, $\nu(r_{p-1}) = 2$) inductively and the Strahler number of the right subtree of $C_{2,p}$, which is a path rooted at one of its endpoints, is one. Hence, $\nu(C_{2,p}) = 2$ and $\mathrm{pw}(C_{2,p}) \leq 2$ by Lemma 3.

Finally, we show that $C_{2,p}$ has cutwidth at most two by inductively constructing a layout $L_{2,p}$ of cutwidth two for $C_{2,p}$, with the additional property that the root is the leftmost vertex in $L_{2,p}$.

For the base case $p = 2$, let $L_{2,2}$ be any layout of $C_{2,2}$ in which the root is the leftmost vertex. Since the root has degree two, this layout has cutwidth at most two (in fact, the cutwidth of $C_{2,2}$ is one).

For $p > 2$, let r be the root of $C_{2,p}$. The subtrees of r consist of a copy of $C_{2,p-1}$ with root r' and a path $(u_1, \ldots, u_{p-1})$, where u_1 is a child of r. We define $L_{2,p}$ to be the layout

$$r, u_1, \ldots, u_{p-1} \circ L_{2,p-1},$$

where the copy of $L_{2,p-1}$ corresponds to the inductively constructed layout of $C_{2,p-1}$ with r' as its leftmost vertex (see Figure 1 right).

Consider any cut in this layout. Any cut occurring before r' intersects at most two edges: possibly one edge from the path $r, u_1, \ldots, u_{p-1}$, and always the edge connecting r to r'.

Any cut occurring after r' intersects at most two edges of the copy of $C_{2,p-1}$ by the induction hypothesis, and no additional edges, as all other edges of $C_{2,p}$ are on the left side of r'.

Thus every cut contains at most two edges, and therefore $C_{2,p}$ has cutwidth at most two. ◀

Next, we build $\mathcal{C}_\nu = (C_{\nu,p})_{p \geq \nu}$ for $\nu \geq 3$, recursively; see Figure 2, left and middle, for an illustration. To construct $C_{\nu,p}$, let T be a copy of $C_{2,p^{2^{\nu-2}}}$, and let $T'_1, \ldots, T'_{p^{2^{\nu-2}}}$ be $p^{2^{\nu-2}}$ copies of $C_{\nu-1,p}$. Now, construct $C_{\nu,p}$ by taking T and identifying its leaves with the roots of $T'_1, \ldots, T'_{p^{2^{\nu-2}}}$.

▶ **Lemma 6.** *For any $\nu \geq 2$ and any $p \geq \nu$, $C_{\nu,p}$ has Strahler number ν, cutwidth at most ν, $p^{2^{\nu-1}-1}$ leaves, and at most $(\nu-1)p^{2^{\nu-1}}$ vertices.*

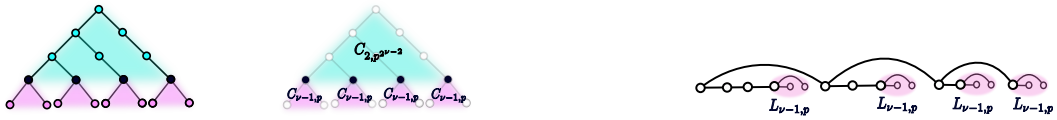


Figure 2 The recursive construction of $C_{\nu,p}$. Left: $C_{3,2}$ is composed of one copy of $C_{2,4}$ and four copies of $C_{2,2}$. Middle: Generally, $C_{\nu,p}$ is composed of one copy of $C_{2,p^{2^{\nu-2}}}$ and $p^{2^{\nu-2}}$ copies of $C_{\nu-1,p}$ – Lemma 6 shows that $C_{\nu-1,p}$ has $p^{2^{\nu-2}}$ leaves. Right: $L_{\nu,p}$ construction. Copies of $L_{\nu-1,p}$ are recursively embedded next to the leaves in the layout $L_{2,p^{2^{\nu-2}}}$.

Proof. We proceed by induction on the value ν. The basis step is provided by Lemma 5.

For the inductive step $\nu \geq 3$, note that for each $i \in \{1, \ldots, p^{2^{\nu-2}}\}$, T'_i is a copy of $C_{\nu-1,p}$. Therefore, by the induction hypothesis, $\nu(T'_i) = \nu - 1$, $\ell(T'_i) = p^{2^{\nu-2}-1}$, and $|V(T'_i)| \leq (\nu-2)p^{2^{\nu-2}}$. Also, by Lemma 5,

$$\ell(T) = p^{2^{\nu-2}},\ |V(T)| = p^{2^{\nu-2}}(p^{2^{\nu-2}}+1)/2 \leq p^{2^{\nu-1}},\text{ and } \nu(T) = 2$$

Therefore, we have

$$\ell(C_{\nu,p}) = \sum_{i\in[\ell(T)]} \ell(T'_i) = \ell(T)\cdot p^{2^{\nu-2}-1} = p^{2^{\nu-2}}\cdot p^{2^{\nu-2}-1} = p^{2^{\nu-1}-1}.$$

Moreover, we have

$$\begin{aligned}
|V(C_{\nu,p})| &= |V(T)| + \sum_{i\in[\ell(T)]} |V(T'_i)| - \ell(T) \\
&\leq p^{2^{\nu-1}} + p^{2^{\nu-2}}(\nu-2)p^{2^{\nu-2}} - p^{2^{\nu-2}} \\
&\leq p^{2^{\nu-1}} + (\nu-2)p^{2^{\nu-1}} \\
&= (\nu-1)\,p^{2^{\nu-1}}.
\end{aligned}$$

To show that the Strahler number of $C_{\nu,p}$ is ν, we recall that by construction $C_{\nu,p}$ consists of a copy of $T = C_{2,p^{2^{\nu-2}}}$ and $p^{2^{\nu-2}}$ copies of $C_{\nu-1,p}$ whose roots are identified with the leaves of the copy of T. Let R be the set of vertices of $C_{\nu,p}$ that correspond to these identified roots (i.e. the roots of the T_i's that were identified with leaves of the copy of T). By the induction hypothesis, the Strahler number on each vertex of R is $\nu - 1$. Observe then that $\nu(C_{\nu,p}) = \nu_t(T)$ for $t = \nu - 1$. Further, $\nu(T) = 2$ by Lemma 5. It follows then that $\nu(C_{\nu,p}) = \nu_t(T) = \nu(T) + t - 1 = 2 + (\nu-1) - 1 = \nu$.

Finally, we show that the cutwidth of $C_{\nu,p}$ is at most ν by inductively constructing a layout $L_{\nu,p}$ of cutwidth ν.

The base case $\nu = 2$ is shown in Lemma 5. For later use, we note an additional property of the layout $L_{2,p}$ constructed there: any cut that appears immediately after a leaf in the layout has width one. In particular, in the (recursive) layout

$$r, u_1, \ldots, u_{p-1} \circ L_{2,p-1},$$

the cut immediately after u_{p-1} contains exactly one edge, namely (r, r'), where r' is the root of the copy of $C_{2,p-1}$. All other cuts immediately following a leaf occur within $L_{2,p-1}$ and therefore have width one within $L_{2,p-1}$ inductively. Moreover, every edge not belonging to the copy of $C_{2,p-1}$ has both endpoints to the left of r' in $L_{2,p}$, so such edges are not crossed by those cuts.

We now construct $L_{\nu,p}$ for $\nu > 2$; see Figure 2, right, for illustration. The tree $C_{\nu,p}$ consists of a copy T of $C_{2,q}$ together with copies $T'_1, \ldots, T'_q$ of $C_{\nu-1,t}$ whose roots are identified with the leaves of $C_{2,q}$, for appropriate values q and t (their exact values are not needed for the remainder of the proof).

To build $L_{\nu,p}$, start with the layout $L_{2,q}$. For each leaf v in this layout, insert to its right a copy of $L_{\nu-1,t}$, and identify the root of that copy with v. Since in $L_{\nu-1,t}$ the root is the leftmost vertex, this identification does not increase the cutwidth of $L_{\nu-1,t}$.

Consider an arbitrary cut in $L_{\nu,p}$. Such a cut either: (i) crosses only edges of $L_{2,q}$, or (ii) crosses one edge of $L_{2,q}$ together with edges from a single copy of $L_{\nu-1,t}$.

In case (i), the cut has cardinality at most two, since $L_{2,q}$ has cutwidth two. In case (ii), the cut crosses at most one edge from $L_{2,q}$ and, by the inductive hypothesis, at most $\nu - 1$ edges from the copy of $L_{\nu-1,t}$; hence its total width is at most ν.

Therefore, $L_{\nu,p}$ has cutwidth at most ν, completing the proof. ◀

Using the lemmas above, we now provide a proof of Theorem 4.

Proof of Theorem 4. Let $\mathcal{C}_\nu = \{C(\nu,p)\}_{p\geq\nu}$. Let $T = C_{\nu,p}$. By Lemma 6, we have

$$\ell(T) = p^{2^{\nu-1}-1},$$

$$|V(T)| \leq (\nu-1)p^{2^{\nu-1}} \;\Rightarrow\; |V(T)|^{\frac{2^{\nu-1}-1}{2^{\nu-1}}} \leq \left((\nu-1)p^{2^{\nu-1}}\right)^{\frac{2^{\nu-1}-1}{2^{\nu-1}}}$$

$$= (\nu-1)^{\frac{2^{\nu-1}-1}{2^{\nu-1}}} \cdot p^{2^{\nu-1}-1} \leq (\nu-1)\cdot\ell(T).$$

So, $\ell(T) \geq \frac{|V(T)|^{1-\frac{1}{2^{\nu-1}}}}{\nu-1}$. Therefore, $\mathrm{bfsw}(T) \geq \frac{n^{1-\frac{1}{2^{\nu-1}}}}{\nu-1}$. ◀

4 Upper Bound on BFS-width of Bounded Cutwidth Graphs

In this section, we prove the following theorem.

▶ **Theorem 7.** *For any graph $G = (V,E)$ we have*

$$\mathrm{bfsw}(G) \leq 2\Delta^2 \cdot n^{1-1/2^{2\,\mathrm{pw}(G)+1}} = O_\Delta(n^{1-\epsilon(\mathrm{pw}(G))}),$$

where $\epsilon(x) = 1/2^{2x+1}$, $\Delta = \Delta(G)$, and $n = |V(G)|$.

To prove Theorem 7, we will first show that any graph G contains a uniform depth tree T with $\ell(T) = \mathrm{bfsw}(G)$ and Strahler number at most $2\,\mathrm{cw}(G) + 2$. Next, we prove an upper bound on the number of leaves of any uniform depth tree with respect to its Strahler number and maximum degree $\Delta(G)$, thereby obtaining an upper bound for $\mathrm{bfsw}(G)$ as a function of $\mathrm{pw}(G)$ or $\mathrm{cw}(G)$ via Strahler number.

We show the following lemma.

▶ **Lemma 8.** *Any graph $G = (V,E)$ contains a uniform depth tree T with Strahler number at most $2\,\mathrm{pw}(G) + 2$ that has exactly $\mathrm{bfsw}(G)$ leaves.*

Proof. We prove the statement by obtaining such a tree from G. Let $\ell = \mathrm{bfsw}(G)$. We first run BFS rooted at a vertex r for which the BFS-width of G is realized i.e. any vertex r such that $\mathrm{bfsw}(G,r) = \mathrm{bfsw}(G)$. Then we scan through the BFS layers and delete every layer deeper than the first layer L that has ℓ vertices. We then drop all edges of G not in the BFS tree and iteratively delete all leaves which are not in layer L. This results in a uniform depth tree T with at most $|V(G)|$ vertices. Since T is a subgraph of G, $\mathrm{pw}(T) \leq \mathrm{pw}(G)$. Also, $\mathrm{bfsw}(T) = \ell$, as we never delete any vertices in layer L. By Lemma 3, we know that T has Strahler number at most $2\,\mathrm{pw}(G) + 2$. ◀

4.1 Lower Bound on Number of Vertices in a Uniform Depth Tree with Bounded Strahler Number

The following theorem gives a lower bound on the order of any uniform depth tree with a given Strahler number and a given number of leaves. This theorem, equivalently, gives an upper bound on the number of leaves of a uniform depth tree with a given Strahler number and a given order.

▶ **Theorem 9.** *Let T be a uniform depth tree with Strahler number $\nu \geq 2$ with maximum degree Δ. Then $\ell(T) \leq 2\Delta^2 |V(T)|^{1-1/2^{\nu-1}}$.*

Proof. We define $f(\nu) = 2^{\nu-1}/(2^{\nu-1}-1) = (1-1/2^{\nu-1})^{-1}$ and prove $|V(T)| \geq (\ell(T))^{f(\nu)}/(2\Delta^2)$. The statement of the lemma then follows from the following calculation

$$|V(T)| \geq \frac{(\ell(T))^{f(\nu)}}{2\Delta^2} \Rightarrow \ell(T) \leq |V(T)|^{1-1/2^{\nu-1}} \cdot (2\Delta^2)^{1-1/2^{\nu-1}} \leq 2\Delta^2 \cdot |V(T)|^{1-1/2^{\nu-1}}.$$

Let $P \subseteq V(T)$ be the subset of all vertices of Strahler number ν in T. Observe that the induced subgraph $T[P]$ is a downward path starting at the root, as (1) the root has value ν, (2) each non-root vertex of value ν has a parent of value ν, and (3) each vertex of value ν has at most one child of value ν. Let $P' \subseteq P$ be the subset of P that includes vertices with at least one child outside P. As P' is a subset of the downward path P, it contains no pair of vertices at the same depth. Let $(p_1, p_2, \ldots, p_t)$ be the set of vertices in P' ordered inversely with respect to their depth.

Let $Q \subseteq V(T)$ be the set of neighbors of P that are not in P, i.e. $Q = N_T(P)$. For each $q \in Q$, let T_q be the subtree of T rooted at q. We use $V(T_q)$ to denote T_q's vertex set, $\ell(T_q)$ to denote the number of its leaves, $\nu(T_q)$ to denote its Strahler number, and $h(T_q)$ the (maximum) number of *vertices* in a root to leaf path in T_q, i.e. $h(T_q) = \text{height}(T_q) + 1$.

First, we derive a lower bound on $|V(T)|$, from the simple lower bound $|V(T_q)| \geq h(T_q)$ for all $q \in Q$. Specifically, $|V(T)| \geq \sum_{q \in Q} h(T_q)$. Further, we observe that each vertex in $p_i \in P'$ has at least one child in Q, by the definition of P'. For $i \in \{1, 2, \ldots, t\}$ let q_i be an arbitrary child of p_i with Strahler number less than ν, and observe that $h(T_{q_i}) \geq i$. Therefore,

$$|V(T)| \geq \sum_{q \in Q} h(T_q) \geq \sum_{i=1}^{t} h(T_{q_i}) \geq \sum_{i=1}^{t} i = \frac{t(t+1)}{2} \geq \frac{t^2}{2}. \tag{1}$$

Now, we consider two cases based on the cardinality of Q. In what follows let $\ell = \ell(T)$.

Case (I) $|Q| \geq \ell^{f(\nu)/2}$. We have,

$$t\Delta \geq |Q| \geq \ell^{f(\nu)/2} \Rightarrow t \geq \frac{\ell^{f(\nu)/2}}{\Delta}.$$

Therefore, by Equation (1), we have

$$|V(T)| \geq \frac{\ell^{f(\nu)}}{2\Delta^2},$$

as desired.

Case (II) $|Q| \leq \ell^{f(\nu)/2}$. We proceed by using induction on ν.

For the base case, $\nu = 2$, the values of all vertices in $V \setminus P$ are one. Hence $T[V \setminus P]$ is a collection of paths, each connecting a leaf of T to a (perhaps identical) vertex in Q. Therefore, $|Q| = \ell = \ell^{f(\nu)/2}$, as $f(\nu) = 2$ for $\nu = 2$. Therefore, the base case is implied by the argument of Case (I).

For the inductive step we first observe that $\ell = \sum_{q\in Q} \ell(T_q)$, as each leaf of T is a descendant of exactly one vertex in Q (where a vertex is considered a descendant of itself). Next, we use Hölder's inequality: for any pair of vectors x, y in Euclidean space and any pair $\alpha, \beta \in \mathbb{R}^+$ such that $1/\alpha + 1/\beta = 1$, we have $\|x \cdot y\|_1 \le \|x\|_\alpha \cdot \|y\|_\beta$. Now let $x = (\ell(T_q))_{q\in Q}$ (with an arbitrary order on Q) and $y = \mathbf{1}$, the all 1 vector. We have:

$$\ell = \sum_{q\in Q} \ell(T_q) \le \left(\sum_{q\in Q} (\ell(T_q))^\alpha\right)^{\frac{1}{\alpha}} \left(\sum_{q\in Q} 1^\beta\right)^{\frac{1}{\beta}} \le \left(\sum_{q\in Q} (\ell(T_q))^\alpha\right)^{\frac{1}{\alpha}} \cdot \Big|Q\Big|^{\frac{\alpha-1}{\alpha}},$$

where the last equality holds as $1/\alpha + 1/\beta = 1 \Rightarrow \beta = \alpha/(\alpha-1)$. Raising the sides to α,

$$\ell^\alpha \le \sum_{q\in Q} (\ell(T_q))^\alpha \cdot |Q|^{\alpha-1}. \tag{2}$$

Now, we set $\alpha := f(\nu{-}1)$, and observe $\alpha = f(\nu{-}1) > 1$ for $\nu \ge 3$, justifying our use of Hölder's inequality. Next, we show $\sum_{q\in Q}(\ell(T_q))^\alpha \le 2\Delta^2 \cdot |V(T)|$ by showing $\ell(T_q)^\alpha \le 2\Delta^2 \cdot |V(T_q)|$ for every $q \in Q$. We consider two possible cases.

If $\nu(T_q) = 1$ then T_q is a path, possibly with just one vertex and no edge. Therefore, $(\ell(T_q))^\alpha \le 2\Delta^2 \cdot |V(T_q)|$, as (i) $\ell(T_q) = 1$, (ii) $|V(T_q)| \ge 1$, and (iii) $\alpha = f(\nu-1) > 0$ as $\nu \ge 3$ and (iv) $\Delta \ge 2$ (it is important to note that Δ is the maximum degree in the original T).

Otherwise, if $\nu(T_q) > 1$, then we use the induction hypothesis. To that end, note $f(\cdot)$ is a decreasing function over the interval $(1, \infty)$, and $\nu(T_q) \le \nu - 1$ for all $q \in Q$. Thus,

$$(\ell(T_q))^\alpha = (\ell(T_q))^{f(\nu-1)} \le (\ell(T_q))^{f(\nu(T_q))} \le 2(\Delta(T_q))^2 \cdot |V(T_q)| \le 2\Delta^2 \cdot |V(T_q)|.$$

Substituting in Equation (2),

$$\ell^{f(\nu-1)} \le 2\Delta^2 \cdot |V(T)| \cdot |Q|^{f(\nu-1)-1} \Rightarrow |V(T)| \ge \frac{\ell^{f(\nu-1)} \cdot |Q|^{1-f(\nu-1)}}{2\Delta^2}.$$

Since $|Q| \le \ell^{f(\nu)/2}$ and $1 - f(\nu-1) < 0$,

$$\ell^{f(\nu-1)} \cdot |Q|^{1-f(\nu-1)} \ge \ell^{(f(\nu-1)+(1-f(\nu-1))\frac{f(\nu)}{2})} = \ell^{f(\nu)},$$

which completes the proof. The last equality is proved below.

$$\begin{aligned} f(\nu-1) + (1-f(\nu-1))f(\nu)/2 &= \frac{2^{\nu-2}}{2^{\nu-2}-1} + \Big(1 - \frac{2^{\nu-2}}{2^{\nu-2}-1}\Big) \cdot \frac{1}{2} \cdot \frac{2^{\nu-1}}{2^{\nu-1}-1} \\ &= \frac{2^{\nu-2}}{2^{\nu-2}-1} + \Big(\frac{-1}{2^{\nu-2}-1}\Big) \cdot \frac{2^{\nu-2}}{2^{\nu-1}-1} \\ &= \frac{2^{\nu-2}}{2^{\nu-2}-1}\Big(1 - \frac{1}{2^{\nu-1}-1}\Big) \\ &= \frac{2^{\nu-2}}{2^{\nu-2}-1} \cdot \frac{2^{\nu-1}-2}{2^{\nu-1}-1} = \frac{2^{\nu-1}}{2^{\nu-1}-1} = f(\nu). \end{aligned}$$

◀

4.2 Proof of Theorem 7

We conclude this section with a proof of Theorem 7 using the results from above.

Proof of Theorem 7. By Lemma 8, G contains a uniform depth tree T of Strahler number $\nu(T) \le 2\,\mathrm{pw}(G) + 2$ with at most $|V(G)|$ vertices and exactly $\mathrm{bfsw}(G)$ leaves, i.e. $\ell(T) = \mathrm{bfsw}(G)$. By Theorem 9,

$$\ell(T) \le 2\Delta^2 |V(T)|^{1-1/2^{\nu(T)-1}} \le 2\Delta^2 |V(T)|^{1-1/2^{2\,\mathrm{pw}(G)+1}} \le 2\Delta^2 |V(G)|^{1-1/2^{2\,\mathrm{pw}(G)+1}},$$

which completes the proof. ◀

5 Graph reconstruction algorithm

In this section, we apply the above results to the hidden graph reconstruction problem mentioned in the introduction. We formally define this problem, briefly summarize some of the known results, and then state our main theorem in this area.

In the **hidden graph reconstruction problem**, only the vertex set $V(G)$ of a hidden graph G is visible, and the goal is to reconstruct the edge set $E(G)$ by making queries to an oracle. A hidden graph reconstruction algorithm, referred to simply as a reconstruction algorithm, aims to discover the edge set with as few adaptively chosen queries as possible. The reconstruction task is considered complete once there is a unique graph, up to isomorphism, consistent with answers to all the queries. Several query models have been studied for graph reconstruction, including edge existence queries, edge counting, effective resistance queries, and shortest path queries [2, 3, 4, 11, 17, 22].

The query model that will be considered in this paper is the shortest path distance query model (SP query model). In the SP query model, we have access to a distance oracle: for any two vertices $u, v \in V(G)$, a query returns the *length* of the shortest path between u and v in G and returns ∞ if no such path exists. There are several motivations and applications to studying the hidden graph reconstruction problem with the SP query model [1, 7, 9, 16, 24].

Problem Background and Prior Work

Any unweighted graph can trivially be reconstructed using $\binom{n}{2}$ SP queries simply by querying every pair of vertices to check whether they are adjacent. Importantly, this upper bound is tight even for trees if we allow unbounded degree [22]. Moreover, the bound is tight if one allows disconnected graphs; reconstructing a graph with n vertices and one edge requires $\Omega(n^2)$ queries. In light of these observations, all non-trivial algorithmic results on graph reconstruction focus on connected, bounded degree graphs.

In 2013, Mathieu et al. [18, 21] showed that general bounded degree graphs can be reconstructed using $\tilde{O}_\Delta(n^{3/2})$ expected queries via a randomized algorithm, where $\tilde{O}(f(n))$ is shorthand for $O(f(n) \cdot \text{polylog}(n))$. This is still the best known general upper bound today. It is unknown whether there exists a deterministic algorithm to reconstruct bounded-degree graphs in $o(n^2)$ queries. Many graph classes less general than bounded degree graphs have been studied. Nearly linear query complexity algorithms, both randomized and deterministic, have been found for graph classes like chordal graphs, nearly chordal graphs, k-chordal graphs, and bounded treelength graphs [5, 6, 23, 19] and matching lower bounds for some of these classes have also been shown [5].

Our Algorithm

Our result for hidden graph reconstruction in this paper is a deterministic reconstruction algorithm for any graph G with bounded cutwidth using a subquadratic number of queries. More generally, we characterize the number of queries with respect to the maximum degree and the pathwidth of the graph – recall a bounded degree bounded pathwidth graph has bounded cutwidth and vice versa. We note again that this implies a subquadratic query reconstruction algorithm for the class of bounded cutwidth graphs and we emphasize that this reconstruction algorithm runs **without** knowledge of the value of parameters $\text{pw}(G)$ or $\text{cw}(G)$.

▶ **Theorem 10.** *Let $G = (V, E)$ be a graph. Then there is an algorithm to reconstruct G with at most $O((\Delta(G))^2 \cdot n^{2-\epsilon(\text{pw}(G))})$ shortest path distance queries where $\epsilon(x) = 1/2^{2x+1}$.*

Proof. By Theorem 7,

$$\mathrm{bfsw}(G) \leq 2(\Delta(G))^2 \cdot n^{1-\epsilon(\mathrm{pw}(G))} =: b \ .$$

The algorithm selects an arbitrary vertex $s \in V$. It runs $n-1$ queries to obtain s to v distances for all $v \in V \setminus \{s\}$. Then, it computes BFS layers $B_1, B_2, \ldots, B_k \subseteq V$ with respect to s, i.e. for each $i \in [k]$, B_i is the subset of vertices at distance i of s. We know all the edges of the graph are between vertices that belong to the same or consecutive layers of BFS. The algorithm, thus, queries all these pairs to find all possible edges, i.e. all pairs $u, v \in V$ that belong to the same B_i or consecutive B_i and B_{i+1}. Hence, setting $x_i = |B_i|$, the number of queries made in this part of the algorithm is bounded above by

$$\sum_{i\in[k]} x_i^2 + \sum_{i\in[k-1]} x_i x_{i+1} \leq 3 \sum_{i\in[k]} x_i^2 \ ,$$

as $x_i x_{i+1} \leq x_i^2 + x_{i+1}^2$ for all $i \in [k-1]$.

Next, we show that $\sum_{i\in[k]} x_i^2 \leq bn$, hence the algorithm makes at most

$$3bn + n - 1 = 6\Delta^2 \cdot n^{2-\epsilon(\mathrm{pw}(G))} + n - 1 = O(\Delta^2 \cdot n^{2-\epsilon(\mathrm{pw}(G))})$$

queries in total.

To that end, let $x = (x_1, x_2, \ldots, x_k) \in \mathbb{R}^k$ be a vector. Recall the Hölder's inequality for two vectors $f, g \in \mathbb{R}^k$ is $\|f \cdot g\|_1 \leq \|f\|_\alpha \cdot \|g\|_\beta$ with $1/\alpha + 1/\beta = 1$. Now, we apply the inequality with $f = g = x$, $\alpha = \infty$ and $\beta = 1$. So,

$$\sum_{i\in[k]} x_i^2 = \|x \cdot x\| \leq \|x\|_\infty \|x\|_1 = \max_{i\in[k]} |x_i| \cdot \sum_{i\in[k]} |x_i| \leq bn \ . \qquad \blacktriangleleft$$

References

1. Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling: Or, power-law degree distributions in regular graphs. *J. ACM*, 56(4), 2009. doi:10.1145/1538902.1538905.
2. Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005. doi:10.1137/S0895480103431071.
3. Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004. doi:10.1137/S0097539702420139.
4. Dana Angluin and Jiang Chen. Learning a hidden graph using o(log n) queries per edge. In John Shawe-Taylor and Yoram Singer, editors, *Learning Theory*, pages 210–223, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27819-1_15.
5. Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles, 2024. doi:10.48550/arXiv.2306.05979.
6. Paul Bastide and Carla Groenland. Quasi-linear distance query reconstruction for graphs of bounded treelength. In Édouard Bonnet and Pawel Rzazewski, editors, *19th International Symposium on Parameterized and Exact Computation, IPEC 2024, September 4-6, 2024, Royal Holloway, University of London, Egham, United Kingdom*, volume 321 of *LIPIcs*, pages 20:1–20:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.IPEC.2024.20.
7. Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matúš Mihaľák, and L. Shankar Ram. Network discovery and verification. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 127–138, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11604686_12.

8 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

9 Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006. Complex Networks. doi:10.1016/j.tcs.2005.12.009.

10 Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002. doi:10.1145/568522.568523.

11 W. Ellens, F.M. Spieksma, P. Van Mieghem, A. Jamakovic, and R.E. Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435(10):2491–2506, 2011. Special Issue in Honor of Dragos Cvetkovic. doi:10.1016/j.laa.2011.02.024.

12 J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994. doi:10.1006/inco.1994.1064.

13 David Eppstein, Michael T. Goodrich, and Songyu (Alfred) Liu. Bandwidth vs BFS Width in Matrix Reordering, Graph Reconstruction, and Graph Drawing. In *33rd Annual European Symposium on Algorithms (ESA 2025)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 69:1–69:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.69.

14 Uriel Feige. Coping with the NP-hardness of the graph bandwidth problem. In *Algorithm Theory—SWAT 2000*, volume 1851 of *Lecture Notes in Computer Science*, pages 10–19, Berlin, Heidelberg, 2000. Springer. doi:10.1007/3-540-44985-X_2.

15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

16 Jotun J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989. doi:10.1016/S0092-8240(89)80102-8.

17 Jeremy Hoskins, Cameron Musco, Christopher Musco, and Babis Tsourakakis. Inferring networks from random walk-based node similarities. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL: https://papers.nips.cc/paper/2018/hash/2f25f6e3.pdf.

18 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4), 2018. doi:10.1145/3199606.

19 Chirag Kaudan and Amir Nayyeri. Reconstructing bounded treelength graphs with linearithmic shortest path distance queries. In Shahin Kamali, editor, *Proceedings of the 37th Canadian Conference on Computational Geometry (CCCG 2025), York University, Toronto, Canada, August 13-15, 2025*, pages 243–248, 2025.

20 Michael Luttenberger. An extension of Parikh's theorem beyond idempotence. *CoRR*, abs/1112.2864, 2011. arXiv:1112.2864.

21 Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 733–744, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-39206-1_62.

22 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In Marcus Hutter, Rocco A. Servedio, and Eiji Takimoto, editors, *Algorithmic Learning Theory*, pages 285–297, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-75225-7_24.

23 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theoretical Computer Science*, 859:48–56, 2021. doi:10.1016/j.tcs.2021.01.006.

24 M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977. doi:10.1016/0022-5193(77)90351-4.

Parameterized Critical Node Cut Revisited

Dušan Knop
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Nikolaos Melissinos
Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Manolis Vasilakis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Abstract

We study how to sparsify connectivity in graphs under a tight deletion budget. Given a graph G and integers $k, x \geq 0$, Critical Node Cut (CNC) asks whether we can delete at most k vertices so that the number of remaining unordered pairs of connected vertices is at most x. CNC generalizes Vertex Cover (the case $x = 0$) and models tasks in network design, epidemiology, and social network analysis. We comprehensively map the structural parameterized complexity landscape for Critical Node Cut. First, we prove W[1]-hardness for the combined parameter $k + \text{fes} + \Delta + \text{pw}$, where fes is the feedback edge set number, Δ the maximum degree, and pw the pathwidth of the input graph, respectively. This significantly improves over the known W[1]-hardness for $k + \text{tw}$, where tw denotes the treewidth, and is tight in that tree-depth together with maximum degree trivially yields FPT. Second, we give new positive results. Specifically, we identify three structural parameters–max-leaf number, vertex integrity, and modular-width–that render the problem fixed-parameter tractable, and develop a polynomial-time algorithm for graphs of constant clique-width. Third, leveraging a technique introduced by Lampis [ICALP '14], we develop an FPT approximation scheme that, for any $\varepsilon > 0$, computes a $(1 + \varepsilon)$-approximate solution in time $(\text{tw}/\varepsilon)^{\mathcal{O}(\text{tw})} n^{\mathcal{O}(1)}$. Finally, we show that CNC admits no polynomial kernel when parameterized by vertex cover number, unless standard assumptions fail. Together, these results substantially sharpen the known complexity landscape for CNC.

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

Keywords and phrases Critical Node Cut, Parameterized Complexity, Treewidth

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.25

Related Version *Full Version*: https://arxiv.org/abs/2506.23363

Funding *Dušan Knop*: Supported by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590).
Nikolaos Melissinos: Partially supported by Charles University projects UNCE 24/SCI/008 and PRIMUS 24/SCI/012, and by the project 25-17221S of GAČR.
Manolis Vasilakis: Supported by the ANR project ANR-21-CE48-0022 (S-EX-AP-PE-AL) and the Barrande Fellowship programme.

1 Introduction

Let G be a simple, undirected, and unweighted graph, and let k and x be two non-negative integers. The Critical Node Cut problem (CNC for short) asks whether we can delete at most k vertices from G so that the remaining graph contains at most x connected vertex pairs. This problem arises in various applications, such as budgeted immunization for contagion control, analyzing brain connectivity, and uncovering structures in social graphs (see [4, 13] and references therein).

© Dušan Knop, Nikolaos Melissinos, and Manolis Vasilakis;
licensed under Creative Commons License CC-BY 4.0

Importantly, CNC generalizes the well-known Vertex Cover problem, which corresponds to the special case $x = 0$. Given its NP-hardness, we study CNC through the lens of parameterized complexity.[1] Previous work has shown that the problem is fixed-parameter tractable (FPT) when parameterized by the vertex cover number of the input graph [22], but W[1]-hard when parameterized by the natural parameter k or by the treewidth tw [13].[2] As a matter of fact, Agrawal, Lokshtanov, and Mouawad have shown that this hardness persists even when parameterizing by the combined parameter $k + \text{tw}$ [3].

Our Contribution. Our first result, which constitutes our main technical contribution, is to significantly improve over the aforementioned hardness result of Agrawal, Lokshtanov, and Mouawad [3]. In particular, in Theorem 2 we prove that CNC remains W[1]-hard even when parameterized by the combined parameter $k + \text{fes} + \Delta + \text{pw}$, where fes, Δ, and pw denote the feedback edge set number, the maximum degree, and the pathwidth of the input graph, respectively. Notice that this also implies that the problem is W[1]-hard when parameterized by $k + \text{ctw}$, where ctw denotes the cutwidth of the input graph, as for any graph it holds that $\text{ctw} \leq \text{pw} \cdot \Delta$. In fact, this intractability result is tight in the sense that tree-depth together with maximum degree trivially yields FPT, since this parameterization bounds the graph size.[3] To obtain our result, we adapt a recent reduction for the related Vertex Integrity problem by Hanaka, Lampis, Vasilakis, and Yoshiwatari [12].

Moving on, we obtain several results concerning the tractability of the problem under different structural parameterizations. We identify three structural parameters that render CNC fixed-parameter tractable, thereby contrasting the hardness result above: max-leaf number (Theorem 8), vertex integrity (Theorem 9), and modular-width (Theorem 10). Our techniques towards obtaining those results include reductions to acyclic instances [12], N-fold Integer Programming [17], and dynamic programming on suitable decompositions. We also consider the parameterization by clique-width cw, for which the problem is known to be W[1]-hard, and obtain an algorithm with running time $n^{\mathcal{O}(2^{\text{cw}})}$ (Theorem 11), thus placing it in XP.

We then revisit the parameterization by treewidth and identify the bottleneck in standard dynamic programming as the need to track large numerical values. To address this, we apply a rounding technique introduced by Lampis [20], obtaining an FPT approximation scheme. Specifically, in Theorem 15 we give an algorithm that, for any $\varepsilon > 0$, computes in $(\text{tw}/\varepsilon)^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$ time a deletion set of size at most k such that the number of connected pairs in the resulting graph is at most $(1 + \varepsilon)$ times the minimum possible.

Finally, we investigate the compressibility of CNC. We prove that, unless standard complexity assumptions fail, the problem does not admit a polynomial kernel when parameterized by the vertex cover number of the input graph (Theorem 16). This negative result highlights that even very restrictive structural parameterizations do not yield efficient preprocessing algorithms (under standard assumptions).

Related Work. CNC is a well-studied problem; see [18, Section 5.2] and the references therein. Approximation algorithms have been proposed [24], while Hermelin, Kaspi, Komusiewicz, and Navon [13] initiated the problem's study under parameterized complexity, presenting a plethora of both positive and negative results with respect to natural and structural parameterizations. Among others, they prove a (tight) $n^{o(k)}$ lower bound under

[1] We assume familiarity with the basics of parameterized complexity, as given e.g. in [7].
[2] More strongly, the reduction in [13] implies W[1]-hardness with respect to tree-depth.
[3] A graph of tree-depth td and maximum degree Δ has at most $\text{td} \cdot \Delta^{\text{td}}$ vertices.

the ETH, W[1]-hardness by tree-depth, and that the problem is FPT parameterized by $x + \mathrm{tw}$. Polynomial-time algorithms are known for trees [23],[4] and more generally, the problem admits a DP algorithm of running time $n^{\mathcal{O}(\mathrm{tw})}$ [2]. As previously mentioned, CNC is known to be W[1]-hard parameterized by $k + \mathrm{tw}$ [3]. On the other hand, it admits FPT algorithms with single-exponential parametric dependence when parameterized by vertex cover number [22] or neighborhood diversity [21].

2 Preliminaries

Throughout the paper we use standard graph notations [8] and assume familiarity with the basic notions of parameterized complexity [7]. Proofs of statements marked with (★) are presented in the full version of the paper.

We use $\mathbb{N}$ to denote the set of non-negative integers. For $x, y \in \mathbb{Z}$, let $[x, y] = \{z \in \mathbb{Z} \mid x \leq z \leq y\}$, while $[x] = [1, x]$. For a set of integers $S \subseteq \mathbb{N}$ let $\binom{S}{c}$ denote the set of subsets of S of size c, i.e., $\binom{S}{c} = \{S' \subseteq S : |S'| = c\}$. For a (multi)set of positive integers S, let $\Sigma(S)$ denote the sum of its elements.

All graphs considered are undirected without loops. Given a graph G, $\mathtt{cc}(G)$ denotes the set of its connected components, while $\mathrm{cp}(G)$ denotes the number of pairs of connected vertices in G, that is, $\mathrm{cp}(G) = \sum_{C \in \mathtt{cc}(G)} \binom{|V(C)|}{2}$. For a vertex $v \in V(G)$, we denote the neighborhood of v in G by $N_G(v)$. For a subset of vertices $S \subseteq V(G)$, $G[S]$ denotes the subgraph induced by S, while $G - S$ denotes $G[V(G) \setminus S]$.

For the basics on tree decompositions we refer to [7, Section 7]. We define the *height of a node of a tree decomposition* as the distance from the root of the tree decomposition to said node. In that case, the *height of a tree decomposition* is defined as the maximum distance from the root to any of its nodes.

A graph of clique-width k can be constructed through a sequence of the following operations on vertices that are labeled with at most k different labels. We can use (1) introducing a single vertex v of an arbitrary label i, denoted $i(v)$, (2) disjoint union of two labeled graphs, denoted $H_1 \oplus H_2$, (3) introducing edges between *all* pairs of vertices of two distinct labels i and j in a labeled graph H, denoted $\eta_{i,j}(H)$, and (4) changing the label of *all* vertices of a given label i in a labeled graph H to a different label j, denoted $\rho_{i \to j}(H)$. An expression describes a graph G if G is the final graph given by the expression (after we remove all the labels). The *width* of an expression is the number of different labels it uses. The clique-width of a graph is the minimum width of an expression describing it [6]. For a labeled graph H and $v \in V(H)$, let $\mathrm{lab}_H(v)$ denote the label of v in H, while $\mathrm{lab}_H^{-1}(i) = \{v \in V(H) \mid \mathrm{lab}_H(v) = i\}$ denotes the set of vertices of H of label i. A clique-width expression is *irredundant* if whenever the operation $\eta_{i,j}$ is applied on a graph G, there is no edge between an i-vertex and a j-vertex in G, and we will use such expressions to simplify our algorithms. We remark that any clique-width expression can be transformed in linear time into an irredundant one of the same width [6].

Critical Node Cut

Instance: A graph $G = (V, E)$ as well as integers k and x.

Goal: Determine whether there exists a set $S \subseteq V$ with $|S| \leq k$ such that in $G - S$ there are at most x pairs of connected vertices, that is, whether $\mathrm{cp}(G - S) \leq x$.

[4] That is the case even for weighted variants of the problem.

▶ **Lemma 1 (★).** *Let $k \in \mathbb{N}$. Assume we are given n functions $f_i \colon [0,k] \to \mathbb{N}$, where $i \in [n]$, and define $f \colon [0,k] \to \mathbb{N}$ such that for all $k' \in [0,k]$,*

$$f(k') = \min_{\substack{k_i \in [0,k'] \\ k_1 + \cdots + k_n = k'}} \sum_{i \in [n]} f_i(k_i).$$

One can compute the function f in time $\mathcal{O}(nk^2)$.

3 W[1]-hardness

In this section we prove that Critical Node Cut is W[1]-hard when parameterized by the combined parameter $k + \text{fes} + \Delta + \text{pw}$, where fes, Δ, and pw denote the feedback edge set number, the maximum degree, and the pathwidth of the input graph, respectively. This significantly improves over the previous result of Agrawal, Lokshtanov, and Mouawad [3] who showed that the problem is W[1]-hard parameterized by k plus the treewidth of the input graph. Our reduction follows along the lines of a recent reduction by Hanaka et al. [12] who showed the W[1]-hardness of Semi-Weighted Component Order Connectivity under a similar parameterization. We reduce from the Restricted Unary Bin Packing problem, which we formally define below and which is known to be W[1]-hard parameterized by the number of bins k [12].

Restricted Unary Bin Packing

Instance: A multiset $A = \{a_1, \ldots, a_n\}$ of integers in unary, $k \in \mathbb{N}$, and a function $f \colon A \to \binom{[k]}{2}$.

Goal: Determine whether there is a partition of A into multisets $\mathcal{A}_1, \ldots, \mathcal{A}_k$, such that for all $i \in [k]$ it holds that (i) $\Sigma(\mathcal{A}_i) = \Sigma(A)/k$, and (ii) $\forall a \in \mathcal{A}_i$, $i \in f(a)$.

We first present a sketch of our reduction. The *weight* of a vertex refers to the length of a path attached to it; this is later formalized in the construction. Similarly to the construction of Hanaka et al. [12], for every bin of the Restricted Unary Bin Packing problem instance we introduce a clique of $\mathcal{O}(k^2)$ heavy vertices, and then connect any pair of such cliques via two paths. The weights are set in such a way that an optimal solution will delete only vertices from the connection paths. In order to construct a path for a pair of bins, we compute the set of all subset sums of the items that can be placed in these two bins, and introduce a vertex of medium weight per such subset sum. This step can be completed in polynomial time, as all items are in unary. Moreover, every such vertex corresponding to subset sum s is preceded by exactly s vertices of weight 1. An optimal solution will cut the path so that the number of vertices of weight 1 will be partitioned between the two bins, encoding the subset sum of the elements placed in each bin. The second path that we introduce has balancing purposes, allowing us to exactly count the number of vertices of medium weight that every connected component will end up with. In the end, the intended solution deletes only vertices of medium weight, resulting in components of only two possible sizes.

▶ **Theorem 2.** Critical Node Cut *is W[1]-hard parameterized by $k + \text{fes} + \Delta + \text{pw}$.*

Proof. Let (A, k, f) be an instance of Restricted Unary Bin Packing, where $A = \{a_1, \ldots, a_n\}$ denotes the multiset of items given in unary, k is the number of bins, and $f \colon A \to \binom{[k]}{2}$ dictates the pair of bins an item may be placed into. In the following let

$B = \Sigma(A)/k$, $c = 6k^2$, $M = kB + 1$, $L = 36k^4BM$, and $T = L + (k-1) \cdot 2BM + B$. Notice that $M > kB$ and $L/c = 6k^2BM$. We will reduce (A, k, f) to an equivalent instance (G, k', x) of Critical Node Cut, where $k' = 2\binom{k}{2}$ and $x = k \cdot \binom{T}{2} + 2\binom{k}{2} \cdot \binom{M-1}{2}$ denote the size of the deletion set and the bound on the number of pairs of connected vertices, respectively.

We proceed to describe the construction of graph G. For ease of presentation, when we say that a vertex $v \in V(G)$ has weight $w \geq 1$, it means that we introduce a path on $w-1$ vertices and connect one of its endpoints to v.

For every $i \in [k]$, we introduce a clique on vertex set $\hat{C}_i$, which is comprised of c vertices, each of weight L/c. Fix i and j such that $1 \leq i < j \leq k$, and let $H_{i,j} = \{a \in A \mid f(a) = \{i, j\}\}$ denote the multiset of all items of A which can be placed either on bin i or bin j, where $\Sigma(H_{i,j}) \leq 2B$. Let $\mathcal{S}(H_{i,j}) = \{\Sigma(H) \mid H \subseteq H_{i,j}\}$ denote the set[5] of all subset sums of $H_{i,j}$, and notice that since every element of $H_{i,j}$ is encoded in unary, $\mathcal{S}(H_{i,j})$ can be computed in polynomial time using, e.g., Bellman's classical DP algorithm [5]. We next construct two paths connecting the vertices of $\hat{C}_i$ and $\hat{C}_j$. First, we introduce the vertex set $\hat{U}_{i,j} = \{v_q^{i,j} \mid q \in [0, 4B - |\mathcal{S}(H_{i,j})| + 1]\}$, all the vertices of which are of weight M. Add edges $(v_q^{i,j}, v_{q+1}^{i,j})$ for all $q \in [0, 4B - |\mathcal{S}(H_{i,j})|]$, as well as $(v_1, v_0^{i,j})$ and $(v_{4B-|\mathcal{S}(H_{i,j})|+1}^{i,j}, v_2)$, for all $v_1 \in \hat{C}_i$ and $v_2 \in \hat{C}_j$. Next, we introduce the vertex set $\hat{D}_{i,j} = \{s_q^{i,j} \mid q \in [\Sigma(H_{i,j})]\} \cup \{\sigma_q^{i,j} \mid q \in \mathcal{S}(H_{i,j})\}$, with the s-vertices being of weight 1 and the σ-vertices of weight M. Then, add the following edges:

- $(v_1, \sigma_0^{i,j})$ and $(\sigma_{\Sigma(H_{i,j})}^{i,j}, v_2)$, for all $v_1 \in \hat{C}_i$ and $v_2 \in \hat{C}_j$,
- for $q \in \mathcal{S}(H_{i,j})$, add the edges $(s_q^{i,j}, \sigma_q^{i,j})$ if $q \neq 0$ and $(\sigma_q^{i,j}, s_{q+1}^{i,j})$ if $q \neq \Sigma(H_{i,j})$, and
- for $q \in [\Sigma(H_{i,j})] \setminus \mathcal{S}(H_{i,j})$, add the edge $(s_q^{i,j}, s_{q+1}^{i,j})$.

Observe that if $H_{i,j} = \varnothing$, then $\Sigma(H_{i,j}) = 0$, $\mathcal{S}(H_{i,j}) = \{0\}$, and $|\mathcal{S}(H_{i,j})| = 1$, thus $\hat{U}_{i,j}$ consists of $4B + 1$ vertices of weight M, while $\hat{D}_{i,j}$ consists only of the single vertex $\sigma_0^{i,j}$ which is connected to the two cliques $\hat{C}_i, \hat{C}_j$.

This concludes the construction of G. See Figure 1 for an illustration. Notice that the number of vertices of G excluding those belonging to the cliques $\hat{C}_1, \ldots, \hat{C}_k$ or to a path attached to them is $(4B+2)M \cdot \binom{k}{2} + kB < 6k^2BM = L/c$.

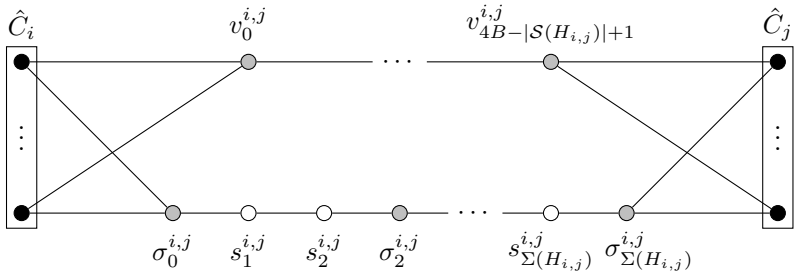


Figure 1 Rectangles denote cliques of size c. Here we assume that $1 \leq i < j \leq k$, $1 \notin \mathcal{S}(H_{i,j})$, and $2 \in \mathcal{S}(H_{i,j})$. The gray and black colors indicate weight of M and L/c, respectively.

▶ **Lemma 3 (★).** *If (A, k, f) is a Yes-instance of* Restricted Unary Bin Packing*, then (G, k', x) is a Yes-instance of* Critical Node Cut*.*

▶ **Lemma 4.** *If (G, k', x) is a Yes-instance of* Critical Node Cut*, then (A, k, f) is a Yes-instance of* Restricted Unary Bin Packing*.*

[5] We stress that $\mathcal{S}(H_{i,j})$ is a set and not a multiset.

Proof. Let $S \subseteq V(G)$ with $|S| \leq k' = 2\binom{k}{2}$ such that the number of pairs of connected vertices in $G - S$ is minimized and is at most x. Notice that $|S| = k'$, as otherwise there always exists a vertex whose deletion reduces said number. For all $i \in [k]$ it holds that $|\hat{C}_i| > k'$, thus the vertices of $\hat{C}_i \setminus S$ belong to the same connected component of $G - S$. Our aim is to argue that S contains exactly one vertex of medium weight per path between cliques. Towards this goal, we first show the following series of Claims.

▷ Claim 5. For any $i, j \in [k]$ with $1 \leq i < j \leq k$ it holds that $|S \cap \hat{U}_{i,j}| \leq 1$ and $|S \cap \hat{D}_{i,j}| \leq 1$.

Proof. For the sake of contradiction, assume there exist $i, j \in [k]$ with $1 \leq i < j \leq k$ such that $|S \cap \hat{Z}_{i,j}| \geq 2$, where $\hat{Z}_{i,j} \in \{\hat{U}_{i,j}, \hat{D}_{i,j}\}$. Let v denote the rightmost vertex of $\hat{Z}_{i,j}$ that belongs to S, that is, no vertex of $\hat{Z}_{i,j}$ between v and the vertices of $\hat{C}_j$ belongs to S. Let $\mathcal{C}_j$ denote the connected component of $G - S$ that contains the vertices of $\hat{C}_j$. It holds that $|V(\mathcal{C}_j)| \geq 2L/3$, since $c \geq 3k'$. Let $v' \in \hat{C}_j \setminus S$ and consider the deletion set $S' = (S \setminus \{v\}) \cup \{v'\}$, where $|S'| = |S|$.

Starting from $G - S$ it holds that not deleting v increases the number of connected pairs by $Y_1 \leq 4BM \cdot |V(\mathcal{C}_j)| + 4BM \cdot M = \frac{2L}{3k^2c} \cdot (|V(\mathcal{C}_j)| + M)$, while subsequently deleting v' decreases the number of pairs by $Y_2 \geq (|V(\mathcal{C}_j)| - L/c) \cdot (L/c)$. In that case, $\mathrm{cp}(G - S') = \mathrm{cp}(G-S)+Y_1-Y_2$. Recall that by the optimality of S, $\mathrm{cp}(G-S) \leq \mathrm{cp}(G-S') \iff Y_2 \leq Y_1$, implying that

$$
\begin{aligned}
(|V(\mathcal{C}_j)| - L/c) \cdot (L/c) \leq \frac{2L}{3k^2c} \cdot (|V(\mathcal{C}_j)| + M) &\iff \\
|V(\mathcal{C}_j)| - L/c \leq \frac{2}{3k^2} \cdot (|V(\mathcal{C}_j)| + M) &\iff \\
|V(\mathcal{C}_j)| \left(1 - \frac{2}{3k^2}\right) - L/c \leq \frac{2M}{3k^2} &\implies \qquad (\text{by } \frac{2M}{3k^2} < \frac{L}{c}) \\
|V(\mathcal{C}_j)| \left(1 - \frac{2}{3k^2}\right) - L/c < L/c &\implies \qquad (\text{by } |V(\mathcal{C}_j)| \geq 2L/3) \\
\frac{2L}{3} \cdot \left(1 - \frac{2}{3k^2}\right) < \frac{2L}{6k^2} &\iff \\
\left(1 - \frac{2}{3k^2}\right) < \frac{1}{2k^2} &\iff \\
1 \leq \frac{7}{6k^2}, &
\end{aligned}
$$

which, since $k \geq 2$, leads to a contradiction. ◁

We next argue that every connected component of $G - S$ contains vertices belonging to at most one clique $\hat{C}_i$.

▷ Claim 6. For any connected component $\mathcal{C}$ of $G - S$ it holds that

$$|\{i \in [k] \mid V(\mathcal{C}) \cap \hat{C}_i \neq \varnothing\}| \leq 1\,.$$

Proof. We first fix some notation used throughout the proof. Let $\Pi \subseteq 2^{[k]}$ be the partition of $[k]$ such that $p \in \Pi$ if and only if there exists a connected component $\mathcal{C}_p \in \mathtt{cc}(G - S)$ with $p = \{i \in [k] \mid V(\mathcal{C}_p) \cap \hat{C}_i \neq \varnothing\}$. Notice that for $p \neq \varnothing$, such a component is unique. For $p \in \Pi$ we further define the set $\hat{R}(p)$ which is composed of the vertices $\bigcup_{i \in p} \hat{C}_i \cup \bigcup_{i,j \in p} (\hat{U}_{i,j} \cup \hat{D}_{i,j})$ plus the vertices belonging to the paths attached to the former vertices. Let $g\colon \mathtt{cc}(G - S) \to [0, k]$ such that $g(\mathcal{C}) = |\{i \in [k] \mid V(\mathcal{C}) \cap \hat{C}_i \neq \varnothing\}|$ for

all $\mathcal{C} \in \mathtt{cc}(G-S)$. Consider a component $\mathcal{C}_{p_{\max}} \in \mathtt{cc}(G-S)$ that maximizes g, where $g(\mathcal{C}_{p_{\max}}) = |p_{\max}| \geq 1$. It suffices to prove that $|p_{\max}| < 2$. Towards a contradiction assume that $|p_{\max}| \geq 2$.

Let $\mathcal{C}_{p'}$ with $g(\mathcal{C}_{p'}) = |p'| < |p_{\max}|$. We argue that $S \cap \hat{R}(p') = \varnothing$. To see this, notice that the deletion of such a vertex results in at most

$$\frac{L}{c} \cdot \left(|p'| \cdot L + \frac{L}{c} \right)$$

disconnected pairs, where we use the fact that the number of vertices of G excluding those belonging to the cliques $\hat{C}_1, \ldots, \hat{C}_k$ or to a path attached to them is less than L/c. On the other hand, $|V(\mathcal{C}_{p_{\max}})| \geq |p_{\max}| \cdot L - k' \cdot (L/c) \geq (|p_{\max}| - 1) \cdot L + (2L)/3$, thus the deletion of a vertex $v \in \hat{C}_i \setminus S$ with $i \in p_{\max}$ results in at least

$$\frac{L}{c} \cdot \left((|p_{\max}| - 1) \cdot L + \frac{2L}{3} \right) > \frac{L}{c} \cdot \left(|p'| \cdot L + \frac{L}{c} \right)$$

disconnected pairs.

The previous paragraph, along with Claim 5 and the fact that $|S| = 2\binom{k}{2}$, imply that there exists $p \in \Pi$ with $|p| = |p_{\max}|$ such that $|S \cap \hat{R}(p)| \geq 2\binom{|p|}{2}$. Let $S' = (S \setminus \hat{R}(p)) \cup \{v_0^{i,j}, \sigma_0^{i,j} \mid i,j \in p\}$, where $|S'| \leq |S|$. We argue that $\mathrm{cp}(G-S') < \mathrm{cp}(G-S)$. Let x_1 (resp., x_2) denote the number of pairs of connected vertices in $G-S$ (resp., $G-S'$) such that at least one vertex belongs to $\hat{R}(p)$ (notice that the rest of the pairs remain the same in the two graphs). It holds that

$$\begin{aligned} 2 \cdot x_1 &> 2 \cdot \binom{(|p|-1) \cdot L + 2L/3}{2} \\ &\geq ((|p|-1)L + 2L/3) \cdot ((|p|-1)L + L/3) \\ &= L^2(|p|^2 - |p| + 2/9), \end{aligned}$$

where the first inequality is due to the component of $G-S$ that contains the vertices of $\{\hat{C}_i \mid i \in p\} \setminus S$. On the other hand, in $G-S'$ the vertices of each $\hat{C}_i$ for $i \in p$ are in distinct connected components, thus it holds that

$$\begin{aligned} 2 \cdot x_2 &< 2|p| \cdot \binom{L + L/c}{2} \\ &\leq |p|(L + L/c)^2 \\ &= L^2\big(|p| + |p| \cdot \frac{2c+1}{c^2}\big) \\ &\leq L^2(|p| + 2/9), \end{aligned}$$

where the first inequality is due to the fact that the number of vertices of G apart from those of $\bigcup_{i \in [k]} \hat{C}_i$ or attached to them is at most L/c. The last inequality is due to the fact that $|p| \cdot \frac{2c+1}{c^2} \leq k \cdot \frac{12k^2+1}{36k^4}$, which for $k \geq 2$ is upper-bounded by $2/9$.

Finally, we have that

$$\begin{aligned} & & x_1 > x_2 \iff \\ L^2(|p|^2 - |p| + 2/9) \geq L^2(|p| + 2/9) & \iff \\ & & |p|^2 \geq 2|p|, \end{aligned}$$

which holds as by assumption $|p| \geq 2$. This contradicts the optimality of set S. ◁

By Claim 6 and the fact that $k' = 2\binom{k}{2}$ it follows that S contains exactly one vertex per path between cliques. In that case, since the vertices of those paths are of weight 1 or M, regarding the connected components of $G - S$ it holds that there are exactly k of size at least L, as well as $a \in [\binom{k}{2}, 2\binom{k}{2}]$ of size exactly $M - 1$, where $a \geq \binom{k}{2}$ since half the paths between cliques contain only vertices of weight M. Further notice that the number of pairs of connected vertices is minimized when each component of size at least L contains $\frac{|V(G)|-a\cdot M}{k}$ vertices,[6] while for all $a \leq 2\binom{k}{2}$ it holds that

$$k\cdot\binom{(|V(G)|-a\cdot M)/k}{2}+a\cdot\binom{M-1}{2} > k\cdot\binom{(|V(G)|-(a+1)\cdot M)/k}{2}+(a+1)\cdot\binom{M-1}{2}.$$

Consequently, the minimum number of pairs of connected vertices is obtained exactly when there are $2\binom{k}{2}$ components in $G - S$ of size $M - 1$, and the rest of the vertices are partitioned equally among the remaining k components. Notice that

$$\begin{aligned}|V(G)| &= k \cdot L + (4B+2)M \cdot \binom{k}{2} + kB \\ &= k \cdot (L + 2BM(k-1) + B) + 2M \cdot \binom{k}{2} \\ &= k \cdot T + 2\binom{k}{2} \cdot M,\end{aligned}$$

thus since $x = k \cdot \binom{T}{2} + 2\binom{k}{2} \cdot \binom{M-1}{2}$ it follows that this is indeed the case for $G - S$. Consequently, for all $1 \leq i < j \leq k$, there exists $\sigma_q^{i,j} \in S$ with $q \in \mathcal{S}(H_{i,j})$.

Let $\mathcal{C}_i$ denote the connected component of $G - S$ containing the vertices of $\hat{C}_i$. Since $\mathcal{C}_i$ contains T vertices, while $kB < M$, it follows that $\mathcal{C}_i$ contains exactly B vertices of weight 1. Let $(\mathcal{A}_1, \ldots, \mathcal{A}_k)$ be a partition of A defined in the following way: for all $1 \leq i < j \leq k$, if $\sigma_q^{i,j} \in S$, then $\mathcal{A}_i \cap H_{i,j} = \mathcal{H}_{i,j}$ and $\mathcal{A}_j \cap H_{i,j} = H_{i,j} \setminus \mathcal{H}_{i,j}$, where $\mathcal{H}_{i,j} \subseteq H_{i,j}$ such that $\Sigma(\mathcal{H}_{i,j}) = q$. Notice that $\Sigma(\mathcal{A}_i)$ is equal to the number of vertices of weight 1 in $\mathcal{C}_i$, therefore $\Sigma(\mathcal{A}_i) = B$ follows. ◀

▶ **Lemma 7.** *It holds that* $\mathrm{fes}(G) = \mathcal{O}(k^5)$, $\Delta(G) = \mathcal{O}(k^2)$, *and* $\mathrm{pw}(G) = \mathcal{O}(k^3)$.

Proof. For the feedback edge set number, let $F \subseteq E(G)$ contain all edges between vertices of $\hat{C}_i$, for all $i \in [k]$, as well as all edges between such vertices and endpoints of paths $\hat{U}_{i,j}$ and $\hat{D}_{i,j}$. Notice that

$$|F| = k \cdot \binom{c}{2} + 4\binom{k}{2} \cdot c = \mathcal{O}(k^5),$$

while the graph remaining after the deletion of the edges in F is a caterpillar forest. For the pathwidth bound notice that $G - \bigcup_{i=1}^{k} \hat{C}_i$ is a caterpillar forest, thus $\mathrm{pw}(G) = \mathcal{O}(k^3)$. Lastly, for the maximum degree notice that for all $v \in \bigcup_{i\in[k]} \hat{C}_i$, $|N(v)| = c + 2(k-1) = \mathcal{O}(k^2)$. As for the vertices of $\hat{D}_{i,j} \cup \hat{U}_{i,j}$, they are of degree at most 3, apart from the endpoints which are neighbors with all the vertices of a single clique, therefore of degree $\mathcal{O}(k^2)$. Any remaining vertex is of degree at most 2. ◀

Due to Lemmas 3, 4, and 7, Theorem 2 follows. ◀

[6] To see this notice that $\binom{n-\varepsilon}{2} + \binom{n+\varepsilon}{2} > 2 \cdot \binom{n}{2}$ for all $\varepsilon > 0$.

4 Algorithms

In this section we identify several structural parameters that render Critical Node Cut fixed-parameter tractable, namely max-leaf number, vertex integrity, and modular-width. Furthermore, we show that the problem parameterized by clique-width belongs to XP.

4.1 Max-Leaf Number

Here we present a single-exponential algorithm for Critical Node Cut parameterized by the max-leaf number of the input graph.[7] This is a well-studied but very restricted parameter [9, 10, 19]. Note that this nicely complements the W[1]-hardness of the problem parameterized by $\mathrm{fes} + \Delta$ which follows by Theorem 2; graphs of max-leaf number at most k are known to be a subdivision of a graph of at most $\mathcal{O}(k)$ vertices [16], thus their feedback edge set number and their maximum degree are bounded by a function of k. As a matter of fact, the reduction of Theorem 2 produces a graph that, at first glance, appears to have bounded max-leaf number. The value of this parameter increases due to attaching long paths on the weighted vertices, and Theorem 8 implies that this increase is unavoidable, as the problem is FPT under this parameterization.

Our algorithm closely follows the one presented by Hanaka, Lampis, Vasilakis, and Yoshiwatari [12] for Vertex Integrity parameterized by max-leaf number. In particular, our first step is to contract any cycles, reducing the instance to a weighted forest on which some of the vertices are marked as undeletable. Then we apply a simple DP algorithm for trees in order to compute, for every connected component of the instance, the minimum number of pairs of connected vertices remaining after at most b vertex deletions from said component, for all $b \in [0, k]$. Having computed this for every component, applying Lemma 1 then allows us to determine the number of deletions per component such that the total number of connected pairs is minimized.

▶ **Theorem 8 (★).** *There is an algorithm that given any instance $\mathcal{I} = (G, k, x)$ of* Critical Node Cut*, decides $\mathcal{I}$ in time $2^{\mathcal{O}(\mathrm{ml})} n^{\mathcal{O}(1)}$, where* ml *denotes the max-leaf number of G.*

4.2 Vertex Integrity

The *vertex integrity* of a graph $G = (V, E)$, denoted $\mathrm{vi}(G)$, is the minimum integer k such that there is a vertex set $U \subseteq V$ with $|U| + \max_{C \in \mathtt{cc}(G-U)} |V(C)| \leq k$, where $\mathtt{cc}(G - U)$ denotes the set of connected components of $G - U$. Here we show that the parameterization by vertex integrity renders Critical Node Cut fixed-parameter tractable. Given the fact that the vertex integrity of a graph is upper-bounded by its vertex cover number, this improves over the known FPT algorithm parameterized by vertex cover number [22]. On the other hand, the vertex integrity of a graph is lower-bounded by its tree-depth, under which parameterization the problem remains W[1]-hard [13], so our result is in that sense tight.

▶ **Theorem 9 (★).** *There is an algorithm that, given any instance $\mathcal{I} = (G, k, x)$ of* Critical Node Cut*, decides $\mathcal{I}$ in time $\mathrm{vi}^{\mathcal{O}(\mathrm{vi}^2)} n^{\mathcal{O}(1)}$, where* vi *denotes the vertex integrity of G.*

[7] The *max-leaf number* of a graph is the maximum number of leaves among all its spanning trees.

4.3 Modular-width

All our previous results are concerned with *sparse* classes of graphs. Here we prove that Critical Node Cut is fixed-parameter tractable parameterized by the modular-width of the input graph, thus improving over the analogous result for neighborhood diversity [21].

▶ **Theorem 10 (★).** *There is an algorithm that, given any instance $\mathcal{I} = (G, k, x)$ of* Critical Node Cut*, decides $\mathcal{I}$ in time $2^{\mathrm{mw}}n^{\mathcal{O}(1)}$, where* mw *denotes the modular-width of G.*

4.4 Clique-width

Theorem 2 implies that Critical Node Cut is W[1]-hard parameterized by clique-width. Here we show that for this parameterization, the problem belongs to XP. To obtain this result, we proceed by presenting a standard dynamic programming algorithm. On a high level, as we go over the clique-width expression of the input graph, we consider a partition of the connected components of our graph based on the labels appearing in their vertices. For every such partitioning, we keep track of *both* the total size as well as the total number of connected pairs of vertices for the connected components belonging to the partitioning. As we prove, this amount of information is sufficient in order to solve Critical Node Cut.

▶ **Theorem 11.** *There is an algorithm that, given a graph G along with an irredundant clique-width expression ψ of G of width* cw*, determines for all $k \in [0, n]$ the minimum value of* $\mathrm{cp}(G - S)$ *over all subsets $S \subseteq V(G)$ with $|S| = k$, as well as the number of subsets achieving this minimum, in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$.*

Proof. Before describing our algorithm we start with some definitions and notations. Let $\mathcal{H}$ denote the set of all cw-labeled graphs generated by subexpressions of ψ. Furthermore, let $\mathcal{L} = 2^{[\mathrm{cw}]} \setminus \varnothing$ denote the set of all non-empty subsets of [cw]. We say that $\sigma = (\alpha, \beta)$ is a *signature*, where σ is a tuple consisting of functions $\alpha, \beta \colon \mathcal{L} \to \mathbb{N}$. Moreover, let Σ denote the set of all possible signatures, where $|\Sigma| = n^{\mathcal{O}(2^{\mathrm{cw}})}$. We say that the *label set* of a connected component of a labeled graph is the set of labels appearing in the component's vertices. For $H \in \mathcal{H}$ and $L \in \mathcal{L}$, let $\mathcal{C}_H^L \subseteq \mathtt{cc}(H)$ denote the set of connected components of H with label set L. Lastly, for $H \in \mathcal{H}$, we define the function $\mathrm{sgn}_H \colon 2^{V(H)} \to \Sigma$, such that for $S \subseteq V(H)$, the *H-signature of S* is the tuple $\mathrm{sgn}_H(S) = \sigma = (\alpha, \beta) \in \Sigma$, where for all $L \in \mathcal{L}$ it holds that

- $\alpha(L)$ is equal to the sum of the sizes of the connected components of $\mathcal{C}_{H-S}^L$,
- $\beta(L)$ is equal to the number of pairs of connected vertices belonging to connected components of $\mathcal{C}_{H-S}^L$.

Our algorithm proceeds by dynamic programming along ψ in a bottom-up fashion. In particular, for every $H \in \mathcal{H}$ it stores a table $\mathrm{DP}_H[\cdot, \cdot]$ where, as we show, for $k \in [0, n]$ and $\sigma \in \Sigma$, $\mathrm{DP}_H[k, \sigma]$ is equal to the number of subsets of $V(H)$ of size k and H-signature σ. We now proceed to describe how to populate the DP tables, as well as establish this invariant by induction.

Singleton $H = i(v)$. For $H = i(v)$, notice that $\mathrm{lab}_H^{-1}(i) = \{v\}$ and $\mathrm{lab}_H^{-1}(w) = \varnothing$ for all $w \in [\mathrm{cw}] \setminus \{i\}$. Consequently, for $k \in [0, n]$ and signature $\sigma = (\alpha, \beta) \in \Sigma$ we set

$$\mathrm{DP}_H[k, \sigma] = \begin{cases} 1 & \text{if } k = 0,\ \alpha \equiv \alpha_0[\{i\} \mapsto 1], \text{ and } \beta \equiv \beta_0, \\ 1 & \text{if } k = 1,\ \alpha \equiv \alpha_0, \text{ and } \beta \equiv \beta_0, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where for all $L \in \mathcal{L}$, $\alpha_0(L) = \beta_0(L) = 0$. By Equation (1) we can fill the table $\mathrm{DP}_H[\cdot,\cdot]$ in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$. Furthermore, notice that $2^{V(H)} = \{\varnothing, \{v\}\}$ and one can easily verify that the invariant holds.

Disjoint union, $H = H_1 \oplus H_2$. We define a function $h\colon \Sigma \times \Sigma \to \Sigma$ such that, for $\sigma_1 = (\alpha_1, \beta_1)$ and $\sigma_2 = (\alpha_2, \beta_2)$, $h(\sigma_1, \sigma_2) = \sigma = (\alpha, \beta)$ where for all $L \in \mathcal{L}$ it holds that

- $\alpha(L) = \alpha_1(L) + \alpha_2(L)$,
- $\beta(L) = \beta_1(L) + \beta_2(L)$.

For a fixed size $k \in [0, n]$ and signature $\sigma \in \Sigma$, we set the value of $\mathrm{DP}_H[k, \sigma]$ to be

$$\mathrm{DP}_H[k,\sigma] = \sum_{k_1+k_2=k} \sum_{(\sigma_1,\sigma_2)\in h^{-1}(\sigma)} \mathrm{DP}_{H_1}[k_1,\sigma_1] \cdot \mathrm{DP}_{H_2}[k_2,\sigma_2]. \tag{2}$$

If $h^{-1}(\sigma) = \varnothing$ we set $\mathrm{DP}_H[k, \sigma] = 0$. Notice that computing $h^{-1}(\sigma)$ requires $n^{\mathcal{O}(2^{\mathrm{cw}})}$ time, thus by Equation (2) we can fill the table $\mathrm{DP}_H[\cdot,\cdot]$ in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$.

▷ Claim 12. Let $H = H_1 \oplus H_2$. Assume that for $H_i \in \{H_1, H_2\}$, $k \in [0, n]$, and $\sigma \in \Sigma$, $\mathrm{DP}_{H_i}[k, \sigma]$ is equal to the number of subsets of $V(H_i)$ of size k and H_i-signature σ. Then it holds that $\mathrm{DP}_H[k, \sigma]$ is equal to the number of subsets of $V(H)$ of size k and H-signature σ.

Proof. Observe that $V(H) = V(H_1) \cup V(H_2)$ and $E(H) = E(H_1) \cup E(H_2)$. Thus, for every $S \subseteq V(H)$, it holds that $S = S_1 \cup S_2$ where $S_i = S \cap V(H_i)$ for $i \in \{1, 2\}$. Furthermore, notice that the connected components of $H - S$ are exactly the connected components of $H_1 - S_1$ and $H_2 - S_2$, each with the same label set. Consequently, for every $L \in \mathcal{L}$ it holds that

- $\alpha(L) = \alpha_1(L) + \alpha_2(L)$,
- $\beta(L) = \beta_1(L) + \beta_2(L)$,

where $\sigma = (\alpha, \beta)$ is the H-signature of S, and $\sigma_i = (\alpha_i, \beta_i)$ is the H_i-signature of S_i for $i \in \{1, 2\}$. This implies that $\sigma = h(\sigma_1, \sigma_2)$. The claim follows by Equation (2) and by the fact that there are $\mathrm{DP}_{H_1}[k_1, \sigma_1]$ ways to choose a subset of size k_1 with H_1-signature σ_1, and $\mathrm{DP}_{H_2}[k_2, \sigma_2]$ ways to choose a subset of size k_2 with H_2-signature σ_2, for every pair $(\sigma_1, \sigma_2) \in h^{-1}(\sigma)$ and every pair (k_1, k_2) such that $k = k_1 + k_2$. ◁

Relabeling, $H = \rho_{i\to j}(H')$. Observe that a connected component with label set $L \in \mathcal{L}$ in H has label set either L, $L \cup \{i\}$, or $L \cup \{i\} \setminus \{j\}$ in H'. For each pair of distinct $i, j \in [\mathrm{cw}]$ we define a function $f_{i\to j}\colon \Sigma \to \Sigma$ such that, for $\sigma' = (\alpha', \beta')$, $f_{i\to j}(\sigma') = \sigma = (\alpha, \beta)$ where for all $L \in \mathcal{L}$ it holds that

- if $i \in L$, then $\alpha(L) = \beta(L) = 0$,
- if $i, j \notin L$, then $\alpha(L) = \alpha'(L)$ and $\beta(L) = \beta'(L)$,
- if $i \notin L$ and $j \in L$, then it holds that
  - $\alpha(L) = \alpha'(L) + \alpha'(L \cup \{i\}) + \alpha'(L \cup \{i\} \setminus \{j\})$,
  - $\beta(L) = \beta'(L) + \beta'(L \cup \{i\}) + \beta'(L \cup \{i\} \setminus \{j\})$.

For a fixed size $k \in [0, n]$ and signature $\sigma \in \Sigma$, we set the value of $\mathrm{DP}_H[k, \sigma]$ to be

$$\mathrm{DP}_H[k,\sigma] = \sum_{\sigma' \in f_{i\to j}^{-1}(\sigma)} \mathrm{DP}_{H'}[k,\sigma']. \tag{3}$$

If $f_{i\to j}^{-1}(\sigma) = \varnothing$ we set $\mathrm{DP}_H[k, \sigma] = 0$. Notice that computing $f_{i\to j}^{-1}(\sigma)$ requires $n^{\mathcal{O}(2^{\mathrm{cw}})}$ time, thus by Equation (3) we can fill the table $\mathrm{DP}_H[\cdot,\cdot]$ in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$.

▷ Claim 13. Let $H = \rho_{i\to j}(H')$. Assume that for $k \in [0,n]$ and $\sigma \in \Sigma$, $\mathrm{DP}_{H'}[k,\sigma]$ is equal to the number of subsets of $V(H')$ of size k and H'-signature σ. Then it holds that $\mathrm{DP}_H[k,\sigma]$ is equal to the number of subsets of $V(H)$ of size k and H-signature σ.

Proof. Observe that $V(H) = V(H')$ and $E(H) = E(H')$. Furthermore, notice that for all $S \subseteq V(H)$, the connected components of $H - S$ are exactly the connected components of $H' - S$, albeit with potentially different label sets. Let $S \subseteq V(H)$ such that $\mathrm{sgn}_H(S) = \sigma = (\alpha,\beta)$ and $\mathrm{sgn}_{H'}(S) = \sigma' = (\alpha',\beta')$. We argue that $f_{i\to j}(\sigma') = \sigma$. Given that each subset S has a unique H'-signature, Equation (3) and the fact that there are $\mathrm{DP}_{H'}[k,\sigma']$ ways to choose a subset of size k with H'-signature σ', this implies the claim.

Let $L \in \mathcal{L}$. If $i \in L$, then since $\mathrm{lab}_H^{-1}(i) = \varnothing$, it holds that no connected component of $H - S$ has label set L, thus $\alpha(L) = \beta(L) = 0$. If $i,j \notin L$, then the connected components of $H - S$ with label set L are exactly the connected components of $H' - S$ with label set L, thus $\alpha(L) = \alpha'(L)$ and $\beta(L) = \beta'(L)$. Lastly, if $i \notin L$ and $j \in L$, then the connected components of $H - S$ with label set L are exactly the connected components of $H' - S$ with label set L, $L \cup \{i\}$, and $L \cup \{i\} \setminus \{j\}$. Thus, $\alpha(L) = \alpha'(L) + \alpha'(L \cup \{i\}) + \alpha'(L \cup \{i\} \setminus \{j\})$ and $\beta(L) = \beta'(L) + \beta'(L \cup \{i\}) + \beta'(L \cup \{i\} \setminus \{j\})$. It follows that $f_{i\to j}(\sigma') = \sigma$, and this concludes the proof. ◁

Joining labels with edges, $H = \eta_{i,j}(H')$. For each pair of distinct $i,j \in [\mathrm{cw}]$ we define a function $g_{i,j} \colon \Sigma \to \Sigma$ as follows. Let for $\sigma' = (\alpha',\beta')$, $g_{i,j}(\sigma') = \sigma = (\alpha,\beta)$. We consider two cases. First, assume that either $\sum_{L\in\mathcal{L}_i} \alpha'(L) = 0$ or $\sum_{L\in\mathcal{L}_j} \alpha'(L) = 0$, where for $z \in \{i,j\}$, $\mathcal{L}_z = \{L \in \mathcal{L} \mid z \in L\}$ denotes the label sets containing label z. In that case, we set $\sigma = \sigma'$. Otherwise, let $\mathcal{L}_{i,j}^{\sigma'} = \{L \in \mathcal{L} \mid \{i,j\} \cap L \neq \varnothing \text{ and } \alpha'(L) \neq 0\}$ and $L_{i,j}^{\sigma'} = \bigcup_{L\in\mathcal{L}_{i,j}^{\sigma'}} L$. For all $L \in \mathcal{L}$ we set the values of $\alpha(L)$ and $\beta(L)$ as follows:

- if $i,j \notin L$, then $\alpha(L) = \alpha'(L)$ and $\beta(L) = \beta'(L)$,
- if $\{i,j\} \cap L \neq \varnothing$ and $L \neq L_{i,j}^{\sigma'}$, then $\alpha(L) = \beta(L) = 0$,
- if $L = L_{i,j}^{\sigma'}$, then $\alpha(L) = \sum_{L\in\mathcal{L}_{i,j}^{\sigma'}} \alpha'(L)$ and $\beta(L) = \binom{\alpha(L)}{2}$.

For a fixed size $k \in [0,n]$ and signature $\sigma \in \Sigma$, we set the value of $\mathrm{DP}_H[k,\sigma]$ to be

$$\mathrm{DP}_H[k,\sigma] = \sum_{\sigma' \in g_{i,j}^{-1}(\sigma)} \mathrm{DP}_{H'}[k,\sigma']. \tag{4}$$

If $g_{i,j}^{-1}(\sigma) = \varnothing$ we set $\mathrm{DP}_H[k,\sigma] = 0$. Notice that computing $g_{i,j}^{-1}(\sigma)$ requires $n^{\mathcal{O}(2^{\mathrm{cw}})}$ time, thus by Equation (4) we can fill the table $\mathrm{DP}_H[\cdot,\cdot]$ in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$.

▷ Claim 14. Let $H = \eta_{i,j}(H')$. Assume that for $k \in [0,n]$ and $\sigma \in \Sigma$, $\mathrm{DP}_{H'}[k,\sigma]$ is equal to the number of subsets of $V(H')$ of size k and H'-signature σ. Then it holds that $\mathrm{DP}_H[k,\sigma]$ is equal to the number of subsets of $V(H)$ of size k and H-signature σ.

Proof. Observe that $V(H) = V(H')$, each vertex has the same label in both H and H', and $E(H) = E(H') \cup \{uv \mid u \in \mathrm{lab}_{H'}^{-1}(i), v \in \mathrm{lab}_{H'}^{-1}(j)\}$. Let $S \subseteq V(H)$ such that $\mathrm{sgn}_H(S) = \sigma = (\alpha,\beta)$ and $\mathrm{sgn}_{H'}(S) = \sigma' = (\alpha',\beta')$. We argue that $g_{i,j}(\sigma') = \sigma$. Given that each subset S has a unique H'-signature, Equation (4) and the fact that there are $\mathrm{DP}_{H'}[k,\sigma']$ ways to choose a subset of size k with H'-signature σ', this implies the claim. Let $L \in \mathcal{L}$. We will consider multiple cases.

First, assume that either $\sum_{L\in\mathcal{L}_i} \alpha'(L) = 0$ or $\sum_{L\in\mathcal{L}_j} \alpha'(L) = 0$, where for $z \in \{i,j\}$, $\mathcal{L}_z = \{L \in \mathcal{L} \mid z \in L\}$ denotes the label sets containing label z. In that case, it follows that either $\mathrm{lab}_{H'}^{-1}(i) \subseteq S$ or $\mathrm{lab}_{H'}^{-1}(j) \subseteq S$. Consequently, no new edges are added in $H - S$ compared to $H' - S$, thus, for all $L \in \mathcal{L}$, it holds that $\alpha(L) = \alpha'(L)$ and $\beta(L) = \beta'(L)$, which implies that $\sigma = \sigma'$.

Alternatively it holds that $\sum_{L\in\mathcal{L}_i}\alpha'(L)\neq 0$ and $\sum_{L\in\mathcal{L}_j}\alpha'(L)\neq 0$. If $i,j\notin L$, then the connected components of $H-S$ with label set L are exactly the connected components of $H'-S$ with label set L, thus $\alpha(L)=\alpha'(L)$ and $\beta(L)=\beta'(L)$ follow. To handle the remaining cases, we let $\mathcal{L}^{\sigma'}_{i,j}=\{L\in\mathcal{L}\mid\{i,j\}\cap L\neq\varnothing \text{ and } \alpha'(L)\neq 0\}$ and $L^{\sigma'}_{i,j}=\bigcup_{L\in\mathcal{L}^{\sigma'}_{i,j}}L$. Since there are vertices u and v in $H'-S$ with labels i and j, respectively, it follows that in $H-S$, all vertices of label i and all vertices of label j belong to the same connected component. Furthermore, the label set of that connected component is exactly $L^{\sigma'}_{i,j}$, as it contains all labels of connected components of $H'-S$ with label set in $\mathcal{L}^{\sigma'}_{i,j}$. As for its size, it holds that $\alpha(L^{\sigma'}_{i,j})=\sum_{L\in\mathcal{L}^{\sigma'}_{i,j}}\alpha'(L)$, from which we can infer the number of pairs of connected vertices $\beta(L^{\sigma'}_{i,j})=\binom{\alpha(L^{\sigma'}_{i,j})}{2}$. For any other label set $L\in\mathcal{L}$ with $\{i,j\}\cap L\neq\varnothing$ and $L\neq L^{\sigma'}_{i,j}$, it holds that $\alpha(L)=\beta(L)=0$ as either (i) $\alpha'(L)=\beta'(L)=0$, or (ii) the connected components of $\mathcal{C}^L_{H'-S}$ are all connected in $H-S$ and are part of the single connected component of label set $L^{\sigma'}_{i,j}$ in $H-S$. It follows that $g_{i,j}(\sigma')=\sigma$, and this concludes the proof. ◁

Correctness follows by induction and Claims 12–14. As for the running time, notice that for every $H\in\mathcal{H}$ the table $\mathrm{DP}_H[\cdot,\cdot]$ is filled in time $n^{\mathcal{O}(2^{\mathrm{cw}})}$. Since $|\mathcal{H}|=n^{\mathcal{O}(1)}$, the overall running time of our algorithm is $n^{\mathcal{O}(2^{\mathrm{cw}})}$. Finally, notice that for any $S\subseteq V(G)$ of size $k\in[0,n]$ and G-signature $\sigma=(\alpha,\beta)$, it holds that $\mathrm{cp}(G-S)=\sum_{L\in\mathcal{L}}\beta(L)$. Consequently, by iterating over all signatures $\sigma\in\Sigma$, we can determine the minimum value of $\mathrm{cp}(G-S)$ over all subsets $S\subseteq V(G)$ of size k, as well as the number of such subsets achieving this minimum. This concludes the proof. ◀

5 FPT Approximation Scheme

Given the fact that, as evidenced by Theorem 2, Critical Node Cut remains W[1]-hard even under severe structural parameterizations, in this section we aim to bypass this computational hardness by adding approximation into the mix. In particular, we design an efficient FPT-AS for the parameterization by treewidth by modifying the standard $n^{\mathcal{O}(\mathrm{tw})}$ DP algorithm [2] and making use of a technique introduced by Lampis [20].

▶ **Theorem 15 (★).** *There is an algorithm which, for all $\varepsilon>0$, when given as input a graph G of treewidth* tw *returns in time $(\mathrm{tw}/\varepsilon)^{\mathcal{O}(\mathrm{tw})}n^{\mathcal{O}(1)}$ a set $S\subseteq V(G)$ of size at most k such that $\mathrm{cp}(G-S)\leq(1+\varepsilon)\cdot\mathrm{cp}(G-S')$ for all $S'\subseteq V(G)$ of size $|S'|\leq k$, for all $k\leq n$.*

Proof sketch. Here we describe the main idea behind our algorithm. On a high level, we aim to develop a DP which, while traversing the tree decomposition, keeps track of the sizes of any *active* components (those whose vertices intersect the bag), while for the rest of the components (i.e., the *inactive* ones) there exists a variable on which we account for their number of pairs of connected vertices. To this end, assuming that the exact size of an active component is c, our DP stores a value $\hat{c}\leq(1+\varepsilon')\cdot c$ where ε' is such that $\binom{\hat{c}}{2}\leq(1+\varepsilon)\cdot\binom{c}{2}$. Notice that for this to hold, $c=1$ implies that $\hat{c}=1$. Consequently, it suffices to present a dynamic program which correctly stores the size of any singleton active component, while for the rest of active components it allows for $(1+\varepsilon')$-approximate values on their sizes. In that case, the number of pairs of connected vertices accounted for every connected component is a $(1+\varepsilon)$-approximation, and since we sum over those, the final value has a $(1+\varepsilon)$-approximation ratio as well. Our algorithm is thus a DP that does exactly as required. ◀

6 Kernelization

Given that, as we have shown in Theorem 9, Critical Node Cut is FPT parameterized by the vertex integrity of the input graph, a natural question arising is whether one can develop a polynomial kernel under this parameterization. In this section we prove that this cannot be the case under standard assumptions, even for the much more restricted parameterization by vertex cover number.

▶ **Theorem 16.** Critical Node Cut *does not admit a polynomial kernel when parameterized by the vertex cover number of the input graph, unless* NP $\subseteq$ coNP/poly.

Proof. We present a polynomial parameter transformation reducing from k-Multicolored Clique. In the latter, we are given a graph $G = (V, E)$ and a partition of V into k independent sets (also called color classes) $V_1, \ldots, V_k$, each of size n, and we are asked to determine whether G contains a k-clique. It is known that k-Multicolored Clique parameterized by $k \log n$ does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly [14].

Construction. Consider an instance (G, k) of k-Multicolored Clique, where $V_i = \{v_j^i \mid j \in [n]\}$ for all $i \in [k]$. By adding isolated vertices to the color classes if necessary, we may assume that n is a power of two. For each color class V_i, we assign to each of its vertices a unique (among the vertices of V_i) bit-string of length $\log n$. For all $w \in [\log n]$, let $b_w \colon V \to \{0, 1\}$ denote the w-th bit in the bit-string of $v_j^i \in V_i$. Construct the graph G' as follows.

- We first introduce a clique on vertex set $K = \{g_z^{i,w} \mid i \in [k], w \in [\log n], z \in \{0, 1\}\}$, which we refer to as *core vertices*. We say that a core vertex $g_z^{i,w}$ *corresponds* to color class i, with K_i being composed of all core vertices corresponding to color class i, for $i \in [k]$. Furthermore, we say that a vertex $v \in V_i$ of G is *encoded* by the vertices $\{g_{b_w(v)}^{i,w} \mid w \in [\log n]\}$.
- Introduce a clique on the vertex set $C = \{c_i \mid i \in [k \log n + 1]\}$, and add edges such that every vertex of C is adjacent to all the core vertices.
- For every edge $e = \{u_1, u_2\} \in E(G)$, where $u_1 \in V_{i_1}$ and $u_2 \in V_{i_2}$, introduce an *adjacency vertex* h_e which is adjacent to all vertices encoding its endpoints, that is, with vertices $g_{b_w(u_1)}^{i_1,w}$ and $g_{b_w(u_2)}^{i_2,w}$ for all $w \in [\log n]$. Let H_{i_1,i_2} denote the set of all adjacency vertices due to edges in G between vertices in V_{i_1} and V_{i_2}.
- For all $\{i_1, i_2\} \in \binom{[k]}{2}$ (that is, for all pairs of different color classes), and for all $w_1, w_2 \in [\log n]$ and $z_1, z_2 \in \{0, 1\}$, introduce an independent set of size $A = |E| + 1$, each vertex of which is incident with $g_{z_1}^{i_1,w_1}$ and $g_{z_2}^{i_2,w_2}$. We refer to the vertices added in this step of the construction as *dummy vertices*.

This concludes the construction of the graph G'. Notice that $G' - (K \cup C)$ is an independent set, consequently the vertex cover number of G' is at most $3k \log n + 1$. We will show that (G', k', x) is an equivalent instance of Critical Node Cut, where $k' = k \log n$ and $x = \binom{|V(G')| - k' - \binom{k}{2} - A\binom{k}{2}\log^2 n}{2}$.

For the forward direction, consider a function $s \colon [k] \to [n]$ such that $\mathcal{V} = \{v_{s(i)}^i \mid i \in [k]\}$ is a k-clique in G. In that case, let $S = \{g_{b_w(v_{s(i)}^i)}^{i,w} \mid i \in [k], w \in [\log n]\}$ be a set of size k'. We will prove that $G' - S$ has at most x pairs of connected vertices. First, notice that for every pair of core vertices belonging to S that correspond to different color classes, their removal results in an independent set of dummy vertices of size A in $G' - S$. Since there are $\binom{k}{2}\log^2 n$ such pairs, $G' - S$ contains $A\binom{k}{2}\log^2 n$ isolated dummy vertices. Furthermore, we argue that the adjacency vertex of any edge in $G[\mathcal{V}]$ is isolated in $G' - S$. To see this,

consider the adjacency vertex $h_e \in V(G')$ where $e = \{v^{i_1}_{s(i_1)}, v^{i_2}_{s(i_2)}\} \in E(G)$. It holds that $N_{G'}(h_e) = \{g^{i_1,w}_{b_w(v^{i_1}_{s(i_1)})}, g^{i_2,w}_{b_w(v^{i_2}_{s(i_2)})} \mid w \in [\log n]\} \subseteq S$, thus h_e is indeed an isolated vertex in $G' - S$. Since $\mathcal{V}$ induces a k-clique, there are $\binom{k}{2}$ such isolated adjacency vertices. Finally, S itself is of size k'. Consequently, the number of pairs of connected vertices in $G' - S$ is at most the number of pairs of non-isolated vertices in the graph, which is at most $\binom{|V(G')| - A\binom{k}{2}\log^2 n - \binom{k}{2} - k'}{2} = x$.

For the opposite direction, let $S \subseteq V(G')$ be of size at most k' such that $G' - S$ has at most x pairs of connected vertices. Notice that $|C| > k'$, consequently there exists a vertex $c \in C \setminus S$. Since $N_{G'}(c) \supseteq K$ and C is a clique, it follows that all vertices of $(K \cup C) \setminus S$ are in the same connected component of $G' - S$; let this component be denoted by $D_{\mathsf{big}} \in \mathsf{cc}(G' - S)$. Furthermore, since for any vertex $v \in V(G') \setminus (K \cup C)$ it holds that $N_{G'}(v) \subseteq K$, any such vertex either belongs to D_{big} or is isolated in $G' - S$.

▷ Claim 17. We have $|S \cap K_i| = \log n$, for all $i \in [k]$.

Proof. To prove the claim we argue about the number of isolated dummy vertices in $G' - S$. First, notice that $(n - \varepsilon)(n + \varepsilon) = n^2 - \varepsilon^2 < n^2$ for all $\varepsilon > 0$. Consequently, the number of isolated dummy vertices in $G' - S$ is maximized when $|S \cap K_i| = \log n$ for all $i \in [k]$, in which case the number of isolated dummy vertices is $A\binom{k}{2}\log^2 n$.

Assume that the claim is false. In that case, the number of isolated dummy vertices in $G' - S$ is at most $A\left(\binom{k}{2}\log^2 n - 1\right)$; any other dummy vertex in $G' - S$ must belong to D_{big}, which also contains all vertices of $(K \cup C) \setminus S$. Consequently, the size of $D_{\mathsf{big}} \in \mathsf{cc}(G' - S)$ is at least $|V(G')| - |E| - k' - A\left(\binom{k}{2}\log^2 n - 1\right)$. In that case

$$|V(G')| - |E| - k' - A\left(\binom{k}{2}\log^2 n - 1\right) > |V(G')| - A\binom{k}{2}\log^2 n - \binom{k}{2} - k' \iff$$
$$-|E| + A > -\binom{k}{2} \impliedby$$
$$A = |E| + 1,$$

thereby yielding a contradiction as $G' - S$ has more than x pairs of connected vertices. ◁

Recall that $G' - S$ is composed of a connected component D_{big} containing all non-isolated vertices, as well as some isolated vertices. By Claim 17 it holds that $|S| = k'$, thus for $G' - S$ to have at most x pairs of connected vertices the number of its isolated vertices must be at least $\binom{k}{2} + A\binom{k}{2}\log^2 n$. Due to Claim 17 it follows that it has exactly $A\binom{k}{2}\log^2 n$ isolated dummy vertices, while none of the remaining vertices of $K \cup C$ can be isolated. Consequently, the deletion of S isolates at least $\binom{k}{2}$ adjacency vertices.

We argue that no two isolated adjacency vertices h_{e_1}, h_{e_2} belong to the same set H_{i_1,i_2}. Assume that this is the case, and let $h_{e_1}, h_{e_2} \in H_{i_1,i_2}$. For $u \in \{h_{e_1}, h_{e_2}\}$ it holds that $|N_{G'}(u) \cap K_{i_1}| = |N_{G'}(u) \cap K_{i_2}| = \log n$, while $N_{G'}(h_{e_1}) \neq N_{G'}(h_{e_2})$. Due to Claim 17, this leads to a contradiction. Consequently, there is exactly one isolated adjacency vertex in $G' - S$ belonging to H_{i_1,i_2} for all $\{i_1, i_2\} \in \binom{[k]}{2}$.

Now consider one such isolated adjacency vertex $h_e \in H_{i_1,i_2}$. Notice that $|N_{G'}(h_e) \cap \{g^{p,w}_0, g^{p,w}_1\}| = 1$ for all $w \in [\log n]$ and $p \in \{i_1, i_2\}$. Since $S \supseteq N_{G'}(h_e)$, and this holds for all isolated adjacency vertices, due to Claim 17 it follows that $|S \cap \{g^{i,w}_0, g^{i,w}_1\}| = 1$ for all $w \in [\log n]$ and $i \in [k]$.

Let $s\colon [k] \to [n]$ such that $v^i_{s(i)}$ is encoded by the vertices $S \cap K_i$, for all $i \in [k]$. We claim that $\mathcal{V} = \{v^i_{s(i)} \mid i \in [k]\}$ induces a k-clique in G. Consider $\{i_1, i_2\} \in \binom{[k]}{2}$. Notice that there exists an isolated adjacency vertex h_e in $G' - S$ belonging to H_{i_1,i_2}; the neighborhood of h_e is exactly the vertices encoding its two endpoints in G, thus $\{v^{i_1}_{s(i_1)}, v^{i_2}_{s(i_2)}\} \in E(G)$. ◀

7 Conclusion

In this paper we have thoroughly studied Critical Node Cut from the perspective of parameterized complexity, and presented a plethora of results, mostly taking into account the structure of the input graph. As a direction of future work, it is currently unknown whether the problem is FPT when parameterized by $k + \mathrm{td}$, where td denotes the tree-depth of the input graph, or when parameterized by the cluster vertex deletion number of the input graph. Another interesting direction would be to (dis)prove the optimality of the $n^{\mathcal{O}(\mathrm{tw})}$ or the $n^{\mathcal{O}(2^{\mathrm{cw}})}$ algorithm; especially for the latter, we note that there are only a handful of natural problems for which such a running time is known to be optimal under the ETH [1, 11, 15]. Given the similarity of the construction of Theorem 2 and the algorithm of Theorem 8 with the corresponding results for Vertex Integrity [12], a natural question is whether we can obtain similar results for other optimization functions that take into account the sizes of the components of the graph remaining after the vertex deletions. Some of our results seem to be easily adaptable to such a more general setting (e.g., for general separately convex functions), however we do not know whether that is the case for the W[1]-hardness as well.

References

1 Pierre Aboulker, Édouard Bonnet, Eun Jung Kim, and Florian Sikora. Grundy coloring and friends, half-graphs, bicliques. *Algorithmica*, 85(1):1–28, 2023. doi:10.1007/S00453-022-01001-2.

2 Bernardetta Addis, Marco Di Summa, and Andrea Grosso. Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth. *Discret. Appl. Math.*, 161(16-17):2349–2360, 2013. doi:10.1016/J.DAM.2013.03.021.

3 Akanksha Agrawal, Daniel Lokshtanov, and Amer E. Mouawad. Critical node cut parameterized by treewidth and solution size is W[1]-hard. In *Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, WG 2017*, volume 10520 of *Lecture Notes in Computer Science*, pages 32–44. Springer, 2017. doi:10.1007/978-3-319-68705-6_3.

4 Ashwin Arulselvan, Clayton W. Commander, Lily Elefteriadou, and Panos M. Pardalos. Detecting critical nodes in sparse graphs. *Comput. Oper. Res.*, 36(7):2193–2200, 2009. doi:10.1016/J.COR.2008.08.016.

5 Richard E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957.

6 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

7 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.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.

9 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.

10 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009. doi:10.1007/S00224-009-9167-9.

11 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

12 Tesshu Hanaka, Michael Lampis, Manolis Vasilakis, and Kanae Yoshiwatari. Parameterized vertex integrity revisited. In *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024*, volume 306 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.MFCS.2024.58.

13 Danny Hermelin, Moshe Kaspi, Christian Komusiewicz, and Barak Navon. Parameterized complexity of critical node cuts. *Theor. Comput. Sci.*, 651:62–75, 2016. doi:10.1016/J.TCS.2016.08.018.

14 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/S00453-014-9910-8.

15 Lars Jaffke, Paloma T. Lima, and Daniel Lokshtanov. b-coloring parameterized by clique-width. *Theory Comput. Syst.*, 68(4):1049–1081, 2024. doi:10.1007/S00224-023-10132-0.

16 Daniel J. Kleitman and Douglas B. West. Spanning trees with many leaves. *SIAM J. Discret. Math.*, 4(1):99–106, 1991. doi:10.1137/0404010.

17 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.85.

18 Mohammed Lalou, Mohammed Amin Tahraoui, and Hamamache Kheddouci. The critical node detection problem in networks: A survey. *Comput. Sci. Rev.*, 28:92–117, 2018. doi:10.1016/J.COSREV.2018.02.002.

19 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

20 Michael Lampis. Parameterized approximation schemes using graph widths. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

21 Jannik Schestag. Critical node problem with vulnerable vertices. Master's thesis, Philipps-Universität Marburg, 2021.

22 Jannik Schestag, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. On critical node problems with vulnerable vertices. *J. Graph Algorithms Appl.*, 28(1):1–26, 2024. doi:10.7155/JGAA.V28I1.2922.

23 Marco Di Summa, Andrea Grosso, and Marco Locatelli. Complexity of the critical node problem over trees. *Comput. Oper. Res.*, 38(12):1766–1774, 2011. doi:10.1016/J.COR.2011.02.016.

24 Mario Ventresca and Dionne M. Aleman. A derandomized approximation algorithm for the critical node detection problem. *Comput. Oper. Res.*, 43:261–270, 2014. doi:10.1016/J.COR.2013.09.012.

On the Parameterized Complexity of Min-Sum-Radii

Pankaj Kumar
University of Birmingham, UK

Haiko Müller
University of Leeds, UK

Sebastian Ordyniak
University of Leeds, UK

Melanie Schmidt
Heinrich Heine University Düsseldorf, Germany

Abstract

In the Min-Sum-Radii (MSR) clustering problem, we are given a finite set X of n points in a metric space. The objective is to find at most k clusters centered at a subset of these points such that every point of X is assigned to one of the clusters, minimizing the sum of the radii of the clusters. The problem is known to be NP-hard even on metrics induced by weighted planar graphs and metrics with constant doubling dimension, as shown by Gibson et al. (SWAT 2008). In this work, we investigate the parameterized complexity of MSR on metrics induced by undirected graphs. We distinguish between weighted graph metrics (with positive edge weights) and unweighted graph metrics (where all edges have unit weight).

Weighted Graph Metrics. We show that MSR is W[1]-hard on metrics induced by weighted bipartite graphs, when parameterized by the combined parameter k the number of clusters and Δ the cost of the clustering. We then investigate the structural parameterized complexity of the problem. Drexler et al. [doi:10.48550/arXiv.2310.02130] showed that the MSR problem admits an XP algorithm on metrics induced by weighted graphs when parameterized by treewidth, and asked whether this can be improved to fixed-parameter tractability. We first answer their question in the negative, and more strongly show that MSR stays W[1]-hard on metrics induced by undirected weighted bipartite graphs when parameterized by the vertex cover number plus k. We then turn our attention to parameters for dense graphs and show that MSR remains W[1]-hard when parameterized by $k + \Delta$ even on cliques and complete bipartite graphs.

On the positive side, we employ the known XP algorithm parameterized by treewidth, to show that the MSR problem is FPT when parameterized by the parameter treewidth plus Δ. Together, these results provide a complete picture of the parameterized complexity of MSR with respect to any combination of parameters k, Δ, as well as structural parameters for sparse graphs above vertex cover and known parameters for dense graphs (such as neighborhood diversity and modular width).

Unweighted Graph Metrics. The story is rather different for unweighted graphs, since it is a long standing open question whether MSR on metrics induced by undirected graphs is solvable in polynomial-time. Although we cannot answer this question, we provide classical and parameterized hardness results for two very closely related problems, namely Exact-MSR (MSR and one wants to find exactly k clusters) and Allowed-Centers-MSR (MSR with an additional set of allowed cluster centers). We also show that MSR as well as these two problems are fixed-parameter tractable parameterized by the treedepth of the input graph.

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

Keywords and phrases Parameterized complexity, Min-Sum-Radii clustering

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.26

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2605.22804

© Pankaj Kumar, Haiko Müller, Sebastian Ordyniak, and Melanie Schmidt;
licensed under Creative Commons License CC-BY 4.0

Funding *Melanie Schmidt*: Melanie Schmidt acknowledges funding by DFG grant 456558332.

Acknowledgements We want to thank the reviewers of SWAT 2026 for their helpful suggestions. One of the authors thanks Tanmay Inamdar for helpful discussions on the problem and for pointing out relevant references.

1 Introduction

Clustering problems with centers are well studied in the fields of operation research, machine learning, data processing, theoretical computer science, and various other disciplines. Given a point-set X consisting of n points, a distance metric d defined on X, and a parameter k, the goal is to partition X into k clusters $C_1, C_2, \ldots, C_k$ with respective centers $c_1, c_2, \ldots, c_k$. The k-Center, k-Median, and k-Means problems are notably the most widely studied clustering problems of this type: [9, 19, 2, 20, 1]. The k-Center objective is to find a clustering such that the maximum distance between the center of a cluster and any point in the cluster, i.e., the maximum radius of a cluster, is minimized. The k-Median objective minimizes the total distance, which can average out larger individual costs, potentially overlooking the needs of outlier points.

The MSR problem offers a compromise by targeting to minimize the sum of the radii of the clusters. We define the MSR problem formally as follows.

Min-Sum-Radii (MSR)

Instance:	A finite set X of n points in a metric space (X, d), an integer k, and a number $\Delta \in \mathbb{R}$.
Question:	Are there center-radius pairs $(c_1, r_1), \ldots, (c_t, r_t)$ with $t \leq k$, $c_i \in X$, and $r_i \geq 0$ such that every point $x \in X$ is of distance at most r_i to some center c_i (i.e., $d(x, c_i) \leq r_i$) and $\sum_{i=1}^{t} r_i \leq \Delta$.

This approach allows for more balanced cluster formation, fine-tuning the radii across all clusters and ensuring that the maximum individual cost is kept within reasonable bounds, although it may be higher than in k-Center – potentially by a factor of k – though this is not always the case. This intermediate objective is particularly useful in applications where balancing the load among clusters is crucial, such as in wireless network design where the goal is to minimize the energy required for transmission, which is proportional to the sum of the radii of the coverage areas of the base stations, see Lev-Tov and Peleg [24].

Charikar and Panigrahy [9] studied the MSR problem and noted that it effectively addresses the "dissection effect" inherent in the k-center approach, where the uniform maximum radius assumption can lead to significant overlap among clusters, resulting in points that should belong together being split across different clusters. By minimizing the sum of the radii, MSR reduces this dissection effect, promoting more coherent and practical clustering solutions.

Gibson et al. [17] showed that the MSR problem is NP-hard, even when restricted to planar metric or metrics with constant doubling dimension. For general metrics, they presented a randomized algorithm for MSR with a runtime of $n^{O(\log n \log D)}$ where D represents the ratio between the maximum and minimum distances. They also obtained a $(1+\epsilon)$-approximation in quasi-polynomial time. It follows that under standard complexity-theoretic assumptions, the problem cannot be APX-hard. Interestingly, Gibson et al. [18] presented a polynomial time algorithm for MSR in constant-dimensional Euclidean metrics. Charikar and Panigrahy [9] developed a $O(1)$-approximation algorithm for the MSR problem (as well as the k-Min-Sum-Diameter problem) on general metrics using the primal-dual framework introduced by

Jain and Vazirani for the k-Median problem. This was further refined by Friggstad and Jamshidian [16], who achieved a 3.389-approximation for MSR. This was recently improved to $3+\varepsilon$, which is currently the best-known approximation factor for the general case [7].

The MSR problem has been studied in constrained settings, such as with lower bounds, outliers, and capacities. A significant line of work focuses on its parameterized complexity. Several recent results [22, 3, 23, 10, 8, 4] discuss FPT-approximation algorithms for the problem and its variants; see the related work section.

We investigate the MSR problem under the *graph metrics*. For an undirected graph $G=(V,E)$ with positive (or unit) edge weights, we denote by $X=V$ the set of input points, and by $d(u,v)$, the length of a shortest u–v path in G. The metric space induced by the graph G is (X,d).

MSR on Graph Metrics	
Instance:	An undirected graph $G=(V,E)$ with positive edge weights, an integer k, and a number $\Delta\in\mathbb{R}$. The input metric d is the graph metric induced by G.
Parameter:	k (the (maximum) number of allowed clusters)
Question:	Are there center–radius pairs $(c_1,r_1),\ldots,(c_t,r_t)$ with $t\leq k$, $c_i\in V$, and $r_i\geq 0$ such that every vertex $v\in V$ is of distance at most r_i from some center c_i (i.e., $d(v,c_i)\leq r_i$) and $\sum_{i=1}^{t} r_i\leq\Delta$.

Throughout this paper, we only consider undirected graphs, and all of our results are stated for undirected graph metrics. We distinguish between weighted graph metrics (with positive edge weights) and (unweighted) graph metrics (where all edges have unit weight).

Weighted Graph Metrics. A naive brute-force algorithm yields an XP algorithm parameterized by k: one can guess the k centers (at most $\binom{n}{k}$ choices) and then determine the optimal radii (for each radius there are at most $n-1$ possible choices–as there are at most $n-1$ possible r-neigborhoods, i.e., neighborhoods around a vertex with radius r), leading to a running time of $n^{\mathcal{O}(k)}$. To the best of our knowledge, the question of whether MSR admits a true FPT algorithm in this setting–an algorithm running in time $f(k)\cdot n^{O(1)}$ for some computable function f–has remained open.

We resolve this by proving that MSR is W[1]-hard parameterized by $k+\Delta$, even on weighted bipartite graphs. Furthermore, we establish a lower bound under the Exponential Time Hypothesis (ETH).

Table 1 Summary of complexity results for MSR on weighted graph metrics. A checkmark (✓) indicates that the parameter in that column is used for the corresponding result. k: number of clusters; Δ: cost of optimal clustering; *vc*: vertex cover number; *tw*: treewidth. Note that these results provide a comprehensive picture for the parameterized complexity of MSR with respect to any combination of these 4 parameters.

k	Δ	vc	tw	Complexity	Reference
✓	✓			W[1]	Theorem 1
✓		✓		W[1]	Theorem 2
✓				XP	naive brute force
			✓	XP	Drexler et al. [14]
	✓		✓	FPT	Corollary 9

▶ **Theorem 1** ($\star$). *MSR on metrics induced by weighted bipartite graphs is* W[1]-hard *parameterized by the number of clusters k plus the cost of the clustering Δ. Moreover, under the Exponential Time Hypothesis, there is no $f(k+\Delta) \cdot N^{o(k+\log_2 \Delta)}$-time algorithm, where N is the number of input points, that solves MSR for metrics induced by weighted bipartite graphs, for any computable function f.*

We note that when the inter-point distances are bounded by a polynomial in the input-size, the algorithm of Gibson et al. [17] admits randomized *quasi-polynomial* time. At this point, we mention that our W[1]-hardness proof also implies NP-hardness of the decision version of the problem and justifies the exponential edge-weights in the reduction of Theorem 1.

We then explore the parameterized complexity of MSR with respect to structural parameters. Drexler et al. [14] showed that the MSR problem admits an XP algorithm on metrics induced by bounded treewidth graphs, and asked the question whether the problem admits an FPT algorithm when parameterized by the treewidth tw. We first answer their question negatively, and more strongly show that MSR problem stays W[1]-hard on metrics induced by weighted graphs when parameterized by vc, the vertex cover number, plus k (even on bipartite graphs).

▶ **Theorem 2.** *The MSR problem is* W[1]-hard *on metrics induced by weighted bipartite graphs when parameterized by the vertex cover number (vc) plus the number of clusters (k). Moreover, there is no $f(\mathsf{vc}+k) \cdot N^{o(\mathsf{vc}+k)}$-time algorithm for any computable function f unless* ETH *fails, where N is the number of input points.*

We then, using the XP algorithm of Drexler et al.[14], show that the MSR problem is FPT when parameterized by the combined parameter tw, the treewidth and Δ, the cost of clustering (Corollary 9).

Taken together, these results already provide a rather comprehensice picture of the parameterized complexity of MSR on metrics induced by weighted graphs for combinations of the parameters k, Δ and any sparse structural parameter above and including the vertex cover number (please refer to Table 1 for an overview of our results). We complete the picture for dense structural parameters (such as neighborhood diversity and modular width) by showing that MSR is already NP-hard and W[1]-hard parameterized by $k+\Delta$ on complete as well as complete bipartite graphs. On the positive side, we obtain fixed-parameter tractability when parameterizing by tw plus Δ (Corollary 9).

(Unweighted) Graph Metrics. We now consider the case of metrics induced by (unweighted) graphs. Note that as the randomized algorithm of Gibson et al. [17], solves MSR on undirected graphs in quasi polynomial time (as the maximum interpoint distance is at most $n-1$), MSR on these metrics should not be NP-hard, unless NP is contained in randomized quasipolynomial time–an unlikely complexity theoretic consequence. However, it is a long standing open question whether MSR on undirected graph metrics, is solvable in polynomial-time; this is only known for the version of MSR without zero clusters (i.e., no singleton clusters are allowed) [5].

Although we cannot answer this question, we provide the following main contributions for these metrics. First, we exhibit two natural variants of MSR, namely, Exact-MSR and Allowed-Centers-MSR, and provide strong hardness results for these variants even on metrics induced by undirected graphs. In the Exact-MSR version, it is required to find a solution with exactly k non-zero clusters. We formally define it as follows.

EXACT-MSR

INSTANCE:	An undirected graph $G = (V, E)$, an integer k, and a number $\Delta \in \mathbb{R}$. The input metric (X, d) is the graph metric induced by G.
PARAMETER:	k (the number of clusters)
QUESTION:	Is there a set $C = \{(c_1, r_1), \ldots, (c_k, r_k)\}$ of exactly k center-radius pairs (c_i, r_i) such that: ▬ $c_i \in X$ and there are at least two points in X with distance at most r_i from c_i, ▬ every point in X is within the distance r_i from c_i for at least one $i \in [k]$, ▬ $\sum_{i \in [k]} r_i \leq \Delta$.

We show that EXACT-MSR in NP-hard and W[1]-hard parameterized by $k + \Delta$ and provide a lower bound under ETH (Theorem 12). The exact-k constraint and prohibition of singleton clusters in EXACT-MSR are not artificial but reflect real-world needs in network partitioning, resource allocation, and community detection. In such settings, singleton clusters are undesirable for robustness and minimum-load considerations, and a fixed number of clusters is required for budget or regulatory constraints.

In the ALLOWED-CENTERS-MSR version, one is additionally required to choose the cluster centers from a given set of allowed centers.

ALLOWED-CENTERS-MSR

INSTANCE:	An undirected graph $G = (V, E)$, an integer k, a set of allowed centers $A \subseteq V$, and a number $\Delta \in RR$. The input metric (X, d) is the graph metric induced by G.
PARAMETER:	k
QUESTION:	Is there a set $C = \{(c_1, r_1), \ldots, (c_t, r_t)\}$ of $t \leq k$ center-radius pairs (c_i, r_i) such that: ▬ $c_i \in A$ and $r_i \geq 0$ is a number, ▬ every point in X is within distance r_i from c_i for at least one $i \in [t]$, ▬ $\sum_{i \in [t]} r_i \leq \Delta$.

The restriction to allowed centers in ALLOWED-CENTERS-MSR models practical settings, where the centers can only be selected from a predefined set of feasible locations. We show that ALLOWED-CENTERS-MSR is W[1]-hard parameterized by either $k + \Delta$ (Theorem 13) or $k + |A| + \mathsf{fvs}$ (Theorem 14), where A is the set of allowed centers and fvs is the *feedback vertex set number* of the underlying undirected graph. It is important to note that we are not able to show NP-hardness for ALLOWED-CENTERS-MSR and in fact we see this as a way around first having to settle the classical complexity of MSR. That is, to obtain a parameterized complexity classification, it is sufficient to show W[1]-hardness, which as our result for ALLOWED-CENTERS-MSR shows is an easier target. Along with these results, we also show that MSR as well as EXACT-MSR and ALLOWED-CENTERS-MSR are fixed-parameter tractable parameterized by the *treedepth* of the underlying graph (Corollary 11).

Remark. To comply with page limits, proofs of statements marked with [⋆] are provided in the appendix; the main text remains self-contained.

2 Preliminaries

We use the standard graph-theoretic notation as in Diestel [12]. We consider graphs that are undirected, connected, finite and simple. We denote by $G = (V, E)$ a graph with vertex set $V(G)$ and edge set $E(G)$. We denote the edge between u and v by $\{u, v\}$. We use

n to denote the number of vertices of G. For a set $X \subseteq V$, the graph $G[X]$ denotes the induced subgraph of G on the vertex set X. The open neighborhood of a vertex v, denoted by $N(v)$, is the set of vertices adjacent to v and the closed neighborhood of v is denoted by $N[v] = N(v) \cup \{v\}$. We write $[n] = \{1, 2, \ldots, n\}$ for a positive integer n. A graph G is a *clique* if for every pair of vertices u and v there is an edge $\{u, v\}$ in $E(G)$. A *path* P on the $\ell \leq |V(G)|$ vertices in G is a sequence of vertices $P = (v_1, v_2, \ldots, v_\ell)$ such that $\{v_{i-1}, v_i\} \in E$ for every $i \in \{2, \ldots, \ell\}$. If G is an edge-weighted graph with weight function $w : E \to \mathbb{R}_{\geq 0}$, the *weight* of P on ℓ vertices, is defined as $w(P) = \sum_{i=2}^{\ell} w(\{v_{i-1}, v_i\})$. If G is unweighted, the *weight* (or *length*) of P on ℓ vertices is simply the number of edges in P, that is, $w(P) = \ell - 1$. For two vertices $u, v \in V(G)$, the *distance* between u and v, denoted by $d(u, v)$, is defined as $d(u, v) = \min\{w(P) \mid P \text{ is a } u\text{–}v \text{ path in } G\}$. If no u–v path exists, we define $d(u, v) = \infty$.

Parameterized Complexity. In Parameterized Complexity theory, the computational complexity of a problem is measured as a function of the input size $|I|$ and a non-negative integer parameter k associated with the input. We say a parameterized problem is *fixed-parameter tractable* (FPT) with respect to a parameter k, if there exists an algorithm that runs in time $f(k) \cdot |I|^{O(1)}$ where f is a computable function independent of the input size $|I|$ and k is a parameter associated with the input instance.

Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from P to Q is an algorithm $\mathcal{A}$ that, given an instance (I, k) of P, outputs an instance (I', k') of Q such that (I, k) is a YES instance of P if and only if (I', k') is a YES instance of Q, $k' \leq g(k)$ for some computable function $g\colon \mathbb{N} \to \mathbb{N}$, and the running time of $\mathcal{A}$ is $f(k) \cdot |I|^{O(1)}$ for some computable function f. The hardness theory in parameterized complexity is developed via the notion of parameterized reductions and a hierarchy of complexity classes forming the W-hierarchy: FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ XP.

In a q-CNF-SAT instance, we are given a q-CNF formula Φ, and the question is to decide whether Φ is satisfiable. [21], in their seminal work, formulated the following hypothesis, called Exponential Time Hypothesis(ETH) which states that q-CNF-SAT, $q \geq 3$ cannot be solved within a running time of $2^{o(p)}$ or $2^{o(m)}$, where p is the number of variables and m is the number of clauses in the input q-CNF formula. The Exponential Time Hypothesis (ETH) is known to imply FPT $\neq$ W[1]. We rely on the definitions of the structural graph parameters such as vertex cover number, treedepth, feedback vertex set, treewidth, cliquewidth, neighborhood diversity, modular width, and modular treewidth etc., for more details on the subject, the reader is deferred to [11].

3 MSR on metrics induced by weighted graphs

In this section, we present our results for MSR on metrics induced by weighted graphs. In particular, we show in Section 3.1 that MSR on metrics induced by weighted bipartite graphs is W[1]-hard parameterized by $k + \Delta$. We then continue in Section 3.2 to show that the same result applies when parameterized by $k + \mathsf{vc}$, where vc is the vertex cover number of the graph. Interestingly, the latter result even applies for the variant of MSR, where the set of centers is given and one merely needs to find the right radius for every center, i.e., the hardness of the problem lies in finding the right radius for every center. We then complete the picture in Section 3.3 by presenting further hardness results for dense graph parameters and an algorithm using the parameter treewidth. Together, those results (which are also given in Table 1 apart from the results for dense graph parameters) provide a comprehensive

picture of the parameterized complexity of MSR with respect to any combination of the parameters k, Δ, structural parameters less restrictive than vertex cover, and common structural parameters for dense graphs (such as neighborhood diversity and modular width).

3.1 Parameterized by $k + \Delta$

Here, we show that MSR on metrics induced by weighted bipartite graphs is W[1]-hard parameterized by $k + \Delta$.

▶ **Theorem 1** ($\star$). *MSR on metrics induced by weighted bipartite graphs is* W[1]-hard *parameterized by the number of clusters k plus the cost of the clustering Δ. Moreover, under the Exponential Time Hypothesis, there is no $f(k+\Delta) \cdot N^{o(k+\log_2 \Delta)}$-time algorithm, where N is the number of input points, that solves MSR for metrics induced by weighted bipartite graphs, for any computable function f.*

We prove the theorem by giving a parameterized reduction from the well-known W[1]-hard problem Multicolored Clique. The reduction carefully encodes the selection of a multicolored clique through weighted clustering constraints. Due to space limitations, the full construction and correctness proof are deferred to the appendix.

3.2 Employing Structural Parameters for Sparse Graphs

In this subsection, we show that MSR on undirected weighted bipartite graphs is W[1]-hard parameterized by $k + \mathsf{vc}$, where vc is the vertex cover number of the graph. We will use a reduction from the Multicolored Clique problem, which is well-known to be W[1]-hard parameterized by the size of the multicolored clique [15, Lemma 1].

▶ **Definition 3** (Multicolored Clique). *Given an undirected graph G, an integer k, and a partition $(V_1, V_2, \ldots, V_k)$ of the vertices of G; the task is to decide whether there is a k-clique containing exactly one vertex from each set V_i.*

▶ **Theorem 2.** *The MSR problem is* W[1]-hard *on metrics induced by weighted bipartite graphs when parameterized by the vertex cover number (vc) plus the number of clusters (k). Moreover, there is no $f(\mathsf{vc}+k) \cdot N^{o(\mathsf{vc}+k)}$-time algorithm for any computable function f unless* ETH *fails, where N is the number of input points.*

Proof. We provide a polynomial-time parameterized reduction from the Multicolored Clique problem. Since the parameter in our reduction only grows linearly, additionally to W[1]-hardness, we also obtain that there is no $f(vc+k) \cdot N^{o(vc+k)}$-time algorithm for MSR under ETH, where N denotes the number of input points in the MSR instance.

Let $(G, k, (V_1, \ldots, V_k))$ denote an instance of the Multicolored Clique problem. Without loss of generality, we can assume that $V_i = \{v_1^i, \ldots, v_n^i\}$ for every $i \in [k]$. We create an instance (X, d, k', Δ) of the MSR problem, where $k' = 2k$, X is the set of vertices of the weighted undirected graph G' and d is the metric induced by G' as follows; see also Figure 1 for an illustration of the construction. The graph G' contains the following vertices:

- one vertex v for every $v \in V(G)$,
- two vertices x_-^i and x_+^i for every $i \in [k]$,
- two sets W_-^i and W_+^i of $2k+1$ vertices each for every $i \in [k]$,
- a vertex w_e for every non-edge $e = \{u, v\} \notin E(G)$ with endpoints $u \in V_i$ and $v \in V_j$ where $1 \leq i < j \leq k$. Let $\overline{E}_{i,j}$ denote the set of these vertices.

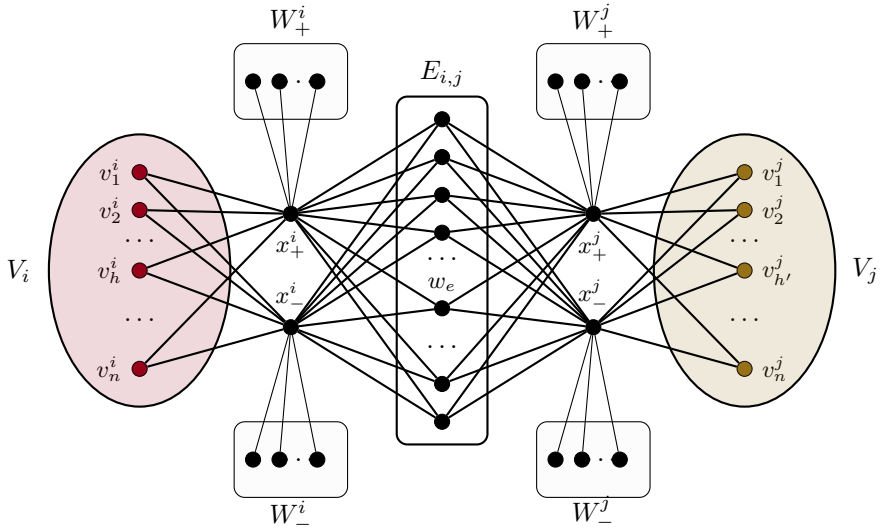


Figure 1 Gadget illustrating the interaction between color classes V_i and V_j in the reduction. For each $i \in [k]$, the two potential centers x_+^i and x_-^i are connected to all vertices of V_i and to two sets of $2k+1$ leaves W_+^i and W_-^i, which enforce the selection of both centers via large-radius penalties. Edges from x_+^i (resp. x_-^i) to $v_h^i \in V_i$ have weight $h + \omega_+(i)$ (resp. $n - h + 1 + \omega_-(i)$), and edges to leaves in W_+^i (resp. W_-^i) have weight $\omega_+(i)$ (resp. $\omega_-(i)$). For every non-edge $e = \{u, v\} \notin E(G)$ with $u = v_h^i \in V_i$ and $v = v_{h'}^j \in V_j$, a vertex w_e (shown in the box $E_{i,j}$) is added and connected to x_+^i, x_-^i with weights $h + 1 + \omega_+(i)$ and $n - h + 1 + \omega_-(i)$ (and symmetrically to x_+^j, x_-^j for j). Selecting incompatible vertices in V_i and V_j forces some w_e to be covered at a prohibitively large radius, while a multicolored clique yields a feasible clustering of total cost at most Δ.

Moreover, G' has the following edges:

- an edge $\{w, x_-^i\}$ of weight $\omega_-(i) = 2^{2i+1} \cdot i \cdot n$ for every $w \in W_-^i$ and every $i \in [k]$,
- an edge $\{w, x_+^i\}$ of weight $\omega_+(i) = 2^{2i} \cdot i \cdot n$ for every $w \in W_+^i$ and every $i \in [k]$,
- an edge $\{x_+^i, v_h^i\}$ of weight $h + \omega_+(i)$ for every $i \in [k]$ and $h \in [n]$,
- an edge $\{x_-^i, v_h^i\}$ of weight $n - h + 1 + \omega_-(i)$ for every $i \in [k]$ and $h \in [n]$,
- an edge $\{x_+^i, w_e\}$ of weight $h+1+\omega_+(i)$ for every $i \in [k]$ and non-edge $e = \{v_h^i, u\} \notin E(G)$ with $u \in V(G) \setminus V_i$.
- an edge $\{x_-^i, w_e\}$ of weight $n - h + 1 + \omega_-(i)$ for every $i \in [k]$ and non-edge $e = \{v_h^i, u\} \notin E(G)$ with $u \in V(G) \setminus V_i$.

Finally, we set the cost Δ to $nk + \sum_{i=1}^{k}(\omega_+(i) + \omega_-(i))$.

This completes the construction of the instance (X, d, k', Δ) of the MSR problem. Note that the construction can be achieved in $\mathsf{poly}(|V|, k)$-time, i.e., polynomial-time.

Moreover, G' is bipartite and because $G' - \{x_+^1, x_-^1, \ldots, x_+^k, x_-^k\}$ is an independent set, it holds that the vertex cover number of G' is at most $k' = 2k$. It therefore only remains to show that G has a multicolored k-clique if and only if (X, d, k', Δ) has a clustering into at most k' clusters of cost at most Δ.

We start by showing the following simple observation.

▶ **Observation 4.** *Let $\{u, v\}$ be an edge of G' of weight $t > 0$. Then, $d(u, v) \geq t$.*

Proof. By construction, every edge of G' is incident to an anchor vertex of the form x_+^i or x_-^i for some $i \in [k]$. Hence, without loss of generality, we may assume that $u = x_+^i$ for some $i \in [k]$; the case $u = x_-^i$ is symmetric.

If $v \in W^i_+$, then v has degree one in G', and hence $\{u, v\}$ is the unique u–v path. Thus $d(u, v) = t$.

Otherwise, let P be any u–v path that does not use the edge $\{u, v\}$. The first edge of P leaves $u = x^i_+$, and by construction every edge incident to x^i_+ has weight at least $\omega_+(i)$. We distinguish the following two cases.

Case 1: $v \in V_i$. Then as v is adjacent exactly to $u = x^i_+$ and x^i_-, the last edge of P must enter v from x^i_-, and hence has weight at least $\omega_-(i)$. Therefore, $w(P) \geq \omega_+(i) + \omega_-(i)$.

Case 2: $v = w_e \in \overline{E}_{i,j}$ **for some** $j \in [k]$ **and** $j \neq i$. Any u–v path P that avoids $\{u, v\}$ must enter v from one of its remaining neighbors x^i_-, x^j_+, or x^j_-. If P enters v from x^i_-, then the last edge has weight at least $\omega_-(i)$, and hence $w(P) \geq \omega_+(i) + \omega_-(i)$. Otherwise, P enters v from x^j_+ or x^j_-, in which case the last edge has weight at least $\omega_+(j)$ or $\omega_-(j)$, respectively, and hence $w(P) \geq \omega_+(i) + \omega_+(j)$. Therefore, $w(P) \geq \omega_+(i) + \min\{\omega_-(i), \omega_+(j)\}$.

In both cases, by construction the weight of the direct $\{u, v\}$ edge satisfies $t \leq \omega_+(i)+n+1$, while for all $j \geq 1$ we have $\omega_+(j) > n + 1$ and $\omega_-(i) > n + 1$. Hence $w(P) > t$, and therefore $d(u, v) \geq t$. ◀

Towards showing the forward direction, let $S = \{v^1_{h_1}, v^2_{h_2}, \ldots, v^k_{h_k}\}$ be a multicolored clique of size k in G for some $h_i \in [n]$ for every $i \in [k]$. We obtain a corresponding clustering $\mathcal{C}$ for (X, d, k', Δ) as follows. For each $1 \leq i \leq k$, the clustering $\mathcal{C}$ contains two clusters C^i_- and C^i_+ with centers x^i_- and x^i_+ and radii $r^i_- = n - h_i + 2^{2i+1} \cdot i \cdot n$ and $r^i_+ = h_i + 2^{2i} \cdot i \cdot n$, respectively.

Clearly, the number of clusters $k' = 2k$ and the overall cost of the clustering $\mathcal{C}$ is

$$\sum_{i\in[k]} \Big((r^i_+) + (r^i_-)\Big) = \sum_{i\in[k]} \Big((h_i + 2^{2i} \cdot i \cdot n) + (n - h_i + 2^{2i+1} \cdot i \cdot n)\Big) = kn + \sum_{i\in[k]} 3 \cdot (2^{2i} \cdot i \cdot n).$$

It therefore merely remains to show that for every vertex $v \in X = V(G')$, there is a cluster in $\mathcal{C}$ containing v. We first show that the vertices in V_i's, W^i_-, and W^i_+ are covered.

- $W^i_- \subseteq C^i_-$ because the weight $2^{2i+1} \cdot i \cdot n$ of any edge $\{x^i_-, w\}$ for $w \in W^i_-$ is at most the radius $r^i_- = n - h_i + 2^{2i+1} \cdot i \cdot n$ of C^i_- (as $1 \leq h_i \leq n$).
- $W^i_+ \subseteq C^i_+$ because the weight $2^{2i} \cdot i \cdot n$ of any edge $\{x^i_+, w\}$ for $w \in W^i_+$ is at most the radius $r^i_+ = h_i + 2^{2i} \cdot i \cdot n$ of C^i_+ (as $1 \leq h_i \leq n$).
- $\{v^i_1, \ldots, v^i_{h_i}\} \subseteq C^i_+$ because the weight of any edge $\{x^i_+, v^i_h\}$ is $h + 2^{2i} \cdot i \cdot n$ and that the radius of C^i_+ is $r^i_+ = h_i + 2^{2i} \cdot i \cdot n$.
- $\{v^i_{h_i+1}, \ldots, v^i_n\} \subseteq C^i_-$ because the weight of any edge $\{x^i_-, v^i_h\}$ is $n - h + 1 + 2^{2i+1} \cdot i \cdot n$ and that the radius of C^i_- is $r^i_- = n - h_i + 2^{2i+1} \cdot i \cdot n$.

It remains to show that the vertices in $\overline{E}_{i,j}$ are covered by $\mathcal{C}$. Towards showing this, we observe the following.

▶ **Observation 5.** *Let $i \in [k]$ and let $v^i_h \in V_i$. Let C^i_+ and C^i_- be the clusters with centers x^i_+ and x^i_- and radii $r^i_+ = h + 2^{2i} \cdot i \cdot n$ and $r^i_- = n - h + 2^{2i+1} \cdot i \cdot n$, respectively. Then, for every $j \in [k] \setminus \{i\}$, it holds that $(C^i_+ \cup C^i_-) \cap \overline{E}_{i,j} = \overline{E}_{i,j} \setminus \{\, w_{\{v^i_h,u\}} \in \overline{E}_{i,j} \mid u \in V_j \,\}$, i.e., $C^i_+ \cup C^i_-$ cover all vertices in $\overline{E}_{i,j}$ apart from the ones that correspond to non-edges incident to v^i_h.*

Proof. Consider a vertex $w_e \in \overline{E}_{i,j}$ with $e = \{v^i_\ell, u\}$ for some $\ell \in [n]$ and $u \in V_j$. Then, the edge $\{x^i_+, w_e\}$ has weight $\ell + 1 + 2^{2i} \cdot i \cdot n$ and because $r^i_+ = h + 2^{2i} \cdot i \cdot n$, it follows from Observation 4 that $w_e \in C^i_+$ if and only if $h \geq \ell + 1$. Therefore, $C^i_+ \cap \overline{E}_{i,j} = \{\, w_{\{v^i_\ell,u\}} \in$

$\overline{E}_{i,j} \mid \ell < h \wedge u \in V_j \}$. Similarly, the edge $\{x^i_-, w_e\}$ has weight $n - \ell + 1 + 2^{2i+1} \cdot i \cdot n$ and because $r^i_- = n - h + 2^{2i+1} \cdot i \cdot n$, it follows from Observation 4 that $w_e \in C^i_-$ if and only if $n - h \geq n - \ell + 1$, i.e., $h \leq \ell - 1$. Therefore, $C^i_- \cap \overline{E}_{i,j} = \{ w_{\{v^i_\ell, u\}} \in \overline{E}_{i,j} \mid \ell > h \wedge u \in V_j \}$. ◀

Now consider the vertices in $\overline{E}_{i,j}$. By Observation 5, $C^i_+ \cup C^i_-$ cover all vertices in $\overline{E}_{i,j}$ apart from the vertices corresponding to non-edges incident to $v^i_{h_i}$. Similarly, $C^j_+ \cup C^j_-$ cover all vertices in $\overline{E}_{i,j}$ apart from the vertices corresponding to non-edges incident to $v^j_{h_j}$. Because S is a multicolored k-clique in G, it holds that $v^i_{h_i}$ and $v^j_{h_j}$ are adjacent in G and therefore there is no non-edge in G that is both incident to $v^i_{h_i}$ and $v^j_{h_j}$ and hence $\overline{E}_{i,j} \subseteq C^i_+ \cup C^i_- \cup C^j_+ \cup C^j_-$.

For the reverse direction, let $\mathcal{C}$ be a solution (clustering) for (X, d, k', Δ). We start by showing that $\mathcal{C}$ has $k' = 2k$ clusters and those use the vertices $x^1_+, x^1_-, \ldots, x^k_+, x^k_-$ as centers with certain radii.

▷ Claim 6. Let $\mathcal{C}$ be a solution for (X, d, k', Δ). Then $\mathcal{C}$ consists of exactly k' clusters with centers $\{x^1_+, x^1_-, \ldots, x^k_+, x^k_-\}$ and no other clusters. Moreover, for every $i \in [k]$ there is an $h_i \in [n]$ such that the cluster centered at x^i_+ has radius $r^i_+ \geq h_i + \omega_+(i)$, and the cluster centered at x^i_- has radius $r^i_- \geq n - h_i + \omega_-(i)$ (recall that $\omega_+(i) = 2^{2i} \cdot i \cdot n$ and $\omega_-(i) = 2^{2i+1} \cdot i \cdot n$).

Proof. Let $\ell \in [k]$ be the largest index such that there is no $h \in [n]$ such that $\mathcal{C}$ has clusters centered either at x^ℓ_+ with radius $r^\ell_- \geq h + \omega_+(\ell)$ or at x^ℓ_- with radius $r^\ell_- \geq n - h + \omega_-(\ell)$, respectively. Note that the statement of the claim now already applies for every $\ell' \in [k]$ with $\ell' > \ell$. We distinguish the following cases.

Case 1: $\mathcal{C}$ has no cluster with center x^ℓ_-. Because $|W^\ell_-| > k'$, there is a vertex, say $w \in W^\ell_-$, that is not the center of any cluster in $\mathcal{C}$. Let C be the cluster in $\mathcal{C}$ that contains w. Then, as w is not the center in $\mathcal{C}$ and has x^ℓ_- as its unique neighbor, C must also contain x^ℓ_-, but since x^ℓ_- is not the center of C and every edge incident to x^ℓ_- has cost at least $\omega_-(\ell)$, we obtain that C has radius at least $2\omega_-(\ell)$. Because the statement of the claim holds for every $i > \ell$, we obtain that the clusters with centers x^i_+ and x^i_- exists for every i where $\ell < i \leq k$ and have total cost at least $(n(k-\ell) + \sum_{i>\ell} \omega_+(i) + \omega_-(i))$. Moreover, even if C is one of these clusters, it is easy to see that in order to cover w the radius of such a cluster would have to be increased by at least $2\omega_-(\ell) - n$ above the lower bound stated in the claim. Therefore, the the cost of $\mathcal{C}$ is at least $(n(k-\ell-1) + \sum_{i>\ell} \omega_+(i) + \omega_-(i)) + 2\omega_-(\ell)$. Because $\Delta = nk + \sum_{i=1}^k (\omega_+(i) + \omega_-(i))$, it suffices to show that $\Delta = nk + \sum_{i=1}^k (\omega_+(i) + \omega_-(i)) < (n(k-\ell-1) + \sum_{i>\ell} \omega_+(i) + \omega_-(i)) + 2\omega_-(\ell)$, which is equivalent to showing that $n(\ell+1) + \sum_{i=1}^{\ell-1} (\omega_+(i) + \omega_-(i)) + \omega_+(\ell) < \omega_-(\ell)$, which is shown as follows. We start by expanding the left hand term

$$
\begin{aligned}
& n(\ell+1) + \omega_+(\ell) + \sum_{i=1}^{\ell-1} (\omega_+(i) + \omega_-(i)) \\
=& n(\ell+1) + 2^{2\ell} \cdot \ell \cdot n + \sum_{i=1}^{\ell-1} (2^{2i} \cdot i \cdot n + 2^{2i+1} \cdot i \cdot n) \\
=& n(\ell+1) + 2^{2\ell} \cdot \ell \cdot n + \sum_{i=1}^{\ell-1} (3 \cdot 2^{2i} \cdot i \cdot n) \\
\leq& n(\ell+1) + 2^{2\ell} \cdot \ell \cdot n + 3n(\ell-1) \sum_{i=1}^{\ell-1} (2^i) \qquad \text{(by replacing } i \text{ with its maximum value)} \\
=& n(\ell+1) + 2^{2\ell} \cdot \ell \cdot n + 3n(\ell-1)\left(\frac{2^{2\ell} - 4}{3}\right) \text{ (by putting the value of the term } \textstyle\sum_{i=1}^{\ell-1} (2^i))
\end{aligned}
$$

$$
\begin{aligned}
&= n(\ell+1) + 2^{2\ell}\cdot\ell\cdot n + n(\ell-1)(2^{2\ell}-4)\\
&= n(\ell+1) + 2^{2\ell}\cdot\ell\cdot n + 2^{2\ell}\cdot\ell\cdot n - 2^{2\ell}\cdot n - 4\cdot\ell\cdot n + 4n\\
&= n(\ell+1) + 2^{2\ell+1}\cdot\ell\cdot n - 2^{2\ell}\cdot n - 4\cdot\ell\cdot n + 4n\\
&= 2^{2\ell+1}\cdot\ell\cdot n - 2^{2\ell}\cdot n - 3\cdot\ell\cdot n + 5n\\
&= \omega_-(\ell) - 2^{2\ell}\cdot n - 3\cdot\ell\cdot n + 5n\\
&\leq \omega_-(\ell) - 2^{2\ell}\cdot n + 2n && \text{(as } -3\cdot\ell\cdot n + 5n \leq 2n \text{ for all } \ell \geq 1\text{)}\\
&< \omega_-(\ell)
\end{aligned}
$$

where the last inequality follows as $-2^{2\ell}\cdot n + 2n < 0$ for all $\ell \geq 1$.

Case 2: $\mathcal{C}$ has no cluster with center x^ℓ_+, but $\mathcal{C}$ has a cluster with center x^ℓ_-. Because $|W^\ell_+| > k'$, there is a vertex, say $w \in W^\ell_+$, that is not the center of any cluster in $\mathcal{C}$. Let C be the cluster in $\mathcal{C}$ that contains w. Then, C must also contain x^ℓ_+, but since x^ℓ_+ is not the center of C and every edge incident to x^ℓ_+ has cost at least $\omega_+(i)$, we obtain that C has radius at least $2\omega_+(i)$. Moreover, the radius of the cluster, say $C' \in \mathcal{C}$, with center x^i_- is at least $\omega_-(i)$, since otherwise C' only contains x^i_- and at least one of the clusters containing a vertex from W^i_- must already contain x^i_- (because $|W^i_-| > k'$). Therefore, the cost of $\mathcal{C}$ is at least $(n(k-\ell)+\sum_{\ell'>\ell}\omega_+(i)+\omega_-(i))+2\omega_+(i)+\omega_-(i) = (n(k-\ell)+\sum_{\ell'>\ell}\omega_+(i)+\omega_-(i))+2\omega_-(i)$ and therefore the same lower bound on the cost obtained in **Case 1**. Therefore, using the same calculation as in **Case 1**, we obtain that this cost lower bound is already larger than Δ.

Case 3: $\mathcal{C}$ has clusters with centers x^ℓ_+ and x^ℓ_-. Note that because of **Case 1** and **Case 2**, we can from now on assume that $\mathcal{C}$ has a cluster with center c for every $c \in \{x^1_+, x^1_-, \ldots, x^k_+, x^k_-\}$ and because $|\{x^1_+, x^1_-, \ldots, x^k_+, x^k_-\}| = k'$, $\mathcal{C}$ does not have any other clusters. Moreover, if $C \in \mathcal{C}$ is a cluster with center x^i_+ (or x^i_-), then its radius r^i_+ (r^i_-) must be at least $\omega_+(i)$ ($\omega_-(i)$). This is because otherwise C only contains x^i_+ (x^i_-) but because $|W^i_+| > k'$ ($|W^i_-| > k'$), there must be another cluster containing some $w \in W^i_+$ ($w \in W^i_-$) and x^i_+ (x^i_-) and therefore C would not be needed. It follows that the cost of the clustering is already at least $n(k-\ell) + \sum_{i=1}^k(\omega_+(i)+\omega_-(i))$, leaving only $n\ell$ for additional costs.

Because the conditions of the claim are not satisfied for ℓ, it must now holds that $r^\ell_+ + r^\ell_- < \omega_+(\ell) + \omega_-(\ell) + n$. But then, there must be some $h \in \{0\} \cup [n-1]$ such that $r^\ell_+ = \omega_+(\ell) + h$ and $r^\ell_- < n - h + \omega_-(\ell)$, which implies that the vertex v^i_{h+1} is not covered by C^ℓ_+ or C^ℓ_-. But then, v^i_{h+1} must be contained in some other cluster, say C, with center $c \in \{x^1_+, x^1_-, \ldots, x^k_+, x^k_-\} \setminus \{x^\ell_+, x^\ell_-\}$. Assume that $c = x^i_\lambda$ for some $i \in [k] \setminus \{\ell\}$ and $\lambda \in \{+,-\}$. Then, in order to cover v^i_{h+1}, the radius of C must be at least by $2\omega_+(\ell)$ larger than the lower bound that we computed for C above, i.e., the cost of the clustering is now at least $n(k-\ell) + \sum_{i=1}^k(\omega_+(i)+\omega_-(i)) + 2\omega_+(\ell)$. Since $2\omega_+(\ell) = 2\cdot 2^{2\ell}\cdot\ell\cdot n > n\ell$, this exceeds the allowed cost of $\Delta = nk + \sum_{i=1}^\ell(\omega_+(i)+\omega_-(i))$ and concludes the proof of the claim. ◁

Because of Claim 6, we obtain that $\mathcal{C}$ contains exactly one cluster C_c with center c for every $c \in \{x^1_+, x^1_-, \ldots, x^k_+, x^k_-\}$ and no other clusters. Moreover, for every $i \in [k]$, there is an $h_i \in [n]$ such that the cluster in $\mathcal{C}$ with center at x^i_+ or x^i_- has radius $r^i_+ \geq h_i + \omega_+(i)$ or $r^i_- \geq n - h_i + \omega_-(i)$, respectively. Since the total cost is at most Δ, these bounds are tight, and hence $r^i_+ = h_i + \omega_+(i)$ and $r^i_- = n - h_i + \omega_-(i)$ for all $i \in [k]$. We now claim that the vertex set $S = \{v^1_{h_1}, \ldots, v^k_{h_k}\}$ is a k-clique of G. In other words, it is sufficient to show that $\{v^i_{h_i}, v^j_{h_j}\} \in E(G)$ for every $1 \leq i < j \leq k$. Because of Observation 5, we obtain that for every $\ell \in \{i,j\}$, it holds that $C_{x^\ell_+} \cup C_{x^\ell_-}$ cover exactly all vertices in $\overline{E}_{i,j}$ that do

not correspond to non-edges incident to $v^{\ell}_{h_\ell}$. Therefore, $C_{x^i_+} \cup C_{x^i_-} \cup C_{x^j_+} \cup C_{x^j_-}$ cover all vertices in $\overline{E}_{i,j}$ if and only if G does not contain a non-edge between $v^i_{h_i}$ and $v^j_{h_j}$, i.e., if and only if $\{v^i_{h_i}, v^j_{h_j}\} \in E(G)$. In the following, we observer that every vertex in $\overline{E}_{i,j}$ must be covered by one of the clusters $C_{x^i_+}$, $C_{x^i_-}$, $C_{x^j_+}$, $C_{x^j_-}$. Suppose this is not the case, i.e., there exists a vertex $w_e \in E_{i,j}$ that is not covered by one of the clusters $C_{x^i_+}$, $C_{x^i_-}$, $C_{x^j_+}$, $C_{x^j_-}$. As $\mathcal{C}$ is a valid clustering of (X, d, k, Δ) and hence w_e must be covered by some other cluster centered at some x^t_+ (or at x^t_-) where $t \in [k] \neq i, j$, which by construction must have radius more than $r^t_+ = 2^{2t} \cdot t \cdot n + h_{t'}$ (or more than $r^t_- = 2^{2t+1} \cdot t \cdot n + n - h_{t'}$) for some $h_{t'} \in V_t$, a contradiction to Claim 6. This concludes the reverse direction of the proof. ◀

It is interesting to note that the result of Theorem 2 even holds for the variant of MSR, where the set of centers is given and one merely needs to find the right radius for every center.

3.3 Completing the Picture for the Weighted Case

Here, we want to complete the picture for the parameterized complexity of MSR on metrics induced by weighted graphs w.r.t. structural parameters. As we have shown in Theorem 2, MSR is W[1]-hard parameterized already by the vertex cover number, which is arguably one of the most restrictive parameters in the context of sparse graphs. Clearly, this already excludes fixed-parameter algorithms for MSR on metrics induced by weighted graphs parameterized by feedback vertex set number, treedepth, treewidth, cliquewidth etc. However, it does not exclude fixed-parameter algorithms for other structural parameters defined for dense graphs such as neighborhood diversity, modular width, and modular treedepth. The following result excludes such algorithms by showing that MSR is already NP-hard on complete graphs and complete bipartite graphs, i.e., graphs where all of these parameters are constant. Furthermore, it even excludes fixed-parameter tractability when combining any of those structural parameters with the parameter $k + \Delta$.

▶ **Theorem 7.** *The MSR problem is* NP-hard *and* W[1]-hard *parameterized by* $k + \Delta$ *even on metrics induced by weighted complete graphs and complete bipartite graphs.*

Proof. It follows from Theorem 1 that MSR is NP-hard and W[1]-hard parameterized by $k + \Delta$ even on metrics induced by weighted bipartite graphs and we will show the theorem by providing a parameterized and polynomial-time reduction from this case to the two cases stated in the theorem. Let (X, d, k, Δ) be an instance of the MSR problem such that X is the set of vertices of the weighted bipartite graph G with partition $\{A, B\}$ and d is the metric induced by G. To reduce to the case of a complete graph, let (X, d', k, Δ) be the instance of MSR such that d' is the metric induced by the graph G' obtained from G after adding an edge of cost $\Delta + 1$ between any two vertices in G that are not adjacent. Similarly, to reduce to the case of a complete bipartite graph, let (X, d'', k, Δ) be the instance of MSR such that d'' is the metric induced by the graph G'' obtained from G after adding an edge of cost $\Delta + 1$ between any vertex in A and any vertex in B.

It is straightforward to show that these two constructions provide parameterized and polynomial-time reductions as required. ◀

While Theorem 7 excludes fixed-parameter tractability for any dense structural parameter plus $k + \Delta$, we will now show that MSR on metrics induced by weighted graphs is fixed-parameter tractable parameterized by the treewidth of the graph plus the cost of the clustering. To do so, we need the following definitions.

Let G be an weighted graph and let $v \in V(G)$. We denote by $N_r[v]$ the *r-neighborhood* of v, i.e., the all vertices that have distance at most r to v. Moreover, we denote by $\#(v)$ the number of distinct sets in $\{ N_r[v] \mid r \in \mathbb{R} \}$ and by $\#(G)$ the maximum of $\#(v)$ over all vertices $v \in V(G)$. Using these notions, fixed-parameter tractability for treewidth plus cost now follows immediately from the algorithm used to show [14, Theorem 1] by observing that the number of vertices $|V|$ in their statement of the run-time of the algorithm can be easily replaced by the more concise parameter $\#(G)$.

▶ **Proposition 8** ([14, Theorem 1]). *The MSR problem on metrics induced by weighted graphs can be solved in time $\mathcal{O}(\omega 2^{3\omega}(\#(G))^{3\omega+1}k^3)$, where ω is the treewidth of the graph G underlying the metric.*

Since $\#(G)$ is at most the cost Δ , we obtain the following corollary from the above proposition.

▶ **Corollary 9.** *The MSR problem on metrics induced by weighted graphs is fixed-parameter tractable parameterized by treewidth plus Δ.*

4 MSR on metrics induced by (unweighted) graphs

We now consider the case of (unweighted) graphs. As mentioned in the introduction, it is a long standing open question whether MSR on metrics induced by unweighted graphs is solvable in polynomial-time [6]. While we are not able to answer this question, we provide the following main contributions. First, we exhibit two natural variants of MSR and provide strong hardness results for these variants even on metrics induced by unweighted graphs. In particular, we introduce the Exact-MSR version (where one is required to find a solution with exactly k non-zero clusters) in Section 4.1 and show that it is NP-hard and W[1]-hard parameterized by $k + \Delta$. Moreover, we introduce the Allowed-Centers-MSR version (where one is additionally required to choose the clusters centers from a given set of allowed centers) in Section 4.2 and show that it is W[1]-hard parameterized by either $k + \Delta$ or $k + |A| + \mathsf{fvs}$, where A is the set of allowed centers and fvs is the feedback vertex number of the underlying undirected graph. It is important to note that we are not able to show NP-hardness for Allowed-Centers-MSR and we believe that our analysis for Allowed-Centers-MSR can be seen as a way around having to first settle the classical complexity of MSR on graphs. That is, in order to obtain a parameterized complexity classification for MSR it might be easier to show W[1]-hardness as we do for Allowed-Centers-MSR. In addition to this, we also show that MSR as well as Exact-MSR and Allowed-Centers-MSR are fixed-parameter tractable parameterized by the treedepth of the underlying graph. We start by providing the algorithmic result and then introduce the two variants and show their hardness in the following two subsections.

Let G be an undirected graph and let $\ell(G)$ denote the length of a longest path in G. Then, $\#(G) \leq \ell(G)$. It is well-known that the longest paths of an undirected graph is asymptotically the same as the well-known parameter treedepth and that the treewidth of a graph is at most equal to its treedepth, i.e.:

▶ **Proposition 10** ([25]). *Let G be an undirected graph. Then, $\mathsf{tw}(G) \leq \mathsf{td}(G)$ and $\log \ell(G) \leq \mathsf{td}(G) \leq \ell(G)$.*

Therefore, we obtain that $\#(G) \leq 2^{\mathsf{td}}(G)$ and $\mathsf{tw}(G) \leq \mathsf{td}(G)$, which together with Proposition 8 implies (it is straightforward to verify that the DP algorithm used to show Proposition 8 also works for Exact-MSRand Allowed-Centers-MSR):

▶ **Corollary 11.** *The MSR problem (and the* Exact-MSR *and* Allowed-Centers-MSR *problems) on metrics induced by unweighted graphs is fixed-parameter tractable parameterized by the treedepth of the graph.*

4.1 Exact-MSR

In this subsection, we study the version of the MSR problem, which we call Exact-MSR that asks for exactly k clusters each containing at least two points (this corresponds to requiring non-zero clusters). Please refer to Section 1 for a formal definition of Exact-MSR. We show that even on metrics induced by an undirected bipartite graphs, Exact-MSR is NP-hard and W[2]-hard parameterized by $k + \Delta$.

▶ **Theorem 12.** *Exact-MSR on metrics induced by unweighted bipartite graphs is* NP-hard *and* W[2]-hard *parameterized by* $k+\Delta$*. Moreover, assuming the Exponential Time Hypothesis, there is no algorithm for Exact-MSR running in time* $f(k+\Delta) \cdot n^{o(k+\Delta)}$ *for any computable function* f*.*

Proof. We provide a parameterized reduction from the Dominating Set problem on bipartite graphs that do not contain isolated vertices, which is well-known to be NP-hard and W[2]-hard when parameterized by the solution size [13]. That is, given an instance (G, k) of the Dominating Set problem (which asks whether the bipartite graph G without isolated vertices has a dominating set of size at most k), we will construct the equivalent instance (X, d, k, k) of Exact-MSR in polynomial-time as follows. We set $X = V(G)$ and d is the metric induced by G. This completes the reduction, which can clearly be performed in polynomial time and it only remains to show that (G, k) is a Yes-instance of Dominating Set if and only if (X, d, k, k) has a solution.

Towards showing the forward direction, let $D = \{d_1, \ldots, d_\ell\}$ be a dominating set of G of size at most $\ell \leq k$. We claim that $\mathcal{C} = \{(d_1, 1), \ldots, (d_\ell, 1), (v_1, 1), \ldots, (v_{k-\ell}, 1)\}$, where the v_j's are arbitrary vertices in G (that are pairwise distinct and distinct from all d_i') is a solution for (X, d, k, k). Clearly, $|\mathcal{C}| = k$, the cost of $\mathcal{C}$ is at most k and since all center have at least one neighbor in G, it holds that at least two vertices are within distance at most 1 from every center. Moreover, because D is a dominating set in G, it holds that every vertex in G is within the radius of at least one of the clusters in $\mathcal{C}$.

Towards showing the reverse direction, let $\mathcal{C} = \{ (x_i, r_i) \mid i \in [k] \}$ be a solution for (X, d, k, k). Because at least two vertices are within distance at most r_i from x_i, it holds that $r_i \geq 1$. Moreover, since $\sum_{i \in [k]} r_i \leq \Delta = k$, it holds that $r_i = 1$. Therefore, every vertex of G is a neighbor of some x_i in G, which shows that the set $\{x_1, \ldots, x_k\}$ is a dominating set of G.

Finally, we note that the existence of a $f(k + \Delta) \cdot n^{o(k+\Delta)}$ time algorithm for any computable function f for Exact-MSR would imply a $f(k) \cdot n^{o(k)}$ time algorithm for Dominating Set problem, a contradiction to ETH, via [11, Corollary 14.23]. ◀

4.2 Allowed-Centers-MSR

In this subsection, we study the version of the MSR problem, which we coin Allowed-Centers-MSR, where we are additionally given a set of allowed centers. Please refer to Section 1 for a formal definition of Allowed-Centers-MSR. We start by showing that even on metrics induced by an unweighted graph, Allowed-Centers-MSR parameterized by $k + \Delta$ is W[1]-hard.

▶ **Theorem 13.** *Allowed-Centers-MSR on metrics induced by unweighted graphs is* W[1]-hard *parameterized by* $k + \Delta$. *Moreover, there is no algorithm that solve Allowed-Centers-MSR in time* $f(k + \Delta) \cdot n^{o(k+\Delta)}$ *time for any computable function* f, *under* ETH.

Proof. Consider the reduction from the Multicolored Clique problem to the MSR problem provided in the proof of Theorem 1. In particular, consider the instance (X, d, k, Δ) of MSR together with the undirected edge-weighted graph G' that induces the metric (X, d) obtained from the instance $(G, k, (V_1, \ldots, V_k))$ of Multicolored Clique in the proof of Theorem 1. The idea is to transform (X, d, k, Δ) into an instance (X', d', A, k, Δ) of Allowed-Centers-MSR such that the metric (X', d') is induced by the undirected graph G'', while preserving equivalence as follows. Let G'' be the undirected graph obtained from G' after subdividing every edge e of G' with weight w, $w - 1$ many times. Moreover, let $X' = V(G'')$, let d' be the metric induced by G'' and let $A = V(G)$ (recall that $V(G) \subseteq V(G')$). Because we are required to perform an exponential number of edge subdivisions, this is no longer a polynomial-time reduction (and therefore does not provide NP-hardness) but it is a parameterized reduction for the parameters $k + \Delta$. It is now straightforward to verify that $(G, k, (V_1, \ldots, V_k))$ has a solution if and only if so does (X', d', A, k, Δ).

Finally, we note that the existence of a $f(k + \Delta) \cdot n^{o(k+\log_2 \Delta)}$ time algorithm for any computable function f for Exact-MSR would imply a $f(k) \cdot n^{o(k)}$ time algorithm for Multicolored Clique problem, a contradiction to ETH, via [11, Corollary 14.23]. ◀

The next theorem shows that Allowed-Centers-MSR parameterized by $k + |A| + \mathsf{fvs}$ is W[1]-hard, where A is the set of allowed centers and fvs is the feedback vertex set number.

▶ **Theorem 14.** *Allowed-Centers-MSR on metrics induced by unweighted graphs is* W[1]-hard *parameterized by* $k + |A| + \mathsf{fvs}$. *Moreover, there is no algorithm that solves Allowed-Centers-MSR in time* $f(k + |A| + \mathsf{fvs}) \cdot n^{o(k+|A|+\mathsf{fvs})}$ *time for any computable function* f, *under* ETH.

Proof. Consider the reduction from the Multicolored Clique problem to the MSR problem provided in the proof of Theorem 2. In particular, consider the instance (X, d, k', Δ) of MSRwith $k' = 2k$ together with the undirected edge-weighted graph G' that induces the metric (X, d) obtained from the instance $(G, k, (V_1, \ldots, V_k))$ of Multicolored Clique in the proof of Theorem 2. The idea is to transform (X, d, k', Δ) into an instance (X', d', A, k', Δ) of Allowed-Centers-MSR such that the metric (X', d') is induced by the undirected graph G'', while preserving equivalence as follows. Let G'' be the undirected graph obtained from G' after subdividing every edge e of G' with weight w, $w - 1$ many times. Moreover, let $X' = V(G'')$, let d' be the metric induced by G'' and let $A = \{ x^i_-, x^i_+ \mid i \in [k] \}$. Note first that A is a feedback vertex set for G'' and therefore the feedback vertex number of G'' is at most $k' = 2k$. Moreover, because we are required to perform an exponential number of edge subdivisions, this is no longer a polynomial-time reduction (and therefore does not provide NP-hardness) but it is a parameterized reduction for the parameters $k + |A| + \mathsf{fvs}$, where fvs is the feedback vertex set number of G''. It is now straightforward to verify that $(G, k, (V_1, \ldots, V_k))$ has a solution if and only if so does (X', d', A, k', Δ).

Finally, since $k + |A| + \mathsf{fvs} = 5k$, the existence of an algorithm for Exact-MSR running in time $f(k + |A| + \mathsf{fvs}) \cdot n^{o(k+|A|+\mathsf{fvs})}$, for any computable function f, would imply an $f(k) \cdot n^{o(k)}$-time algorithm for the Multicolored Clique problem. This would contradict ETH via [11, Corollary 14.23]. ◀

5 Conclusion

Our main contribution is a comprehensive classification of the parameterized complexity of MSR for metrics induced by undirected weighted graphs for any combination of the parameters k, Δ, any structural parameter at least as restrictive as vertex cover, as well as the most common structural parameters for dense graphs such as neighborhood diversity, modular width, and cliquewidth. We also made some contribution for the unweighted case by showing (parameterized) complexity lower bounds for two natural and closely related variants. While the weighted case is now fairly well understood, we think that it would be natural and interesting to look at what happens to the parameterized complexity on planar graphs (or generalizations of planar graphs), where neither of our two main hardness results apply.

Finally, a major task for future work is to obtain a similarly detailed (parameterized) complexity landscape for the unweighted case. Here, the main obstacle is the open complexity status of MSR on undirected unweighted graphs, i.e., only the version with non-zero clusters is known to be solvable in polynomial-time and for the general case there is a randomized quasi polynomial-time algorithm, which makes it unlikely to be NP-hard. While it may be difficult to settle this question, our W[1]-hardness result for Allowed-Centers-MSR may offer a way out, since it suffices to show W[1]-hardness instead of NP-hardness to obtain a parameterized complexity classification. Note that we leave it open whether Allowed-Centers-MSR is NP-hard on undirected (unweighted) graph metrics, but are able to show that the problem is W[1]-hard.

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 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

3 Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT constant-approximations for capacitated clustering to minimize the sum of cluster radii. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SOCG.2023.12.

4 Sandip Banerjee, Yair Bartal, Lee-Ad Gottlieb, and Alon Hovav. Improved fixed-parameter bounds for min-sum-radii and diameters k-clustering and their fair variants. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 15481–15488. AAAI Press, 2025. doi:10.1609/AAAI.V39I15.33699.

5 Babak Behsaz and Mohammad R. Salavatipour. On minimum sum of radii and diameters clustering. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2012. doi:10.1007/978-3-642-31155-0_7.

6 Babak Behsaz and Mohammad R. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73(1):143–165, 2015. doi:10.1007/S00453-014-9907-3.

7 Moritz Buchem, Katja Ettmayr, Hugo K. K. Rosado, and Andreas Wiese. A $(3 + \varepsilon)$-approximation algorithm for the minimum sum of radii problem with outliers and extensions for generalized lower bounds. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1738–1765. SIAM, 2024. doi:10.1137/1.9781611977912.69.

8 Lena Carta, Lukas Drexler, Annika Hennes, Clemens Rösner, and Melanie Schmidt. FPT approximations for fair k-min-sum-radii. In *Proceedings of the 35th International Symposium on Algorithms and Computation (ISAAC)*, volume 322 of *LIPIcs*, pages 16:1–16:18, 2024. doi:10.4230/LIPIcs.ISAAC.2024.16.

9 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 1–10. ACM, 2001. doi:10.1145/380752.380753.

10 Xianrun Chen, Dachuan Xu, Yicheng Xu, and Yong Zhang. Parameterized approximation algorithms for sum of radii clustering and variants. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 20666–20673. AAAI Press, 2024. doi:10.1609/AAAI.V38I18.30053.

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 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49. IEEE Computer Society, 1992. doi:10.1109/SCT.1992.215379.

14 Lukas Drexler, Jan Höckendorff, Joshua Könen, and Kevin Schewior. Clustering graphs of bounded treewidth to minimize the sum of radius-dependent costs. *CoRR*, abs/2310.02130, 2023. doi:10.48550/arXiv.2310.02130.

15 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/J.TCS.2008.09.065.

16 Zachary Friggstad and Mahya Jamshidian. Improved polynomial-time approximations for clustering with minimum sum of radii or diameters. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 56:1–56:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.56.

17 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On metric clustering to minimize the sum of radii. In Joachim Gudmundsson, editor, *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings*, volume 5124 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2008. doi:10.1007/978-3-540-69903-3_26.

18 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi R. Varadarajan. On clustering to minimize the sum of radii. *SIAM J. Comput.*, 41(1):47–60, 2012. doi:10.1137/100798144.

19 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

20 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discret. Appl. Math.*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

22 Tanmay Inamdar and Kasturi R. Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 62:1–62:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.62.

23 Ragesh Jaiswal, Amit Kumar, and Jatin Yadav. FPT approximation for capacitated sum of radii. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 65:1–65:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ITCS.2024.65.

24 Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Comput. Networks*, 47(4):489–501, 2005. doi:10.1016/J.COMNET.2004.08.012.

25 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

New Results on Three-Sided Skyline Range Counting and Reporting

Suruchi Kushwaha
Michigan Technological University, Houghton, MI, USA

Yakov Nekrich
Michigan Technological University, Houghton, MI, USA

Abstract

In the orthogonal skyline range counting (resp. reporting) problem we store the set of points P in a data structure so that for any query range Q the number of points (resp. the list of all points) on the skyline of $Q \cap P$ can be found efficiently. In this paper we study two-dimensional range counting and reporting problems in the case when the query range is bounded on three sides.

We describe a linear-space data structure that answers *top-open* three-sided skyline counting queries in $O(\log \log N)$ time, where N is the number of points stored in the data structure. We also show that *bottom-open* three-sided skyline counting queries are as difficult as general four-sided queries and any data structure that uses $O(N \log^c N)$ space for a constant c requires $\Omega(\log N / \log \log N)$ time to answer such queries.

Next, we turn to skyline *color* range queries. In this variant of the problem each point in P is assigned a color and we must count (resp. report) the distinct colors of point on the skyline of $Q \cap P$. We describe an $O(N)$-space data structure that answers top-open three-sided color counting queries in $O(\log N / \log \log N)$ time. Finally, we study top-open three-sided skyline color reporting in the EM model and describe a data structure that uses linear space and answers queries in $O(\frac{k}{B} + 1)$ I/Os where k is the number of colors on the skyline. This is the first external-memory data structure with optimal query cost and space usage for this problem.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Data Structures, Range Searching, Skyline Queries

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.27

1 Introduction

A two-dimensional point $p = (p.x, p.y)$ is said to be dominated by a point $q = (q.x, q.y)$ if $q.x > p.x$ and $q.y \geq p.y$, or $q.x \geq p.x$ and $q.y > p.y$. For a set of points P, the skyline of P is the subset of points that are not dominated by any other point in P. A skyline range searching query Q on a set of points P asks for (some information) about the skyline of points in $P \cap Q$.

In this paper we study different variants of two-dimensional skyline range searching in the case when the query range is bounded on three sides. See Figure 1. We present new results for three-sided skyline range counting queries and three-sided skyline color range counting queries. We also describe a data structure for three-sided skyline color range reporting in the external memory model.

Previous Work

A skyline range counting query Q asks for the number of points on the skyline of $Q \cap P$. Orthogonal skyline range counting queries were considered in [10, 6, 3]. Brodal and Larsen [3] described a $O(N)$-space data structure with $O(\log N / \log \log N)$ query time for a four-sided range. In [3] the authors also showed that this bound is tight: any data structure that uses $N \log^{O(1)} N$ space supports skyline range counting queries in $\Omega(\log N / \log \log N)$ time.

© Suruchi Kushwaha and Yakov Nekrich;
licensed under Creative Commons License CC-BY 4.0

A skyline range reporting query Q asks for the list of all points on the skyline of $P \cap Q$. Data structures for orthogonal skyline range reporting queries were studied in [5, 3, 11, 9]. Das et al. [5] presented a data structure that uses $O(N \log N / \log \log N)$ space and supports four-sided skyline range reporting queries in $O(\log N / \log \log N + k)$ time, where k is the number of reported skyline points. Brodal and Larsen [3] imporved their result and described a data structure that requires query time $O(\log N / \log \log N + k)$ and space usage $O(N \log^{\varepsilon} N)$.

In the external memory model [17], Kejlberg-Rasmussen et al. [11] present a linear space data structure that answers four-sided skyline reporting queries in $O((N/B)^{\varepsilon} + k/B)$ I/Os, where k is the number of reported points and B is the block size. In the same paper [11] the authors present a linear-space data structure that answers top-open three-sided skyline reporting queries in $O(1 + k/B)$ I/Os when all points are on $N \times N$ grid.

Ganguly et al. [9] studied the colored version of skyline range reporting problem in the external-memory model. In this variant of the problem each point is assigned a color (or category). The answer to a query Q is the list of colors that occur on the skyline of $P \cap Q$. For the case when all points are on the $N \times N$ grid, Ganguly et al. [9] describe a data structure that uses $O(N \log^* N)$ space and answers top-open three-sided queries in $O(1 + k/B)$ I/Os and a data structure that uses $O(N)$ space and answers queries in $O((\log^* N)^2 + k/B)$ I/Os, where k is the number of reported colors.

Our Results

In this paper we describe a linear-space data structure that answers *top-open* three-sided skyline range counting queries on $N \times N$ grid in time $O(\log \log N)$. We also show that any data structure with $O(N \log^c N)$ space for some constant $c > 0$ needs $\Omega(\log N / \log \log N)$ time to answer *bottom-open* three-sided skyline range counting queries. This result demonstrates that there is a fundamental gap between top-open queries (i.e., when the query range is unbounded in $+y$ direction) and bottom-open queries (i.e., when the query range is unbounded in $-y$ direction) with respect to skyline counting queries. Moreover, due to the lower bound on orthogonal range counting [16], we need $\Omega(\log N / \log \log N)$ time to answer two-dimensional range counting queries using $O(N\text{polylog(N)})$ space. This lower bound is also valid when the query range is a top-open three-sided range. Our result thus shows that counting points on a skyline of a query range can be fundamentally faster than counting all points in a query range.

Next we turn to the skyline color range counting problem. We describe a data structure that answers top-open three-sided skyline color counting queries in $O(\log N / \log \log N)$ time. That is, our data structure calculates, for any top-open three-sided query range Q, the number of distinct colors of points on the skyline of Q. This is the first data structure for the skyline color counting problem.

Finally we consider top-open three-sided color range reporting in the external-memory model. We improve upon the previous result by Ganguly et al. [9] and describe a data structure that uses $O(N)$ space and supports queries in $O(\frac{k}{B} + 1)$ I/Os, where k is the number of reported colors and B is the block size. Our result demonstrates that three-sided skyline color reporting can be implemented with optimal space and query cost. For comparison, the best currently known solution for the three-sided color reporting problem (i.e., for the case when *all* colors in a three-sided query range must be reported) either uses super-linear space or has super-constant query cost [15]. Thus our result shows that skyline reporting can be more efficient than general color reporting in the case of top-open three-sided ranges (with respect to the best currently known results).

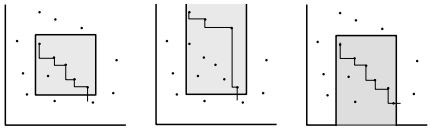
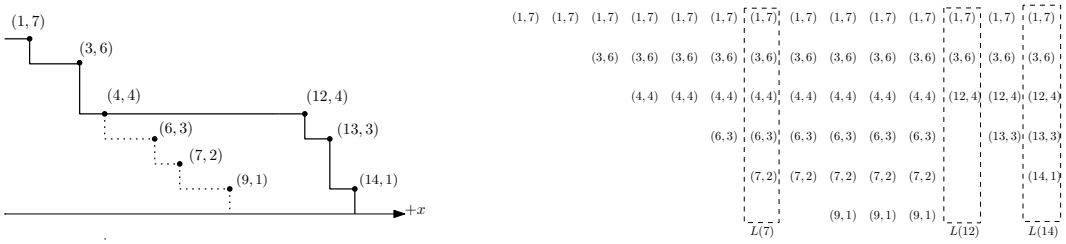


Figure 1 Examples of four-sided (left) top-open three-sided (center) and bottom-open three-sided (right) skyline queries.



(a) Example skyline points.

(b) Example skyline points in partially persistent list.

Figure 2 Skyline points in our data structure.

For simplicity we will assume throughout this paper that point coordinates are in the rank space [8, 1], i.e., all point coordinates are distinct integers from $\{1, \ldots, N\}$. We reserve N to denote the total number of points in a data structure and B will denote the block size in the external memory model.

2 Top-Open Skyline Counting

In this section we describe a simple linear-space data structure that answers three-sided skyline counting queries in $O(\log \log N)$ time. Our solution uses the sweepline technique: a vertical sweepline l is moved in $+x$ direction and a subset of points to the left of l is stored in a partially persistent list L. When a point p is hit by the sweepline, the list L is updated as follows: we remove all points $p' \in L$ with $p'.y \leq p.y$; then we add p to L. Points in L are stored in decreasing order of their y-coordinates. We will denote by $L(a)$ the version of L after removing all p' and inserting the point p with $p.x = a$. See Figure 2 for an example.

▶ **Lemma 1.** *When the vertical sweepline hits a point p, p is appended at the end of L.*

Proof. Follows directly from the description. When a point p is inserted into L, all points p' with $p'.y < p.y$ must be removed from L. ◀

The *rank* of a point p in the list L, denoted by $\text{rank}(p)$, is equal to the number of points that precede p in L (plus one). By Lemma 1, when a point p is inserted into $L(p.x)$, the rank of p equals the size of $L(p.x)$; the rank of any point p is not changed while p is in L.

▶ **Lemma 2.** *A point p is in $L(a)$ iff p is on the the skyline of $Q_a = (-\infty, a] \times (-\infty, +\infty)$.*

Proof. If p is not on the skyline of Q_a, then p is dominated by some point $p' \in Q_a$. Since $p.x < p'.x \leq a$ and $p.y \leq p'.y$, p was removed from $L(a')$ for some $a' \leq p'.x \leq a$ and $p \notin L(a)$. If p is on the skyline of Q_a, then there is no point p'' such that $p.x \leq p''.x \leq a$ and $p.y \leq p''.y$. By definition of L, $p \in L(a)$. ◀

If a point p precedes a point q in L, we will denote this by $p \prec q$.

▶ **Lemma 3.** *If $p \prec q$ in L, then $p.x < q.x$ and $p.y > q.y$.*

Proof. Consider the version $L(q.x)$ of L.
By Lemma 1, q is appended at the end of $L(q.x)$ and p precedes q in $L(q.x)$; since L is sorted by y-coordinates, $p.y > q.y$. By Lemma 2, p and q are on the skyline of $(-\infty, a] \times (-\infty, +\infty)$. Since $p.y \geq q.y$ and p does not dominate q, $p.x < q.x$. ◀

▶ **Lemma 4.** *A point p is on the skyline of $Q = [a, b] \times [c, +\infty)$ iff p is on the skyline of $(-\infty, b] \times (-\infty, +\infty)$, $p.x \geq a$ and $p.y \geq c$.*

Proof. Consider a point p on the skyline of $[a, b] \times [c, +\infty)$. Then $p.x \geq a$ and $p.y \geq c$. There is no point $p' \in [a, b] \times [c, +\infty)$ that dominates p. Any point p' that dominates p must satisfy $p'.x \geq a$ and $p'.y \geq c$. Hence, there is no point in $(-\infty, b] \times (-\infty, +\infty)$ that dominates p. Conversely, consider a point p on the skyline of $(-\infty, b] \times (-\infty, +\infty)$ such that $p.x \geq a$ and $p.y \geq c$. There is no point p' that dominates p in the region $(-\infty, b] \times (-\infty, +\infty)$. Hence there is no point p' that dominates p in $[a, b] \times [c, +\infty)$. Since p is in Q and p is not dominated by any other $p' \in Q$, p is on the skyline of Q. ◀

▶ **Theorem 5.** *There exists an $O(N)$-space data structure that answers top-open three-sided skyline counting queries on $N \times N$ grid in time $O(\log \log N)$.*

Proof. We use the partially persistent predecessor data structure by Chan [4] that supports queries in $O(\log \log N)$ time and uses $O(N)$ space where N is the total number of elements stored in the data structure. We employ two instances of Chan's data structure, one that supports queries on x-coordinates of points in L and one that supports queries on y-coordinates of points in L. Suppose that we must answer a skyline counting query on a range $Q = [a, b] \times [c, +\infty)$. By Lemmas 2 and 4, a point p is on the skyline of $[a, b] \times [c, +\infty)$ iff $p \in L(b)$, $p.x \geq a$ and $p.y \leq b$. Points in $L(b)$ are sorted in increasing order by their x-coordinates and in decreasing order by their y-coordinates. Let p_c denote the point in $L(b)$ with the smallest y-coordinate such that $p_c.y \geq c$. Let p_a denote the point in $L(b)$ with the largest x-coordinate such that $p_c.x \geq a$. By Lemma 3, the skyline of Q consists of all points p in $L(b)$ such that $p_a \preceq p \preceq p_c$. The number of such points is $\text{rank}(p_c) - \text{rank}(p_a) + 1$, where $\text{rank}(q)$ denotes the rank of a point q in L. ◀

Using the reduction to rank space technique, we can extend the result of Theorem 5 to the general case of N points on a $U \times U$ grid; see e.g., [1]. In this general case, the space usage of the data structure remains $O(N)$ and the query time is increased by an additive term $q_{pred}(N) = O(\min(\sqrt{\log N / \log \log N}, \log \log U))$, where $q_{pred}(N)$ is the time needed to answer a predecessor query [13].

3 Adjusted Right Boundary

For a given set of points P and query range Q the range predecessor of $P \cap Q$ is the point p_s with the largest x-coordinate in $P \cap Q$. The *adjusted right boundary* of $P \cap Q$, denoted by $\texttt{adj}(P, Q)$, is the x-coordinate of p_s. Thus $\texttt{adj}(P, Q)$ is the x-coordinate of the rightmost point in $P \cap Q$. For simplicity we will denote $\texttt{adj}(P, Q)$ as $\texttt{adj}(Q)$, since P is the entire input set. Using the adjusted right boundary, we can convert a skyline query on a top-open three-sided range $Q = [a, b] \times [c, +\infty)$ into a skyline query on an one-dimensional range $Q' = [a, b'] \times (-\infty, +\infty)$ where $b' = \texttt{adj}(Q)$. We will use the partially persistent list L, introduced in Section 2 to describe our result in this section.

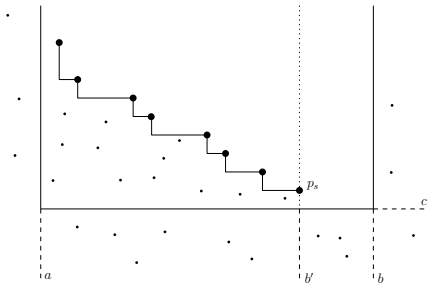


Figure 3 An example of adjusted right boundary; $b' = \texttt{adj}(Q)$ for $Q = [a, b] \times [c, +\infty)$.

▶ **Lemma 6.** *A point p is on the skyline of $Q = [a, b] \times [c, +\infty)$ iff p is on the skyline of $Q' = [a, b'] \times (-\infty, +\infty)$ where $b' = \texttt{adj}(Q)$.*

Proof. Suppose that a point p is on the skyline of Q, i.e there is no other point $q \in Q$ that dominates p. By definition of $\texttt{adj}(Q)$, b' is the largest x-coordinate of a point in Q. Hence, $p.x \leq b'$ and therefore p must be in Q'. There is no point in $[a, b'] \times [c, +\infty)$ that dominates p because $[a, b'] \times [c, +\infty) \subset [a, b] \times (-\infty, +\infty)$ and there is no point with y-coordinate less than c that dominates p because $p.y \geq c$. Therefore, p is on the skyline of Q'. Now, suppose the point p is on the skyline of Q'. Since $p.x \leq b'$ and $b' \leq b$, $p.x \leq b$. Let p_r be the point such that $b' = p_r.x$. Then p_r is the point with the largest x-coordinate in Q and $p_r.y \geq c$. Since $p.x \leq p_r.x$ and p is on the skyline of Q', $p.y \geq p_r.y \geq c$. There is also no point q with $q.x \in [b', b]$ and $q.y \geq c$. Therefore $p \in Q$ and is also on the skyline of Q. ◀

The following lemma enables us to compute the range predecessor query in a top-open three-sided range.

▶ **Lemma 7.** *Let $Q = [a, b] \times [c, +\infty)$ and let p_c denote the point that is the successor of c in $L(b)$ with respect to y-coordinates. If Q is not empty, then p_c is the range predecessor of Q.*

Proof. According to Lemma 2, $L(b)$ is the skyline of points in $(-\infty, b] \times (-\infty, +\infty)$, hence $p_c.x \leq b$. Since, p_c is the successor of c in $L(b)$ with respect to y-coordinates, $p_c.y \geq c$ and by Lemma 3, $p_c.x$ must be the point with largest x-coordinate in $L(b)$ among the points that have y-coordinate greater than c. Therefore, p_c is the point with largest x-coordinate in the range $(-\infty, b] \times [c, +\infty)$. Since Q is not empty, $p_c.x \geq a$. Hence, $p_c \in Q$ and is also the range predecessor of Q. ◀

To find the adjusted right boundary of a top-open three-sided query range $Q = [a, b] \times [c, +\infty)$, we visit the version $L(b)$ of the partially persistent list, i.e. the version that is created when the vertical sweepline hits $x = b$. By Lemma 7, we can find the adjusted right boundary in $O(\log \log N)$ time using the persistent predecessor data structure from [4].

4 Bottom-open Three-sided Queries

In this section we show that a four-sided skyline range searching query $Q = [a, b] \times [c, d]$ can be reduced to a bottom-open three-sided skyline range searching query $Q' = [a, b'] \times (-\infty, d]$ where b' is the x-coordinate of the rightmost point in Q (aka the range predecessor of Q). In other words the skyline of Q is the same as the skyline of the bottom-open range $Q' = [a, b'] \times (-\infty, d]$, where $b' = p_s.x$.

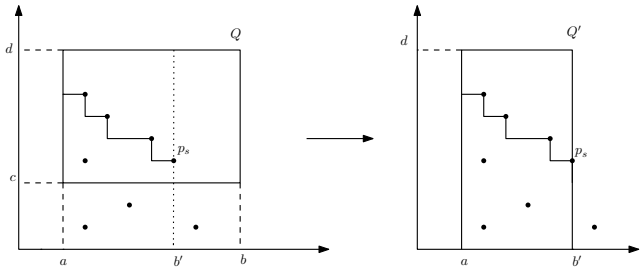


Figure 4 Reduction of a four-sided skyline query to a bottom-open three-sided skyline query.

▶ **Lemma 8.** *Let p_s denote the rightmost point in a four-sided range $Q = [a, b] \times [c, d]$. A point p is on the skyline of Q iff p is on the skyline of $Q' = [a, b'] \times (-\infty, d]$ where $b' = p_s.x$.*

Proof. Consider some point p on the skyline of Q. Since p_s is the rightmost point in Q, $p.x \leq p_s.x$ and hence p is in Q'. Since p is on the skyline of Q and $b' \leq b$ there is no point in $[a, b'] \times [c, d]$ that dominates p. No point in $[a, b'] \times (-\infty, c]$ can dominate p because $p.y \geq c$. Hence, no point in Q' dominates p and p is on the skyline of Q'. Now, suppose a point p is on the skyline of Q'. Since $b' = p_s.x$, $p.x \leq p_s.x$. The point p is not dominated by p_s because p is on the skyline of Q' and hence $p.y \geq p_s.y \geq c$ since p_s is the rightmost point of Q. This implies that $p \in Q$ and, since $[b', b] \times [c, d]$ is empty, p is also on the skyline of Q. ◀

We can use any data structure for bottom-open three-sided skyline queries to answer four-sided skyline queries, with an additional cost of finding the rightmost point in the four-sided range. Through this reduction, we can demonstrate that solving a bottom-open three-sided skyline query is as hard as solving a four-sided skyline query.

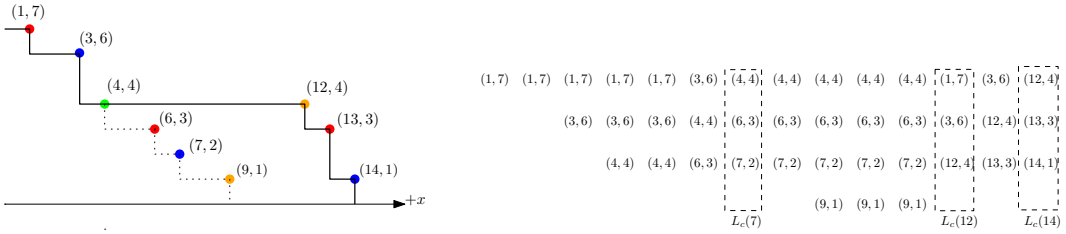
▶ **Theorem 9.** *If we can answer a bottom-open three-sided skyline counting query in time $O(Q(N))$ and $O(S(N))$ space, then we can answer a four-sided skyline counting query in time $O(Q(N) + Q'(N))$ and $O(S(N) + S'(N))$ space where $Q'(N)$ and $S'(N)$ are the time and space required for answering a range predecessor query.*

Using the result of Nekrich and Navarro [14], the range predecessor query can be answered in $O(N)$ space and $O(\log^{\varepsilon} N)$ time. Hence we can set $S'(N) = O(N)$ and $Q'(N) = O(\log^{\varepsilon} N)$ in Theorem 9. As a direct corollary of this result, we establish the lower bound of $\Omega(\log N / \log\log N)$ for bottom-open three-sided skyline counting queries. This lower bound is also valid when all points are on $N \times N$ grid.

▶ **Corollary 10.** *Any data structure that uses $O(N\text{polylog}(N))$ space needs $\Omega(\log N / \log\log N)$ time to answer a bottom-open three-sided skyline counting query.*

Proof. In [3], Brodal and Larsen prove the $\Omega(\log N / \log\log N)$ lower bound for four-sided skyline counting queries. By Theorem 9 and [14], a data structure that supports three-sided bottom-open queries in $O(S(N))$ space and $O(Q(N))$ time can be used to answer four-sided queries in $O(S(N)+N)$ space and $O(Q(N)+\log^{\varepsilon} N)$ time. Hence, the lower bound of Brodal and Larsen [3] is also valid for bottom-open three-sided queries. ◀

We remark that Corollary 10 is valid even if points are in the rank space. On the other hand, we have shown in Theorem 5 that top-open three-sided skyline counting queries can be answered in $O(\log\log N)$ time using linear space. Theorem 5 and Corollary 10 show that there is a significant complexity gap between the top-open and bottom-open variants of three-sided skyline counting.



(a) Example color skyline points. **(b)** Example color skyline points in partially persistent list.

Figure 5 Color skyline points in our data structure.

5 Top-Open Skyline Color Counting

In this section, we present a linear space data structure for skyline color counting queries on a top-open three-sided range in $O(\log N/\log\log N)$ time. We use the adjusted right boundary, defined in Section 3, to reduce a top-open three-sided query to a two-sided query; then we count the distinct colors on the skyline of a two-sided range. The data structure will use the persistent list approach, similar to Section 2. But now our task is more challenging because distinct colors of points on the skyline must be counted and there could be several points of the same color. Therefore we will employ a different persistent list and combine this approach with the reduction of three-sided to two-sided queries from Section 3.

Let L denote the partially persistent list that was defined in Section 2 and let L_c be the sub-list of the partially persistent list L where for every version k and each color α that occurs in $L(k)$, $L_c(k)$ stores the bottommost point of color α from $L(k)$. Thus, as the vertical sweepline l moves along $+x$ direction, we keep only the bottommost point of each distinct color on the skyline to the left of l in the partially persistent list L_c. Our definition of the sublist is motivated by the fact that we must count the number of distinct colors on the skyline. Thus, L_c must contain exactly one point for every color on the skyline of the respective two-sided range. The correctness of our method will be proved in Lemma 11, 12, and 13.

We distinguish between three different kinds of updates on L_c. When l hits a point p with $p.x = a$, we *erase* all points $p' \in L_c$ such that $p.y \geq p'.y$. An erased point p' is permanently deleted from L_c as it will no longer be on the skyline; that is, p' will not be reinserted into any later versions of the list L_c. When a point p' is erased, we reinsert the bottommost previously removed point p'_r such that $p'_r.color = p'.color$ and $p'_r.y > p.y$ (provided such point p'_r exists). If there is a point $q \in L_c$ such that $q.color = p.color$ and $q.y > p.y$, we *remove* q. A removed point q is still on the skyline but is not the bottommost point of its color. Hence it must be deleted from the current version of L_c, but it can be reinserted into a later version of L_c. Then we add p to L_c. We will denote by $L_c(a)$ the version of L_c after erasing all p', removing q, and inserting p.

▶ **Lemma 11.** *There is at least one point of color α on the skyline of $Q = (-\infty, b] \times (-\infty, \infty)$ iff there is exactly one point of color α in $L_c(b)$.*

Proof. Suppose there is one or more points $p_1, p_2, p_3, ..., p_t$ where $t \geq 1$ of color α on the skyline of Q. Then by Lemma 2, $p_1, p_2, p_3, ..., p_t \in L(b)$ and by definition the bottommost point is in $L_c(b)$. Therefore $L_c(b)$ contains exactly one point of color α. Conversely, let p_α be the point of color α in $L_c(b)$. Then p_α is also in $L(b)$ because $L_c(b) \subset L(b)$. Hence, by Lemma 2, p_α is on the skyline of the query range Q and there is at least one point of color α in Q. ◀

▶ **Lemma 12.** *There is at least one point of color α on the skyline of $Q = [a, b] \times (-\infty, +\infty)$ iff there is exactly one point $p \in L_c(b)$ such that $p.x \geq a$.*

Proof. Suppose there are one or more points $p_1, p_2, p_3, ..., p_t$ where $t \geq 1$ of color α on the skyline of Q. Then by Lemma 2, $p_1, p_2, p_3, ..., p_t \in L(b)$. Let p_b denote the bottommost point of color α in $L(b)$. Since $p_b \in Q$, $p_b.x \geq a$. By definition, only $p_b \in L_c(b)$. Therefore, $L_c(b)$ contains exactly one point of color α such that $p.x \geq a$. Conversely, consider the point $p \in L_c(b)$ with $p.x \geq a$. Then by definition, $p \in L(b)$ and by Lemma 2, p is on the skyline of $(-\infty, b] \times (-\infty, +\infty)$. Finally, using a special case of Lemma 4, where $c = -\infty$, p is on the skyline of Q. ◀

▶ **Lemma 13.** *There is at least one point of color α on the skyline of $Q = [a, b] \times [c, +\infty)$ iff there is exactly one point p of color α in $L_c(b')$ such that $p.x \geq a$ and $b' = \texttt{adj}(Q)$.*

Proof. Suppose there is at least one point p of color α on the skyline of Q. By Lemma 6, p_b is on the skyline of $[a, b'] \times (-\infty, +\infty)$ where $b' = \texttt{adj}(Q)$. Hence by Lemma 13, there is exactly one point of color α in $L_c(b')$. Conversely, suppose the point p of color α is in $L_c(b')$. By Lemma 11, p is on the skyline of $(-\infty, b') \times (-\infty, +\infty)$. Given that $p.x \geq a$, hence by Lemma 6, p is on the skyline of Q. ◀

We can count the number of points $r \in L_c(b')$ with $r.x \geq a$ and $b' = \texttt{adj}(Q)$ by answering a partially persistent range counting query. We use the data structure of Munro et. al. [12] that answers such queries in $O(\log m / \log \log m)$ time and uses $O(m)$ space, where m is the total number of updates in L_c. We will show below that $m = O(N)$. Thus we can count the number of points $r \in L_c(b')$ with $r.x \geq a$ in $O(\log m / \log \log m) = O(\log N / \log \log N)$ time and the total query time is $O(\log N / \log \log N)$.

Space Usage

To demonstrate that the space requirement for the data structure is $O(N)$, we need to evaluate the number of updates for L_c. Suppose that every update costs \$1 and each point is initially assigned \$4 before being inserted into the partially persistent list. We will show below that this overall budget of $\$4N$ is sufficient to execute all necessary updates of L_c. The insert operation costs \$1, leaving the point with \$3. At any time, a point $p \in L_c$ can be either removed or erased from the list. When p is removed, \$1 is spent; thus each removed point has \$2. A removed point p is reinserted into L_c only when a point p_o with the same color and smaller y-coordinate is erased from L_c. The erased point, with \$3, spends \$1 on the erase operation; the remaining \$2 are transferred from p_o to the reinserted point p. Thus p spends \$1 on an insertion and receives \$2 from p_o. Hence when a point is re-inserted into p_b, it has a budget of \$3. In summary, the list L_c is updated $m \leq 4N$ times. Since the total number of updates is $m = O(N)$, the space used by L_c and the range counting data structure is also bounded by $O(N)$.

▶ **Theorem 14.** *There exists an $O(N)$-space data structure that answers top-open three-sided skyline color counting queries in $O(\log N / \log \log N)$ time.*

6 Top-Open Skyline Color Reporting

In this section we describe an external memory data structure that uses linear space and supports top-open three-sided skyline color reporting queries in $O(\frac{k}{B}) + 1$ I/Os, where k is the number of reported colors. Our data structure uses the persistent list L_c defined in

Section 5. By Lemma 12, a color occurs on the skyline of $Q = [a, b] \times [c, +\infty)$ iff there is a point p of that color in $L_c(b')$, such that $p.x \geq a$ and b' is the adjusted right boundary of Q. Thus, a query can be answered by accessing the version $L_c(b')$ of L_c. However, it is not clear whether we can compute the adjusted right boundary in $O(1)$ time. In this paper we use a different approach that relies on the results from previous sections. We will show that we can either (1) calculate b' in $O(\frac{k}{B})$ I/Os when the number of colors k is sufficiently large or (2) calculate the adjusted right boundary in $O(1)$ time for an appropriately selected small subset of points.

The following two lemmas will be used in this section.

▶ **Lemma 15.** *Let S be a set of integers such that $|S| = O(B \log^2 N)$. There exists a data structure that uses space $O(|S|)$ and supports predecessor queries on S in $O(1)$ I/Os.*

▶ **Lemma 16.** *Let P' be a set of two-dimensional points such that $|P'| = O\big(B \log^2 N\big)$. There exists a data structure that uses space $O(|P'|)$ and answers range predecessor queries on P' in $O(1)$ I/Os.*

Both lemmas can be easily proved by combining standard data structures; for completeness, we provide a proof in Appendix.

Data Structure

We divide the points into *chunks* of size $\Delta = B \log^2 N$ according to their x-coordinates. Thus, the i-th chunk contains points p, such that $i\Delta < p.x \leq (i+1)\Delta$ and the total number of chunks is $\frac{N}{\Delta}$. Each chunk is stored in the data structure of Lemma 16.

Chunks are also grouped into a number of (intersecting) sets, further called *canonical sets*. For every i such that $1 \leq i < \frac{N}{\Delta}$ and for every j such that $(i + 2^j)\Delta \leq N$, the canonical set $I_{i,j}$ consists of all points with x-coordinates in $[i\Delta + 1, (i + 2^j)\Delta]$. For each canonical set I, we store the list $TS(I)$ that contains $B \log N$ topmost points with distinct colors that occur on the skyline of I. More formally: Let $V(I)$ denote the (conceptual) list of all points on the skyline of I, sorted by y-coordinates in decreasing order. Let $V'(I)$ denote the sub-list of $V(I)$ that contains the topmost point of every color α that occurs in $V(I)$. The set $TS(I)$ contains the first $B \log N$ points from $V'(I)$. The y-coordinates of points in $TS(I)$ are stored[1] in the data structure of Lemma 15.

To find $\texttt{adj}(Q)$ for a top-open three-sided query range $Q = [a, b] \times [c, +\infty)$, we first identify the chunks containing a and b. Let the chunks containing a and b be C_a and C_b respectively. The query $Q = [a, b] \times [c, +\infty)$ is then divided into three parts: the left marginal query $Q_{lmarg} = [a, a_1 - 1] \times [c, +\infty)$, the right marginal query $Q_{rmarg} = [b_1 + 1, b] \times [c, +\infty)$ and the middle query $Q_{mid} = [a_1, b_1] \times [c, +\infty)$; the parameters a_1 and b_1 are chosen so that $a_1 - 1$ is the right x-boundary of the chunk C_a and $b_1 + 1$ is the left x-boundary of the chunk C_b. See Fig 6 for illustration. The marginal queries fit into individual chunks and the middle query spans an integer number of chunks. We will search for the appropriate adjusted right boundary in Q_{rmarg}, Q_{middle} and Q_{lmarg} (in this order).

The following lemma provides an explanation for our method. We will say that a query range Q is *empty* if $P \cap Q = \emptyset$.

[1] We remark that lists $V(I)$ and $V'(I)$ are not stored in our data structure. These lists were introduced in order to define $TS(I)$.

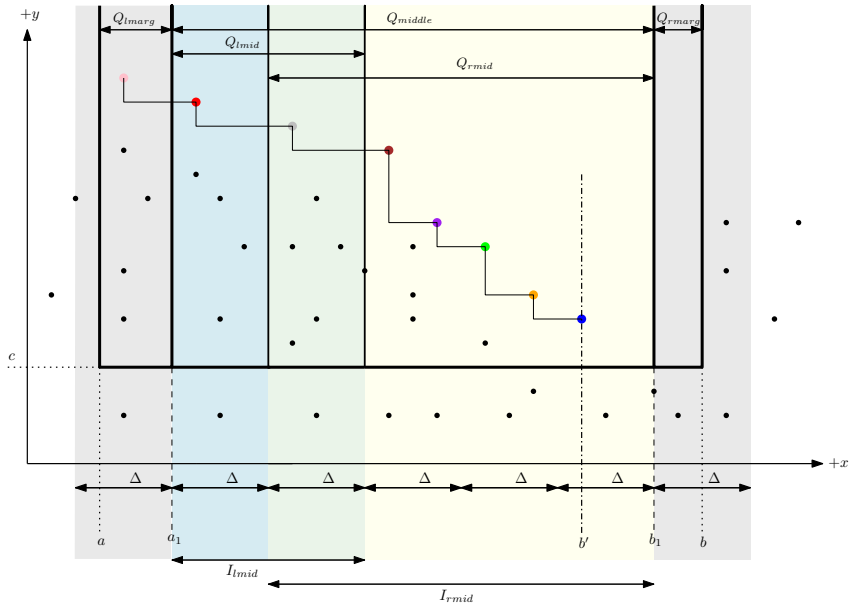


Figure 6 Procedure for finding $\texttt{adj}(Q)$. A query range Q is decomposed into Q_{lmarg}, Q_{middle}, and Q_{rmarg}. Q_{middle} is then decomposed into two overlapping query ranges, Q_{lmid} and Q_{rmid}.

▶ **Lemma 17.** *Let $Q = [a, b] \times [c, +\infty)$ and consider any r such that $a \leq r \leq b$. If $Q_r = [r, b] \times [c, +\infty)$ is empty, then the adjusted right boundary of $Q_l = [a, r-1] \times [c, +\infty)$ is the adjusted right boundary of Q.*
If $[r, b] \times [c, +\infty)$ is not empty, then the adjusted right boundary of $[r, b] \times [c, +\infty)$ is the adjusted right boundary of Q.

Proof. By definition, $\texttt{adj}(Q)$ is the largest x-coordinate of a point in Q, see Section 3. If the right region $[r, b] \times [c, +\infty)$ is empty the rightmost point must lie in the left region $Q_l = [a, r-1] \times [c, +\infty)$. Hence, $\texttt{adj}(Q) = \texttt{adj}([a, r-1] \times [c, +\infty))$. But if the right region $Q_r = [r, b] \times [c, +\infty)$ is not empty, there must be a point p in the right region such that $p.x > p'.x$ for any point $p' \in Q_l$; hence $\texttt{adj}(Q)$ is the same as $\texttt{adj}([r, b] \times [c, +\infty))$. ◀

Lemma 17 can be applied iteratively and we can divide the query range into more than two parts in the same manner. Next, we present two more lemmas that will be used to describe the method for searching the adjusted right boundary.

▶ **Lemma 18.** *Consider a canonical set $I = [x_1, x_2]$ and a query range $Q(I) = [x_1, x_2] \times [c, +\infty)$ for some c. Let $right(TS(I)) = (r_x, r_y)$ denote the lowermost point in $TS(I)$. If $c \leq r_y$, then there are at least $B \log N$ distinct colors on the skyline of $Q(I)$, where B is the size of the block.*

Proof. By definition, $TS(I)$ contains $B \log N$ topmost points with distinct colors from the skyline of I. Each such point $p = (p_x, p_y) \in TS(I)$ satisfies $p_x \in [x_1, x_2]$ and $p_y \geq r_y$. Since $c \leq r_y \leq p_y$, we conclude that $p \in Q(I)$. Therefore all points from $TS(I)$ are also in $Q(I)$. Since points in $TS(I)$ are on the skyline of I, they are also on the skyline of $Q(I)$. Since all points in $TS(I)$ have different colors, the skyline of $Q(I)$ must contain at least $B \log N$ distinct colors. ◀

▶ **Lemma 19.** *Consider a canonical set $I = [x_1, x_2] \times (-\infty, +\infty)$ and a query range $Q = [x_1, x_2] \times [c, +\infty)$ such that $c > r_y$, where right$(TS(I)) = (r_x, r_y)$ is the rightmost (i.e., the lowermost) point in $TS(I)$. A color α occurs on the skyline of $Q \cap P$ if and only if it occurs on the skyline of $Q' \cap P$, where $Q' = [x_1, b'] \times (-\infty, +\infty)$ and $b' = \texttt{adj}(TS(I), Q)$.*

Proof. Let p be the topmost point of color α on the skyline of $Q \cap P$. By definition of $\texttt{adj}(TS(I), Q)$, we have $p_x \leq b'$, and since b' is the rightmost x-coordinate among the topmost points of distinct colors on the skyline of Q, there is no point of color α in $[b', x_2] \times [c, +\infty)$ that dominates p. Also, since $p_y \geq c$, no point in $[x_1, b'] \times (-\infty, c]$ dominates p. Hence, p is on the skyline of $Q' \cap P$ and the color α occurs on the skyline of $Q' \cap P$. Conversely, let p' be the topmost point of color α on the skyline of $Q' \cap P$. Since $p' \in Q' \cap P$, $p_x \leq b'$. By definition of $\texttt{adj}(TS(I), Q)$, $p_y \geq c$ and $b' \leq x_2$. This implies $p'_x \leq b' \leq x_2$ and hence $p' \in Q \cap P$. As p' is already the topmost point of its color and lies to the left of b', no point in $[b', x_2] \times [c, +\infty) \subseteq Q$ can dominate it. Therefore, p' also lies on the skyline of $Q \cap P$ and the color α must also be on the skyline of $Q \cap P$. ◀

In what follows, we describe the method for searching the adjusted right boundary in detail (see Figure 6 for a visual illustration).

`Step 1.` We calculate $\texttt{adj}(P, Q_{rmarg})$ or determine that $Q_{rmarg} \cap P = \emptyset$ using the data structure for the chunk that contains Q_{rmarg}. By Lemma 17, $\texttt{adj}(P, Q) = \texttt{adj}(P, Q_{rmarg})$ if $Q_{rmarg} \cap P \neq \emptyset$, and using Lemma 16 we can find $\texttt{adj}(P, Q_{rmarg})$ in $O(1)$ time. If Q_{rmarg} is empty, we proceed with Step 2.

`Step 2.` Let $h = \lfloor \frac{b_1 - a_1 + 1}{\Delta} \rfloor$. We divide Q_{middle} into two query ranges, Q_{lmid} and Q_{rmid}. Let $Q_{lmid} = [a_1, a_1 + 2^h\Delta - 1] \times [c, +\infty)$ and $Q_{rmid} = [b_1 - 2^h\Delta + 1, b_1] \times [c, +\infty)$. Let I_{lmid} denote the canonical set that contains all points with x-coordinates in $[a_1, a_1 + 2^h\Delta - 1]$. Let I_{rmid} denote the canonical set that contains all points with x-coordinates in $[b_1 - 2^h\Delta + 1, b_1]$. We observe that $Q_{lmid} \cap P \subseteq I_{lmid}$ and $Q_{rmid} \cap P \subseteq I_{rmid}$ for two canonical sets $I_{lmid} = [a_1, a_1 + 2^h\Delta - 1] \times (-\infty, +\infty)$ and $I_{rmid} = [b_1 - 2^h\Delta + 1, b_1] \times (-\infty, +\infty)$.

`Step 2(a).` Let p_t denote the topmost point in $TS(I_{rmid})$ and let p_l denote the lowermost point in $TS(I_{rmid})$. Since points in $TS(I_{rmid})$ are sorted by y-coordinates, we can identify p_t and p_l in $O(1)$ I/Os. We distinguish between three different cases.

`Case 1.` $\boldsymbol{p_t.y < c}$: In this case $Q_{rmid} \cap P = \emptyset$ and we proceed with Step 3.

`Case 2.` $\boldsymbol{p_l.y \geq c}$: In this case we find $\texttt{adj}(Q)$ in $O(\log_B N) = O(\log N)$ I/Os, but we can show that $k \geq B \log N$. By Lemma 18, the total number of colors on the skyline of Q_{rmid} is at least $B \log N$. Since $Q_{rmid} \subseteq Q$, $k \geq B \log N$, where k is the total number of colors on the skyline of Q. Since $k \geq B \log N$, the adjusted right boundary $\texttt{adj}(Q_{rmid}, P)$ is found in $O(\log_B N) = O(\frac{k}{B})$ I/Os.

`Case 3.` $\boldsymbol{p_t.y \geq c}$ `and` $\boldsymbol{p_l.y < c}$: In this case it is sufficient to find $\texttt{adj}\big(TS(I_{\text{rmid}}), Q\big)$ as can be concluded from Lemma 19. Since $TS(I_{rmid})$ contains $B \log N$ points, we can find $\texttt{adj}(TS(I_{rmid}), Q)$ in $O(1)$ I/Os using Lemma 15.

`Step 3.` If $Q_{rmid} \cap P = \emptyset$, we process Q_{lmid} in the same way as was described for Q_{rmid} in Step 2.

`Step 4.` If $Q_{lmid} \cap P = \emptyset$, we find $\texttt{adj}(P, Q) = \texttt{adj}(P, Q_{lmarg})$ in $O(1)$ I/Os, in the same way as described for Q_{rmarg} in Step 1.

Space Usage

Each individual chunk contains $B \log^2 N$ points, and there are N/Δ chunks, where $\Delta = B \log^2 N$. Therefore, the total space usage of chunk data structures is $O(N)$. Additionally, $TS(I)$ for every canonical set I stores $O(B \log N)$ points, and there are $O\big(\frac{N}{\Delta} \log \frac{N}{\Delta}\big)$ canonical

sets because for every chunk there are $O\big(\log(N/\Delta)\big)$ sets of exponentially increasing size starting at the left boundary of that chunk. In total, the data structure occupies $O(N)$ words.

Our procedure for transforming a three-sided top-open query range into a two-sided query range can be summarized in the following lemma.

▶ **Lemma 20.** *For any query range $Q = [a, b] \times [c, +\infty)$, we can find some value b' that satisfies the following property: a color α occurs on a skyline of $Q \cap P$ iff it occurs on the skyline of $Q' \cap P$ for $Q' = [a, b'] \times (-\infty, +\infty)$. The cost of finding b' is $O(k/B)$ I/Os, where k is the number of colors on the skyline of Q and the underlying data structure uses space $O(N)$.*

Color Reporting Queries

The distinct colors on the skyline of $Q = [a, b] \times [c, +\infty)$ can be reported as follows. We find the value of b', such that a color occurs on the skyline of Q iff it occurs on the skyline of $Q' = [a, b'] \times (-\infty, +\infty)$. By Lemma 20 this can be done in $O(k/B)$ I/Os where k is the number of colors on the skyline of Q. Then we access the partially persistent list L_c at the version b', i.e $L_c(b')$, and traverse the list in order of decreasing x-coordinates. We stop when a point p with $p.x < a$ is reached. For each point we report its color. The time required to examine all such points and report their colors is $O(k'/B)$ I/Os, where k' is the number of points $p \in L_c(b')$ with $p.x \geq a$. By Lemma 12, for every color on the skyline of Q' there is exactly one point p with that color, such that $p \in L_c(b')$ and $p.x \geq a$.

▶ **Theorem 21.** *In the external memory model, there exists a $O(N)$-space data structure that answers top-open three-sided skyline color reporting queries on an $N \times N$ grid in $O(\frac{k}{B}) + 1$ I/Os.*

The general case can be reduced to the case when points are on $N \times N$ grid at the cost of increasing the query cost by $q_{pred}(N)$; see the remark after Theorem 5.

7 Conclusion

In this paper we show that there is a gap between top-open and bottom-open three-sided skyline counting queries: while bottom-open queries have the same $\Theta(\log N/\log\log N)$ time complexity as four-sided skyline counting queries, top-open queries can be answered in $O(\log\log N)$ time. Our technique also lead to improved results on (top-open) three-sided skyline color counting and (top-open) three-sided range reporting in external memory.

One important open question related to our work concerns the complexity of four-sided skyline range reporting. The best data structure for four-sided range reporting in 2-D answers queries in $O(\log\log N + k)$ time. This time is known to be optimal. On the other hand, the fastest currently known data structure for four-sided skyline range reporting answers queries is $O(\log N/\log\log N + k)$ time. Is there a data structure that uses $N\log^{O(1)} N$ space and supports four-sided skyline reporting queries in $O(\log\log N + k)$ time?

References

1 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000. doi:10.1109/SFCS.2000.892088.

2 Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An asymptotically optimal multiversion b-tree. *The VLDB Journal*, pages 264–275, 1996. doi:10.1007/s007780050028.

3 Gerth Stølting Brodal and Kasper Green Larsen. Optimal planar orthogonal skyline counting queries. In *14th Scandinavian Symposium and Workshops (SWAT)*, pages 110–121, 2014. doi:10.1007/978-3-319-08404-6_10.

4 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, pages 22:1–22:22, 2013. doi:10.1145/2483699.2483702.

5 Ananda Swarup Das, Prosenjit Gupta, Anil Kishore Kalavagattu, Jatin Agarwal, Kannan Srinathan, and Kishore Kothapalli. Range aggregate maximal points in the plane. In *6th International Workshop on Algorithms and Computation (WALCOM)*, 2012. doi:10.1007/978-3-642-28076-4_8.

6 Ananda Swarup Das, Prosenjit Gupta, and Kannan Srinathan. Counting maximal points in a query orthogonal rectangle. In *7th International Workshop on Algorithms and Computation (WALCOM)*, 2013. doi:10.1007/978-3-642-36065-7_8.

7 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, pages 424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

8 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th annual ACM Symposium on Theory of computing (STOC)*, pages 135–143, 1984. doi:10.1145/800057.808675.

9 Arnab Ganguly, Daniel Gibney, Sharma V. Thankachan, and Rahul Shah. I/O-optimal categorical 3-sided skyline queries. *Theoretical Computer Science*, pages 132–144, 2021. doi:10.1016/j.tcs.2021.10.010.

10 Anil Kishore Kalavagattu, Jatin Agarwal, Ananda Swarup Das, and Kishore Kothapalli. On counting range maxima points in plane. In *Combinatorial Algorithms, 23rd International Workshop (IWOCA)*, pages 263–273, 2012. doi:10.1007/978-3-642-35926-2_28.

11 Casper Kejlberg-Rasmussen, Yufei Tao, Konstantinos Tsakalidis, Kostas Tsichlas, and Jeonghun Yoon. I/O-efficient planar range skyline and attrition priority queues. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 103–114, 2013. doi:10.1145/2463664.2465225.

12 J. Ian Munro, Yakov Nekrich, and Sharma V. Thankachan. Range counting with distinct constraints. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015. URL: http://research.cs.queensu.ca/cccg2015/CCCG15-papers/44.pdf.

13 Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Computing Surveys*, 53(5):105:1–105:35, 2021. doi:10.1145/3409371.

14 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *13th Scandinavian Symposium and Workshops (SWAT)*, pages 271–282. Springer, 2012. doi:10.1007/978-3-642-31155-0_24.

15 Manish Patil, Sharma V. Thankachan, Rahul Shah, Yakov Nekrich, and Jeffrey Scott Vitter. Categorical range maxima queries. In *33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 266–277. ACM, 2014. doi:10.1145/2594538.2594557.

16 Mihai Patrascu. Lower bounds for 2-dimensional range counting. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 40–46, 2007. doi:10.1145/1250790.1250797.

17 Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, pages 209–271, 2001. doi:10.1145/384192.384193.

A Proofs of Lemmas 15 and 16

▶ **Lemma 15.** *Let S be a set of integers such that $|S| = O(B \log^2 N)$. There exists a data structure that uses space $O(|S|)$ and supports predecessor queries on S in $O(1)$ I/Os.*

Proof. If $B \geq \log N$, then $|S| \leq B^2$. In this case we store S in a B-tree so that a predecessor query is answered in $O(\log_B |S|) = O(1)$ I/Os. If $B < \log N$, $|S| \leq \log^2 N$. In this case we store S in a fusion tree [7], so that a predecessor query is answered in $O(\log_{\log n} |S|) = O(\frac{\log |S|}{\log \log n}) = O(1)$. ◀

▶ **Lemma 16.** *Let P' be a set of two-dimensional points such that $|P'| = O(B \log^2 N)$. There exists a data structure that uses space $O(|P'|)$ and answers range predecessor queries on P' in $O(1)$ I/Os.*

Proof. According to Lemma 7, the rightmost point in a query range Q can be found using a partially persistent data structure. If $B \geq \log N$, we store the points in a partially persistent B-Tree [2]. If $B < \log N$, we store the points in a partially persistent variant of the fusion tree. ◀

The Parameterized Complexity of Coloring Mixed Graphs

Antonio Lauerbach
Julius-Maximilians-Universität Würzburg, Germany

Konstanty Junosza-Szaniawski
Warsaw University of Technology, Poland

Marie Diana Sieper
Julius-Maximilians-Universität Würzburg, Germany

Alexander Wolff
Julius-Maximilians-Universität Würzburg, Germany

Abstract

A mixed graph contains (undirected) edges as well as (directed) arcs, thus generalizing undirected and directed graphs. A proper coloring c of a mixed graph G assigns a positive integer to each vertex such that $c(u) \neq c(v)$ for every edge $\{u, v\}$ and $c(u) < c(v)$ for every arc (u, v) of G. As in classical coloring, the objective is to minimize the number of colors. Thus, mixed (graph) coloring generalizes classical coloring of undirected graphs and allows for more general applications, such as scheduling with precedence constraints, modeling metabolic pathways, and process management in operating systems; see a survey by Sotskov [Mathematics, 2020].

We initiate the systematic study of the parameterized complexity of mixed coloring. We focus on structural graph parameters that lie between cliquewidth and vertex cover, primarily with respect to the underlying undirected graph. Unlike classical coloring, which is fixed-parameter tractable (FPT) parameterized by treewidth or neighborhood diversity, we show that mixed coloring is W[1]-hard for treewidth and even paraNP-hard for neighborhood diversity. To utilize the directedness of arcs, we introduce and analyze natural generalizations of neighborhood diversity and cliquewidth to mixed graphs, and show that mixed coloring becomes FPT when parameterized by (the generalized) mixed neighborhood diversity. Further, we investigate how these parameters are affected if we add transitive arcs, which do not affect colorings. Finally, we provide tight bounds on the chromatic number of mixed graphs, generalizing known bounds on mixed interval graphs.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph coloring

Keywords and phrases Mixed Graphs, Coloring, Parameterized Complexity, Structural Graph Parameters

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.28

Related Version *Full Version*: `https://arxiv.org/abs/2604.15274` [26]

Acknowledgements This work was started at the workshop Homonolo 2024 in Nová Louka. We thank the organizers and the other participants.

1 Introduction

The problem of coloring graphs, i.e., assigning as few colors as possible to vertices of an undirected graph such that adjacent vertices receive distinct colors, is an old and extensively studied problem in graph theory with applications in various areas, such as scheduling, frequency assignment, and graph drawing [24]. An example of a scheduling problem that can be modeled using graph coloring is timetabling, where courses (vertices) need to be scheduled in certain time slots (colors) while respecting conflicts (edges), e.g., due to two courses having

© Antonio Lauerbach, Konstanty Junosza-Szaniawski, Marie Diana Sieper, and Alexander Wolff;
licensed under Creative Commons License CC-BY 4.0

the same teacher or students. These types of problems are referred to as chromatic scheduling problems. In real world applications, there are usually several additional constraints, such as certain courses having to be scheduled in specific time slots, certain teachers not being available at certain times, courses using multiple consecutive time slots, or a lecture having to be held before the corresponding exercises. Thus, there exist several extensions of coloring to accommodate these additional constraints, such as using list coloring to restrict courses to certain slots [9], or using interval coloring to enforce consecutive time slots [23]. To account for precedence constraints, for example, if a course has to be scheduled before another, Hansen et al. [20] added arcs to the graph, with the idea that the arc (u, v) expresses that u has to be scheduled before v. This leads to the notion of a *mixed graph* G, which consists of a set $V(G)$ of *vertices*, a set $E(G)$ of (undirected) *edges*, and a set $A(G)$ of (directed) *arcs*. A mixed graph combines the concepts of undirected and directed graphs. We require mixed graphs to be *simple*, i.e., without loops and without parallel edges or arcs.

A *k-coloring* $c\colon V(G) \to \{1, \ldots, k\}$ of a mixed graph G is *proper* if it holds for every edge $\{u, v\} \in E(G)$ that $c(u) \neq c(v)$ and it holds for every arc $(u, v) \in A(G)$ that $c(u) < c(v)$. We call the problem of deciding whether a mixed graph can be properly colored with k colors k-MixedColoring. As a mixed graph cannot be properly colored if it contains a directed cycle, we consider only mixed graphs without directed cycles. Apart from scheduling, mixed graph colorings have also been applied in graph drawing in order to compact layered orthogonal drawings [19]. See the survey by Sotskov [32] for more details and applications of mixed graph colorings.

Since MixedColoring generalizes the classical Coloring problem for undirected graphs, it is also NP-hard for three or more colors. As such, it is natural to study its parameterized complexity. However, to our knowledge, there has not yet been a systematic study of the parameterized complexity of MixedColoring. The only known results in this direction are an FPT-algorithm parameterized by the number of edges (i.e., not counting arcs) by Hansen et al. [20], with subsequent improvements by Damaschke [8], as well as an XP-algorithm parameterized by treewidth by Ries and de Werra [30].

Meanwhile, the parameterized complexity of Coloring has been extensively studied. On the positive side, results include the well-known dynamic program that runs in $\mathcal{O}^*(k^{\mathsf{tw}})$ time, where k is the number of colors, tw is the treewidth of the graph, and the $\mathcal{O}^*$-notation hides polynomial factors. Since treewidth bounds the chromatic number of undirected graphs, this implies that Coloring is FPT parameterized by treewidth. Lokshtanov et al. [27] even showed that under SETH (a hypothesis about the complexity of SAT with at most k literals per clause) there is no $\mathcal{O}^*((k-\varepsilon)^{\mathsf{tw}})$-time algorithm for any $\varepsilon > 0$, i.e., the $\mathcal{O}^*(k^{\mathsf{tw}})$ algorithm is probably optimal. Further results include Coloring being FPT parameterized by neighborhood diversity, as shown by Ganian [15]. On the negative side, Fomin et al. [13] showed that Coloring is W[1]-hard parameterized by cliquewidth. However, Coloring still admits an XP-algorithm parameterized by cliquewidth, as shown by Kobler and Rotics [22]. Further, applying Courcelle's theorem [4, 5], a famous meta-theorem that yields FPT-algorithms, it follows that Coloring is FPT parameterized by cliquewidth plus chromatic number.

Understanding the relationships between various structural graph parameters is crucial for analyzing the parameterized complexity of a problem. We say that a graph parameter α *(upper) bounds* another parameter β if there exists a computable function f such that for every (mixed) graph G, it holds that $\beta(G) \leqslant f(\alpha(G))$. If no such function exists, we say that α *does not (upper) bound* β. If α (upper) bounds β and β does not (upper) bound α, we say that α *strictly (upper) bounds* β. Meanwhile, if neither parameter bounds the other, we

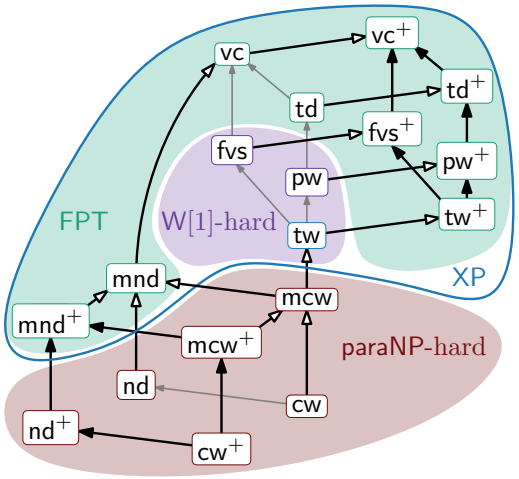


Figure 1 Overview of the hierarchy of structural parameters of mixed graphs (without directed cycles), and the parameterized complexity landscape for MixedColoring. Arrows represent strict (upper) bounds between parameters. Bold arrows indicate new relationships established in this work. White-tipped arrows highlight results that are not straightforward consequences of known results. Composite parameters are omitted for clarity.

Table 1 Our parameterized complexity results for MixedColoring including results for composite parameters. The complexity of other parameters can be inferred from the parameter hierarchy in Figure 1.

Parameter	Complexity	Reference
mcw + χ	FPT	Thm. 18
mnd	FPT	Thm. 28
nd + χ	FPT	Thm. 31
tw	XP	Cor. 20
fvs + pw	W[1]-hard	Thm. 23
mcw + Λ	paraNP-hard	Thm. 24
nd^+	paraNP-hard	Thm. 25

say that they are *incomparable*. Conversely, if two parameters bound each other, we say that they are *equivalent*. For example, since treewidth is bounded by vertex cover, this implies that any problem that is FPT parameterized by treewidth, such as Coloring, is also FPT parameterized by vertex cover.

For mixed graphs however, it is not immediately clear how to apply the existing parameters, as they are defined for undirected graphs. The approach taken in previous works is to consider the parameters w.r.t. the *underlying undirected graph*, i.e., the graph obtained by replacing all arcs with edges. The downside of this approach is that it neglects the directedness of arcs. In fact, Courcelle and Olariu [6] originally defined cliquewidth for undirected as well as for directed graphs, yielding a natural generalization for mixed graphs. Further, Fernau et al. [11] generalized neighborhood diversity to directed graphs by distinguishing between incoming and outgoing neighbors; this can also be generalized to mixed graphs.

Contribution. In order to better explore the parameterized complexity of MixedColoring, we first introduce and analyze the generalizations of neighborhood diversity and cliquewidth to mixed graphs, which we call mixed neighborhood diversity (mnd) and mixed cliquewidth (mcw), respectively. Since in MixedColoring we can assume that the input graph does not contain directed cycles, we only analyze these parameters for mixed graphs without directed cycles. We explore their relationships with existing parameters defined on the underlying undirected graph, such as vertex cover (vc), treedepth (td), pathwidth (pw), feedback vertex set (fvs), treewidth (tw), as well as neighborhood diversity (nd) and cliquewidth (cw); see the full version [26] for detailed definitions. Specifically, we show that mcw strictly bounds cw (Proposition 1) and is strictly bounded by tw (Proposition 2). Similarly, we show that mnd strictly bounds nd (Proposition 5) and is strictly bounded by vc (Proposition 6). Furthermore, we show that mnd strictly bounds mcw (Proposition 7), while mcw and nd are incomparable (Proposition 8).

Since arcs enforce precedence constraints on the colors of the vertices, *transitive arcs*, i.e., arcs between vertices connected by a directed path along other arcs, do not influence the properness of colorings of mixed graphs. Adding all transitive arcs (and removing any edges parallel to transitive arcs) results in the *transitive closure* G^+ of a mixed graph G (without directed cycles). We denote with α^+ the *transitive closure counterpart* of the parameter α, i.e., $\alpha^+(G) = \alpha(G^+)$. We show that vc, td, pw, fvs, and tw are strictly bounded by their transitive closure counterparts (Proposition 9), while for mcw and mnd the situation is inverted, with mcw and mnd strictly bounding mcw^+ and mnd^+, respectively (Proposition 12). Meanwhile, cw and nd are incomparable to their transitive closure counterparts (Proposition 13). The resulting hierarchy of parameters is visualized in Figure 1.

Furthermore, we also consider bounds for the *chromatic number* (χ) of mixed graphs, i.e., the minimum number of colors needed for a proper coloring. After providing some simple lower bounds, i.e., that the chromatic number of a mixed graph is lower-bounded by the chromatic number of the underlying undirected graph as well as by *maxrank* (Λ), the length of the longest directed path, we generalize and tighten a bound by Gutowski et al. [18] on the chromatic number of mixed interval graphs (Theorem 15). Based on this, we show that the chromatic number of a mixed graph is bounded by tw^+ as well as td and vc (Corollary 17).

Using the newly established parameter relationships, we analyze the parameterized complexity of MixedColoring in Section 3. The results are summarized in Table 1 and visualized in Figure 1. We conclude with an outlook on open problems in Section 4.

Coloring Directed Graphs. For directed graphs, Neumann-Lara [28] introduced the dichromatic number and the corresponding coloring, where each color class induces an acyclic subgraph. This problem, known as DigraphColoring [21], also generalizes coloring undirected graphs, as each edge can be replaced by two opposite arcs. Therefore, DigraphColoring is NP-hard. In contrast, MixedColoring is easy on directed graphs: if the graph is acyclic, greedily coloring vertices in topological order yields an optimal coloring. Otherwise, the graph cannot be properly colored. This difference arises due to the fact that DigraphColoring treats arcs very differently: in MixedColoring, there cannot be arcs between vertices of the same color, whereas in DigraphColoring there can be arcs between vertices of the same color, as long as these arcs do not form a directed cycle.

On directed acyclic graphs, MixedColoring, especially when considering the transitive closure, behaves somewhat similar to OrientedColoring [31], where, between each pair of color classes, all arcs must have the same orientation. However, contrary to MixedColoring, OrientedColoring does not forbid directed cycles.

Conventions. For positive integer k, we use $[k]$ as shorthand for $\{1, \ldots, k\}$. We say that a mixed graph G is a *partial orientation* of its underlying undirected graph. The full versions of proofs marked with a (clickable) star ($\star$) can be found in the full version of this paper [26].

2 Mixed Graph Parameters

We define the *mixed cliquewidth* of a mixed graph G, denoted by $\mathsf{mcw}(G)$, to be the minimum number of distinct labels needed to construct G using the following operations:

- creation of a new vertex with label i, denoted by i.
- disjoint union of two labeled mixed graphs, denoted by $\oplus$.
- adding an (undirected) edge between every vertex with label i and every vertex with label j, for $i \neq j$, denoted by $\eta_{i,j}$.

- adding a (directed) arc from every vertex with label i to every vertex with label j, for $i \neq j$, denoted by $\alpha_{i,j}$.
- renaming label i to label j, denoted by $\varrho_{i \to j}$.

These are exactly the operations used by Courcelle and Olariu [6] in their definitions of cliquewidth for undirected and directed graphs. Instead of using only $\eta_{i,j}$ for undirected graphs and only $\alpha_{i,j}$ for directed graphs, we allow both operations to be used in order to construct mixed graphs, thereby generalizing the concept of cliquewidth to mixed graphs. We call the sequence of such operations constructing a given mixed graph using ℓ labels a *mixed ℓ-expression*. For example, the directed path of length 4 is constructed by the mixed 3-expression $\alpha_{2,3}(\varrho_{3\to2}(\varrho_{2\to1}(\alpha_{2,3}(\alpha_{1,2}(1 \oplus 2) \oplus 3))) \oplus 3)$. As it turns out, mixed cliquewidth is not bounded by cliquewidth.

▶ **Proposition 1.** *For any mixed graph G (without directed cycles),* $\mathsf{cw}(G) \leqslant \mathsf{mcw}(G)$. *Furthermore,* cw *does not bound* mcw.

Proof. The first part follows directly from the definitions, as replacing all arc operations by edge operations in a mixed ℓ-expression constructing G yields an ℓ-expression constructing the underlying undirected graph of G.

Let $\mathrm{AG}(G)$ be the subgraph of G resulting from omitting all edges. It holds that $\mathsf{mcw}(\mathrm{AG}(G)) \leqslant \mathsf{mcw}(G)$, as by omitting the edge operations in a mixed ℓ-expression constructing G we obtain a mixed ℓ-expression for $\mathrm{AG}(G)$. Using this, we construct a family of mixed graphs $(G_\ell)_{\ell \geqslant 1}$ with $\mathsf{cw}(G_\ell) = 2$ and $\mathsf{mcw}(G_\ell) \geqslant \ell + 1$. We construct G_ℓ by orienting an $(\ell \times \ell)$-grid graph such that no directed cycles are created. As shown by Golumbic and Rotics [17], the cliquewidth of an $(\ell \times \ell)$-grid graph is $\ell + 1$ and thus $\mathsf{mcw}(G_\ell) \geqslant \mathsf{mcw}(\mathrm{AG}(G_\ell)) \geqslant \mathsf{cw}(\mathrm{AG}(G_\ell)) = \ell + 1$. Furthermore, between any pair of vertices of G_ℓ that are not connected by an arc, we add an edge. This results in the underlying undirected graph of G_ℓ being a complete graph, which has cliquewidth 2, and thus $\mathsf{cw}(G_\ell) = 2$. ◀

Courcelle and Olariu [6] showed that the cliquewidth of undirected as well as of directed graphs is bounded by treewidth. We can utilize their result for directed graphs to show that mixed cliquewidth (on mixed graphs without directed cycles) is also bounded by treewidth. To this end, we transform a mixed graph G without directed cycles into its *corresponding directed graph $D(G)$* by replacing each edge $\{u, v\}$ by two opposite arcs (u, v) and (v, u). Since mixed graphs without directed cycles cannot contain opposite arcs (as they would form a directed cycle of length two) this transformation is injective. We define the *directed cliquewidth* of a mixed graph G without directed cycles, denoted $\mathsf{dcw}(G)$, to be the (directed) cliquewidth of the corresponding directed graph $D(G)$. As we show now, mixed and directed cliquewidth are, while not equal, equivalent under the lens of parameterized complexity.

▶ **Proposition 2** (⋆). *For any mixed graph G (without directed cycles), it holds that* $\mathsf{dcw}(G) \leqslant \mathsf{mcw}(G) \leqslant 2\,\mathsf{dcw}(G) \leqslant 2(2^{\mathsf{tw}(G)+1} + 1)$. *Further, there exists a mixed graph $\tilde{G}$ such that* $\mathsf{dcw}(\tilde{G}) \neq \mathsf{mcw}(\tilde{G})$.

Idea. Converting a mixed ℓ-expression for G into a directed ℓ-expression for $D(G)$ is straightforward, as we can replace each edge operation by two opposite arc operations. The inverse requires a careful manipulation of the ℓ-expression for $D(G)$, to ensure that we can replace pairs of opposite arc operations by edge operations.

The partial orientation $\tilde{G}$ of the star $K_{1,2}$, where the center vertex is connected to one leaf by an edge and to the other leaf by an outgoing arc, has $\mathsf{dcw}(\tilde{G}) = 2$ and $\mathsf{mcw}(\tilde{G}) = 3$. ◀

In order to generalize neighborhood diversity to mixed graphs without directed cycles, we first need to define the different types of neighbors that occur in a mixed graph. Given a vertex v of a mixed graph G, the *outgoing neighbors* of v, denoted by $N^+(v)$, are all vertices u to which v has an *outgoing arc* (v,u). Analogously, the *incoming neighbors* of v, denoted by $N^-(v)$, are all vertices u from which v has an *incoming arc* (u,v). The *undirected neighbors* of v, denoted by $N^{\mathrm{u}}(v)$, are all vertices u which are connected to v by an edge $\{u,v\}$. Thus, the *neighborhood* of a vertex v, denoted by $N(v)$, is the union of its incoming, outgoing, and undirected neighbors.

The parameter neighborhood diversity is defined for undirected graphs via an equivalence relation on the vertices, where two vertices u,v are of the same type, denoted $u \sim_{\mathsf{nd}} v$, if they have the same neighbors, i.e., if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. We generalize this equivalence relation to mixed graphs by further distinguishing between incoming, outgoing, and undirected neighbors. Therefore, two vertices u,v of a mixed graph G are of the same *(mixed) type*, denoted $u \sim_{\mathsf{mnd}} v$, if they have the same in-, out-, and undirected neighbors, i.e., if $N^{\mathrm{u}}(u) \setminus \{v\} = N^{\mathrm{u}}(v) \setminus \{u\}$, $N^-(u) = N^-(v)$, and $N^+(u) = N^+(v)$.

▶ **Lemma 3** (⋆). *For any mixed graph G (without directed cycles), the relation $\sim_{\mathsf{mnd}}$ is an equivalence relation. Further, vertices of the same type induce an (undirected) clique or an independent set.*

We call the partition of the vertices into equivalence classes under $\sim_{\mathsf{mnd}}$ a *mixed neighborhood partition*. The *mixed neighborhood diversity* of a mixed graph G, denoted $\mathsf{mnd}(G)$, is the number of equivalence classes under $\sim_{\mathsf{mnd}}$. Note that this generalizes the definition of neighborhood diversity for undirected graphs. Furthermore, for mixed graphs without directed cycles, this is also a generalization of the definition of neighborhood diversity for directed graphs by Fernau et al. [11], as in directed graphs without directed cycles there are no opposite arcs. Note further that a mixed neighborhood partition can be obtained easily: it suffices to check, for each pair of vertices, whether they are of the same type.

As is the case for mixed cliquewidth and cliquewidth, the mixed neighborhood diversity can differ arbitrarily from the neighborhood diversity of the underlying undirected graph since each vertex in a directed path has to be of a different type, as we show now.

▶ **Lemma 4.** *Given a mixed graph G (without directed cycles) that contains a directed path of length ℓ, it holds that $\mathsf{mnd}(G) \geqslant \ell + 1$, with each vertex on the path being of a different type.*

Proof. We know from Lemma 3 that there cannot be an arc between two vertices of the same type. Let $\langle v_0, v_1, \ldots, v_\ell \rangle$ be a directed path of length ℓ in G. Suppose that there are two vertices v_i, v_j with $i < j$ that are of the same type. Since $v_{j-1} \in N^-(v_j)$, it follows that $v_{j-1} \in N^-(v_i)$. However, as $i \leqslant j-1$, it follows that there exists a directed path from v_i to v_{j-1} in G. Together with the arc (v_{j-1}, v_i), this yields a directed cycle, a contradiction to G containing no directed cycles. ◀

▶ **Proposition 5.** *For any mixed graph G (without directed cycles), $\mathsf{nd}(G) \leqslant \mathsf{mnd}(G)$. Furthermore, nd does not bound mnd.*

Proof. It follows directly from the definitions that two vertices of the same type w.r.t. mnd must also be of the same type w.r.t. nd. Indeed, if $u \sim_{\mathsf{mnd}} v$, it holds that

$$N(u) \setminus \{v\} = N^-(u) \cup N^+(u) \cup N^{\mathrm{u}}(u) \setminus \{v\} = N^-(v) \cup N^+(v) \cup N^{\mathrm{u}}(v) \setminus \{u\} = N(v) \setminus \{u\}$$

(as there cannot be any arc between u and v) and therefore $u \sim_{\mathsf{nd}} v$.

The inverse does not hold, as two vertices of the same type w.r.t. nd may differ in their in- or out-neighbors, as the following family $(G_\ell)_{\ell \geqslant 1}$ of graphs shows. The graph G_ℓ is a partial orientation of the complete graph K_ℓ obtained by orienting $\ell - 1$ edges such that the resulting mixed graph contains a directed Hamiltonian path. It holds that $\mathsf{nd}(G_\ell) = 1$ but, due to Lemma 4, $\mathsf{mnd}(G_\ell) \geqslant \ell$. ◀

Lampis [25] showed that neighborhood diversity is bounded by vertex cover, as the type of vertices not contained in the cover is determined uniquely by their neighbors (in the cover), resulting in a bound of $\mathsf{vc} + 2^{\mathsf{vc}}$. Since mixed neighborhood diversity further distinguishes between in-, out-, and undirected neighbors, we obtain the following bound.

▶ **Proposition 6.** *For any mixed graph G (without directed cycles),* $\mathsf{mnd}(G) \leqslant \mathsf{vc}(G) + 4^{\mathsf{vc}(G)}$.

Proof. Let C be a vertex cover of G of size $\mathsf{vc}(G)$. Since each vertex in $V(G) \setminus C$ only contains neighbors in C, there are at most $4^{\mathsf{vc}(G)}$ different types of vertices in $V(G) \setminus C$, as each vertex in C can be an in-, out-, or undirected neighbor, or not a neighbor at all. Further, there can be at most $\mathsf{vc}(G)$ different types of vertices in C, as C contains $\mathsf{vc}(G)$ vertices. Thus, the total number of types is bounded by $\mathsf{vc}(G) + 4^{\mathsf{vc}(G)}$. ◀

If a parameter does not bound nd (cw) on undirected graphs (or vice versa), it also does not bound mnd (mcw) (or vice versa), as it holds that mnd = nd (mcw = cw) on undirected graphs. Lampis [25] showed that nd bounds cw and that tw (and therefore cw) does not bound nd (nd is even incomparable to tw). It follows from this that mcw does not bound mnd. Using a proof analogous to Lampis, we further obtain the following.

▶ **Proposition 7** (⋆). *For any mixed graph G (without directed cycles),* $\mathsf{mnd}(G)+1 \geqslant \mathsf{mcw}(G)$. *Further,* mcw *does not bound* mnd.

Regarding the relationship between mcw and nd, we already know that mcw does not bound nd (due to cw not bounding nd). However, while nd bounds cw, it does not bound mcw. We show this by reusing the construction that we used to show that cw does not bound mcw (Proposition 1), as detailed in the following.

▶ **Proposition 8** (⋆). *The parameters* mcw *and* nd *are incomparable.*

Transitive Closure. Recall that an arc (u, v) is transitive if there exists a directed path from u to v using other arcs. The transitive closure G^+ of a mixed graph G is the mixed graph obtained by adding every transitive arc to G (and removing any resulting parallel edges). As transitively closed graphs are a subset of mixed graphs, all bounds between parameters on mixed graphs also apply between their transitive closure counterparts. Further, if a parameter α does not bound a parameter β on undirected graphs, then α^+ does not bound β^+, as it holds that $\alpha^+ = \alpha$ and $\beta^+ = \beta$ on undirected graphs. Thus, it remains to analyze the relationship between the parameters and their transitive closure counterparts.

All parameters considered in this paper, except for neighborhood diversity and cliquewidth, have the well-known property that they bound their value on any subgraph. Since the underlying graph of the transitive closure of a mixed graph is a subgraph of the underlying graph of the original graph, it follows that these parameters are bounded by their transitive closure counterparts.

▶ **Proposition 9** (⋆). *For every mixed graph G (without directed cycles) and parameter* $\alpha \in \{\mathsf{vc}, \mathsf{td}, \mathsf{fvs}, \mathsf{pw}, \mathsf{tw}\}$ *it holds that* $\alpha(G) \leqslant \alpha^+(G)$. *Furthermore, these parameters do not bound their transitive closure counterparts.*

Idea. The second part can be shown using a family of oriented stars, where half of the edges are oriented towards the center vertex and the other half are oriented away from the center vertex. ◀

However, for neighborhood diversity and cliquewidth (and their mixed variants), the situation is different. Contrary to the other parameters, these parameters are not bounded by their transitive closure counterparts, as shown in the following.

▶ **Proposition 10.** *The parameters* mnd, nd, mcw, *and* cw *are not bounded by their transitive closure counterparts.*

Proof. We first construct a family $(G_\ell)_{\ell \geqslant 1}$ of grid graphs where $2 \geqslant \mathsf{mcw}^+(G_\ell) \geqslant \mathsf{cw}^+(G_\ell)$ but $\mathsf{mcw}(G_\ell) \geqslant \mathsf{cw}(G_\ell) = \ell + 1$. The graph G_ℓ is a partial orientation of the $(\ell \times \ell)$-grid graph where $\ell^2 - 1$ edges are oriented such that G_ℓ contains a directed Hamiltonian path, i.e., a directed path of length $\ell^2 - 1$ containing all vertices. Golumbic and Rotics [17] showed that the cliquewidth of $(\ell \times \ell)$-grid graphs is $\ell + 1$. Together with Proposition 1 we obtain that $\mathsf{mcw}(G_\ell) \geqslant \mathsf{cw}(G_\ell) = \ell + 1$. As G_ℓ contains a directed Hamiltonian path, it follows that G_ℓ^+ is the *acylic tournament graph*, the directed graph with exactly one arc between each pair of vertices (and without directed cycles). Using the following claim, we obtain that $2 \geqslant \mathsf{mcw}^+(G_\ell) \geqslant \mathsf{cw}^+(G_\ell)$.

▷ Claim 11 (⋆). The mixed cliquewidth of an acyclic tournament graph is (at most) 2.

For neighborhood diversity, we construct a separate family $(G'_\ell)_{\ell \geqslant 1}$ based on tripartite graphs. The graph G'_ℓ consists of three independent sets $\{u_1, \ldots, u_\ell\}$, $\{v_1, \ldots, v_\ell\}$, and $\{w_1, \ldots, w_\ell\}$. We add arcs (u_i, v_j) and (v_i, w_j) for $i, j \in [\ell]$, such that every u has an outgoing arc to every v and every v has an outgoing arc to every w. To ensure that vertices of different sets are of different types, we further add three vertices u^*, v^*, and w^*. We then add edges $\{u^*, u_i\}$, $\{v^*, v_i\}$, and $\{w^*, w_i\}$ for all $i \in [\ell]$. Lastly, we add arcs (u_i, w_i) for $i \in [\ell]$. Thus, each vertex u_i has a distinct type, as does every vertex w_i. Meanwhile, all vertices v_i are of the same type. Further, the vertices u^*, v^*, and w^* have distinct types. This results in $\mathsf{mnd}(G'_\ell) = \mathsf{nd}(G'_\ell) = 2\ell + 4$. However, in the transitive closure, each vertex u_i has an outgoing arc to every vertex w_j and therefore all u_i are of the same type and all w_i are of the same type, resulting in $\mathsf{mnd}^+(G'_\ell) = \mathsf{nd}^+(G'_\ell) = 6$. ◀

In fact, both mixed neighborhood diversity and mixed cliquewidth bound their transitive closure counterparts, as shown in the following.

▶ **Proposition 12** (⋆). *For any mixed graph G (without directed cycles), it holds that* $\mathsf{mnd}^+(G) \leqslant \mathsf{mnd}(G)$ *and* $\mathsf{mcw}^+(G) \leqslant 4^{\mathsf{mcw}(G)} \cdot \mathsf{mcw}(G)$.

Idea. For mixed neighborhood diversity, it holds that two vertices u, v of the same type in G are also of the same type in G^+, as the set of vertices to (from) which u, v have outgoing (incoming) paths in G are equivalent.

For mixed cliquewidth, we show that given a mixed ℓ-expression constructing G, we can obtain a $(4^\ell \cdot \ell)$-expression constructing G^+. The idea is to keep track of the labels of vertices to which a given vertex v has outgoing paths and the labels of vertices from which it has incoming paths. This can be achieved by using 4^ℓ additional labels for each original label. These additional labels then allow us to add the transitive arcs, resulting in the claimed bound. ◀

The same does not hold for neighborhood diversity and cliquewidth, as these are in fact incomparable to their transitive closure counterparts, as the following shows.

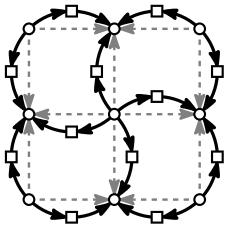


Figure 2 Example of the graph G_ℓ for $\ell = 3$. The grid vertices are depicted as disks and the arc vertices as squares. The grid vertices form an independent set in G_ℓ but induce a (3×3)-grid in G_ℓ^+ (the transitive arcs are depicted as dashed gray). The edges between arc vertices and grid vertices are omitted for visual clarity.

▶ **Proposition 13.** *The parameters* nd *and* cw *are incomparable to their transitive closure counterparts.*

Proof. We already know from Proposition 10 that these parameters are not bounded by their transitive closure counterparts. Thus, it remains to show that they do not bound their transitive closure counterparts. For this, we construct a family of graphs $(G_\ell)_{\ell \geqslant 2}$ such that $\mathsf{nd}(G_\ell) = \mathsf{cw}(G_\ell) = 2$ while $\mathsf{nd}^+(G_\ell) \geqslant \mathsf{cw}^+(G_\ell) \geqslant \ell + 1$.

The graph G_ℓ consists of ℓ^2 *grid vertices* and $2\ell(\ell-1)$ *arc vertices* and is constructed such that the grid vertices form an independent set in G_ℓ but induce an $(\ell \times \ell)$-grid in G_ℓ^+. This is achieved through the usage of the arc vertices, which connect two grid vertices adjacent in G_ℓ^+ with a directed path, thus ensuring that the transitive arc is present in G_ℓ^+. Further, each arc vertex is connected to all grid vertices via edges, except for those to which it is connected via an arc. This ensures that all arc vertices are of the same type and that all grid vertices are of the same type, resulting in $\mathsf{nd}(G_\ell) = \mathsf{cw}(G_\ell) = 2$; see Figure 2 for an example of the construction for $\ell = 3$. As shown by Golumbic and Roctis [17], the cliquewidth of an $(\ell \times \ell)$-grid is $\ell + 1$. Furthermore, it is easy to see that the cliquewidth of a graph bounds the cliquewidth of induced subgraphs. Thus, it follows that $\mathsf{nd}^+(G_\ell) \geqslant \mathsf{cw}^+(G_\ell) \geqslant \ell + 1$. ◀

Chromatic Number. Clearly, any proper coloring of a mixed graph is also a proper coloring of the underlying undirected graph and thus the chromatic number of the underlying undirected graph lower bounds the chromatic number of the mixed graph. Recall that the maxrank (Λ) of a mixed graph is the length of the longest directed path in the graph. For a given vertex v, we define the *inrank*, denoted as $\lambda^-(v)$, as the length of the longest directed path ending in v. Furthermore, as vertices along a directed path must be colored increasingly, every vertex has a color that is at least its inrank. Thus, it holds that the chromatic number bounds the maxrank. We summarize these insights in the following proposition.

▶ **Proposition 14.** *For any mixed graph G (without directed cycles), $\chi_{\mathrm{u}}(G) \leqslant \chi(G)$ and $\Lambda(G) \leqslant \chi(G)$.*

For undirected graphs, several (upper) bounds on the chromatic number are known, such as treewidth and maximum degree. However, these bounds do not generalize to mixed graphs, as a simple directed path of length ℓ has treewidth 1 and maximum degree 2, but chromatic number $\ell + 1$. Gutowski et al. [18, Theorem 9 & Proposition 10] showed that for mixed interval graphs it holds that $\chi \leqslant (\Lambda + 1) \cdot \omega$, where ω is the clique number, and that this bound is asymptotically tight. They achieved this by introducing the concept of *layering*, which is a partition of the vertex set into $\Lambda + 1$ layers $\langle L_0, \ldots, L_\Lambda \rangle$ such that the layer L_i contains all vertices of inrank i. In a layering, each arc is oriented towards the higher layer,

as it must hold for every arc (u, v) that $\lambda^-(u) < \lambda^-(v)$. It follows that each subgraph $G[L_i]$ induced by a layer L_i is an undirected graph. Thus, we can iteratively color each layer with disjoint color sets, resulting in a proper coloring of the entire graph as formalized in the following theorem.

▶ **Theorem 15** (⋆). *For any mixed graph G (without directed cycles) and its corresponding layering $\langle L_0, L_1, \ldots, L_\Lambda \rangle$, it holds that $\chi(G) \leqslant \sum_{i=0}^{\Lambda} \chi_u(G[L_i])$. Further, this bound is tight.*

Regarding the transitive closure, note that $\Lambda^+ = \Lambda$ and $\chi^+ = \chi$, as adding transitive arcs neither changes the length of the longest directed path nor the validity of a coloring. Thus, we obtain the following bounds on maxrank.

▶ **Proposition 16** (⋆). *For any mixed graph G (without directed cycles), it holds that $\Lambda(G) + 1 \leqslant \mathsf{mnd}^+(G)$, $\Lambda(G) \leqslant \mathsf{tw}^+(G)$, $\Lambda(G) \leqslant 2^{\mathsf{td}(G)} - 2$, and $\Lambda(G) \leqslant 2\mathsf{vc}(G)$.*

As the chromatic number of a subgraph is a lower bound for the chromatic number of the entire graph, it follows from Theorem 15 and Proposition 16 that any parameter that bounds the chromatic number of undirected graphs as well as the maxrank also bounds the chromatic number of mixed graphs.

▶ **Corollary 17** (⋆). *The chromatic number of mixed graphs is bounded by tw^+ as well as td and vc. In fact, it even holds for every mixed graph G that $\chi(G) \leqslant 2\mathsf{vc}(G) + 1$.*

The resulting hierarchy is visualized in Figure 1.

3 Parameterized Complexity of MixedColoring

Recall that a problem is *fixed-parameter tractable* (FPT) with respect to a parameter k if it can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time, where f is a computable function and n is the size of the input. A problem is *slice-wise polynomial* (XP) with respect to k if it can be solved in $\mathcal{O}(n^{f(k)})$ time (where f and n are as above). Clearly, FPT $\subseteq$ XP. Note that, given an FPT- or XP-algorithm for k-(Mixed)Coloring that is parameterized by the number of colors, k, we can easily obtain an FPT- or XP-algorithm parameterized by the chromatic number. Therefore, we state our results with respect to the parameter chromatic number (if the runtime is FPT or XP parameterized by the number of colors).

Courcelle's theorem yields for any problem that can be expressed in monadic second-order logic on graphs an FPT-algorithm parameterized by tree- or even cliquewidth (depending on the logic variant used). Like cliquewidth, monadic second-order logic on graphs extends intuitively to mixed graphs, and expressing MixedColoring as a monadic second-order formula is straightforward. However, to our knowledge, Courcelle's theorem has not yet been entirely proven for mixed graphs, with Arnborg et al. [1] only proving the theorem for treewidth but not for cliquewidth. Still, by converting our mixed graphs into directed graphs as in Proposition 2, we can apply the directed version of Courcelle's theorem to obtain the following result.

▶ **Theorem 18** (⋆). *The problem MixedColoring is* FPT *parameterized by mixed cliquewidth plus chromatic number.*

While Courcelle's theorem yields an FPT-algorithm for MixedColoring, the resulting algorithm is quite impractical. For example, the runtime contains huge constants depending on the length of the formula, i.e., the number of colors, which cannot be bounded by an

elementary function unless $\mathsf{P} = \mathsf{NP}$, as shown by Frick and Grohe [14]. Since polynomials are elementary functions, this also implies that this algorithm is not an XP-algorithm parameterized solely by tree- or cliquewidth.

Another Coloring algorithm can also be generalized to MixedColoring: it is well known [7, Theorem 7.9] that Coloring, given a tree-decomposition of width tw, can be solved in $\mathcal{O}^*(k^{\mathsf{tw}})$ time via a dynamic program. By exchanging the checks for a proper coloring with checks for a proper mixed coloring (i.e., additionally checking that no arc is violated), MixedColoring can be solved within the same time bound, as we now summarize.

▶ **Theorem 19.** *Given a mixed graph and a tree-decomposition of width* tw*, the problem k-MixedColoring can be solved in* $\mathcal{O}^*(k^{\mathsf{tw}})$ *time.*

As an optimal tree-decomposition can be computed in FPT-time w.r.t. treewidth [2] and the chromatic number is bounded by the number of vertices, this yields FPT and XP algorithms for MixedColoring, as summarized in the following.

▶ **Corollary 20.** *MixedColoring is* FPT *parameterized by treewidth plus chromatic number and* XP *parameterized solely by treewidth.*

Note that this algorithm is an improvement over the XP-algorithm developed by Ries and de Werra [30], which runs in $\mathcal{O}(n^{2\mathsf{tw}+4}m^{\mathsf{tw}+2})$ time, where m is the number of edges and n the number of vertices.

We cannot hope for an FPT-algorithm parameterized by treewidth alone, as we show in the following that MixedColoring is W[1]-hard w.r.t. pathwidth and feedback vertex set, which are parameters that bound treewidth. We achieve this via a parameterized reduction from ListColoring, a generalization of Coloring where each vertex can only be assigned a color from a given list. While Fellows et al. [10] only showed that ListColoring is W[1]-hard w.r.t. treewidth, their reduction actually even yields W[1]-hardness w.r.t. vertex cover, as we show in the following lemma.

▶ **Lemma 21** (⋆)**.** *ListColoring parameterized by vertex cover is* W[1]*-hard.*

To reduce ListColoring to k-MixedColoring, where k is the total number of distinct colors in the ListColoring instance, we enforce the color list of each vertex by adding adequate edges to newly inserted directed paths of vertex-length k. This yields the following.

▶ **Lemma 22** (⋆)**.** *There exists a parameterized reduction w.r.t. pathwidth as well as w.r.t. feedback vertex set from ListColoring to MixedColoring.*

Combining the W[1]-hardness of ListColoring with the parameterized reduction to MixedColoring we obtain the following theorem.

▶ **Theorem 23.** *MixedColoring is* W[1]*-hard w.r.t. feedback vertex set as well as w.r.t. pathwidth.*

Recall that Coloring is FPT with respect to neighborhood diversity [15] and XP with respect to cliquewidth [22]. In contrast to that, we now show that MixedColoring is paraNP-hard with respect to both of these parameters, even in more restricted settings.

▶ **Theorem 24** (⋆)**.** *MixedColoring is paraNP-hard w.r.t. mixed cliquewidth plus maxrank.*

Idea. Given a set of strings $\mathcal{S}$ over the binary alphabet $\{0, 1\}$ and a positive integer k, the problem ShortestSuperstring asks whether there is a string of length at most k that is a superstring of each string in $\mathcal{S}$. ShortestSuperstring is NP-hard [29].

Figure 3 (a) The mixed graph resulting from an instance of ShortestSuperstring with strings $\{01, 100, 11\}$ and shortest superstring 1010. (b) The construction resulting from splitting the path corresponding to the string 100 at each vertex and connecting the resulting vertices to distinct cliques to enforce that they receive the same color in a proper k-coloring.

We reduce from ShortestSuperstring by constructing a mixed graph G such that a proper k-coloring of G corresponds to a length-k superstring of $\mathcal{S}$. We achieve this by creating a directed path P_S for each string S in $\mathcal{S}$ such that the vertices of the path correspond to the characters of the string; we add edges such that vertices corresponding to different characters cannot receive the same color. See Figure 3a for an example. Thus, the color of a vertex corresponds to the position of the corresponding character in the superstring, and the directions of the paths enforce that the characters of each string appear in the correct order.

To ensure a constant maxrank, we then split each directed path P_S at every vertex into two vertices, with one vertex obtaining the incoming and the other the outgoing arc. We then connect each new pair of vertices to a distinct $(k-1)$-clique, thus enforcing that the two vertices receive the same color in any proper k-coloring; see Figure 3b. Therefore, regarding proper k-colorings, they can be treated as a single vertex, while the maxrank of the resulting graph is 1, due to all directed paths having been split. Furthermore, the resulting graph has a mixed cliquewidth of at most 6. ◀

▶ **Theorem 25.** *MixedColoring is paraNP-hard parameterized by neighborhood diversity, even if considering the neighborhood diversity of the transitive closure.*

Proof. We reduce from the following problem, which is NP-hard [16, SS9].

Problem:	PrecedenceConstrainedScheduling
Input:	Sets T_1, T_2 of unit-length tasks, a partial order $\prec$ on $T_1 \cup T_2$, an integer D.
Question:	Can the tasks be scheduled with makespan D on two parallel machines M_1 and M_2 such that the tasks in T_1 are assigned to M_1, the tasks in T_2 are assigned to M_2, and all precedence constraints are respected (i.e., if $t \prec t'$, then t must finish before t' starts)?

Given an instance $(T_1, T_2, \prec, D)$ of PrecedenceConstrainedScheduling, we create a $4D$-MixedColoring instance G such that each block of four consecutive colors corresponds to one time unit on the machines.

First, we create a directed path P consisting of $4D$ vertices $\langle p_1, \ldots, p_{4D} \rangle$, which we will use to ensure that each task is scheduled on its assigned machine. Note that in any proper $4D$-coloring, the path P has the same unique coloring, with each vertex p_i receiving the color i. For each task $t \in T_1 \cup T_2$, we create two *task vertices*, the *start vertex* v_t^- and the *end vertex* v_t^+, connected via the arc (v_t^-, v_t^+). We denote with $[i]_k$ the set of integers that are congruent to i modulo k. For each start vertex v_t^- of a task t in T_1, we add an edge to

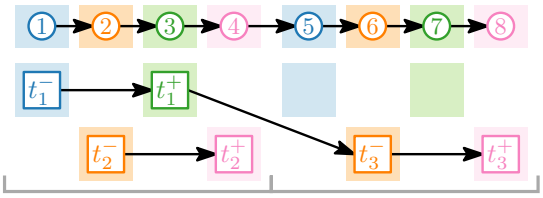


Figure 4 The mixed graph resulting from an instance of PrecedenceConstrainedScheduling with $T_1 = \{t_1\}$ and $T_2 = \{t_2, t_3\}$ as well as precedence constraints $t_1 \prec t_3$ and deadline 2. Four consecutive colors correspond to one time unit on the machines, and the colored slots correspond to the colors that the task vertices of the corresponding type can receive. The edges are omitted for visual clarity.

each vertex p_i of the path P with $i \notin [1]_4$. As a result, v_t^- receives a color in $[1]_4$ in every proper $4D$-coloring of G. Analogously, we add edges between each end vertex v_t^+ of a task t in T_1 and each vertex p_i of the path P with $i \notin [3]_4$ to ensure that v_t^+ receives a color in $[3]_4$. For tasks in T_2, we similarly add edges to ensure that the start vertices receive colors in $[2]_4$ and the end vertices receive colors in $[4]_4$. The precedence constraints are enforced by adding arcs between the corresponding tasks, i.e., for each $t, t' \in T_1 \cup T_2$ with $t \prec t'$ we add the arc $(v_t^+, v_{t'}^-)$. Lastly, it remains to ensure that no two tasks are scheduled at the same time on the same machine. We achieve this by adding (undirected) edges such that the underlying undirected graph induced by the task vertices forms a clique. An example of the resulting graph is shown in Figure 4.

▷ Claim 26 (⋆). The resulting graph G has a proper $4D$-coloring if and only if there is a schedule of the tasks on their assigned machines that respects the precedence constraints and finishes by time D.

▷ Claim 27 (⋆). The transitive closure of G has a neighborhood diversity of 8.

As these claims show, the resulting MixedColoring instance is equivalent to the original PrecedenceConstrainedScheduling instance, with the neighborhood diversity of the transitive closure being 8. Thus, MixedColoring is paraNP-hard w.r.t. neighborhood diversity, even when considering the neighborhood diversity of the transitive closure. ◀

The picture changes if we parameterize by *mixed* neighborhood diversity.

▶ **Theorem 28.** *MixedColoring is* FPT *parameterized by mixed neighborhood diversity.*

Proof. Let G be a mixed graph. We can assume w.l.o.g. that each type of G induces a clique, as all vertices of a type inducing an independent set can be merged into a single vertex. This is due to the fact that they have the same neighborhoods and therefore can be assigned the same color. Let $\mathcal{S} = \{C_1, \ldots, C_{\mathsf{mnd}}\}$ be the set of types of G, where each C_i is a clique in G. In the following, we write $\{C_i, C_j\} \in E(G)$ and $(C_i, C_j) \in A(G)$ to express that there is an undirected edge or a directed arc, respectively, between the types C_i and C_j in G. For a coloring c of G and a set $U \subseteq V(G)$, let $c(U) = \{c(u) : u \in U\}$. If c is proper, then, for every $(C_i, C_j) \in A(G)$, all colors in $c(C_i)$ are smaller than all colors in $c(C_j)$. We define a *type-endpoint preorder* p as a function that assigns, to each type C, two integers $p^-(C), p^+(C) \in [\ell]$ (with $p^-(C) < p^+(C)$ and $\ell \leqslant 2\,\mathsf{mnd}$), which we call the *type endpoints* of C. We say that a coloring c *corresponds* to a type-endpoint preorder p if there is a sequence $\langle c_1, \ldots, c_\ell \rangle$ of ℓ ascending (not necessarily consecutive) colors such that, for each type C, the set $c(C)$ is contained in the halfopen interval $[c_{p^-(C)}, c_{p^+(C)})$. We

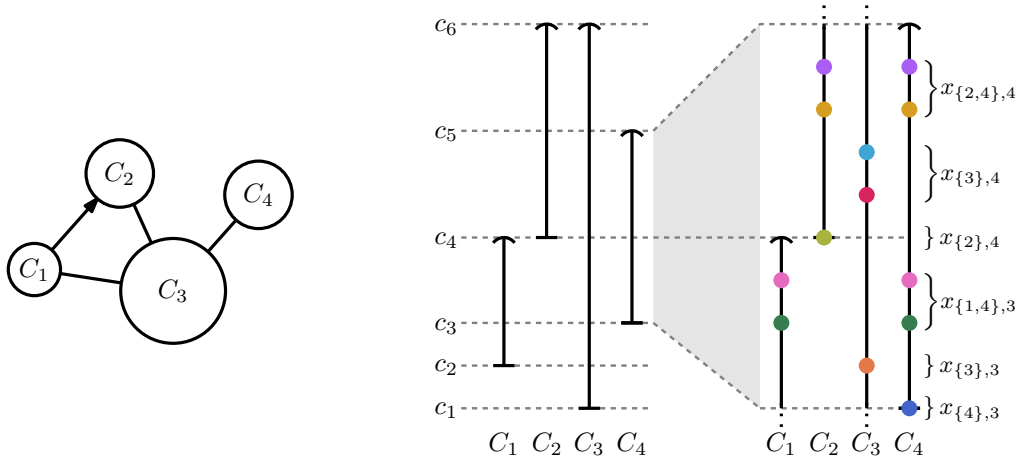


Figure 5 A mixed graph with four types $C_1, \dots, C_4$ and a preorder with $\ell = 6$ type endpoints. Since it holds for its only arc (C_1, C_2) that $p^+(C_1) = c_4 = p^-(C_2)$, the preorder is proper. On the right, we depict a part of a coloring corresponding to the preorder, together with the variables of the ILP. Note that most of the variables are zero, either due to a constraint of the ILP, such as for $x_{\{1\},4}$ or $x_{\{3,4\},3}$, or simply due to the fact that there are no colors shared exclusively by the given set of types in the given interval, such as for $x_{\{1\},3}$.

say that a (type-endpoint) preorder p is *proper* if, for each arc $(C_i, C_j) \in A(G)$, it holds that $p^+(C_i) \leqslant p^-(C_j)$. An example of a proper preorder and a corresponding coloring is shown in Figure 5.

▷ Claim 29 (⋆). Given a proper (type-endpoint) preorder p, any coloring that corresponds to p does not violate any arcs. Further, any proper coloring corresponds to at least one proper preorder.

In total, there are $2\,\mathsf{mnd}$ type endpoints and thus at most $(2\,\mathsf{mnd})!$ orders of these endpoints. Since type endpoints can be equal, there are at most $(2\,\mathsf{mnd})! \cdot 2^{2\,\mathsf{mnd}} \in 2^{\mathcal{O}(\mathsf{mnd} \log \mathsf{mnd})}$ (proper) preorders. Therefore, we can enumerate all proper preorders in FPT-time w.r.t. mixed neighborhood diversity. We can thus check whether G is k-colorable by checking, for each proper preorder p, whether there exists a proper k-coloring of G that corresponds to p. Since we know that such a coloring does not violate any arcs, it suffices to check whether it does not violate any edges. This problem has similarities to Coloring, differing in that we additionally require the colors within each type to adhere to the given preorder. For Coloring, Ganian [15] provided an FPT-algorithm w.r.t. neighborhood diversity based on an ILP. In the following, we adapt his ILP to additionally respect a given proper preorder p.

Since we only need to decide whether there exists a k-coloring c of G that corresponds to the given proper preorder p, we do not need an objective function. For an overview of the ILP, see the full version [26]. We have two kinds of variables. First, for every $i \in [\ell]$, we introduce an integer variable $c_i \in [k]$ to obtain a sequence $\langle c_1, \dots, c_\ell \rangle$ of colors. We enforce that these colors form an ascending sequence (by adding, for each $i \in [\ell - 1]$, the constraint $c_i + 1 \leqslant c_{i+1}$). Second, for each $i \in [\ell - 1]$ and each subset $\mathcal{S}' \subseteq \mathcal{S}$, we introduce an integer variable $x_{\mathcal{S}',i} \in \{0, \dots, k\}$, which represents the number of colors in the interval $[c_i, c_{i+1})$ that are shared exclusively by the types in $\mathcal{S}'$. Formally, we want that $x_{\mathcal{S}',i} = |\{d \in [c_i, c_{i+1}) \mid (\forall C \in \mathcal{S}' \colon d \in c(C)) \wedge (\forall C \in \mathcal{S} \setminus \mathcal{S}' \colon d \notin c(C))\}|$, where $c(C)$ is the set of colors assigned to the vertices of type C by the coloring c; see Figure 5 for an example. For each $i \in [\ell - 1]$, we need to ensure that the ILP does not assign too many colors in

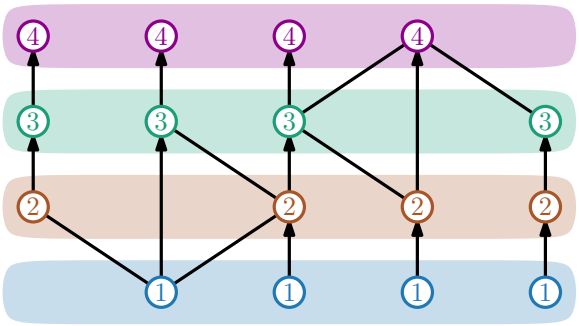


Figure 6 A uniquely 4-colorable mixed graph where no color class is a maximal independent set.

the interval $[c_i, c_{i+1})$. We achieve this by adding the constraint $\sum_{\mathcal{S}' \subseteq \mathcal{S}} x_{\mathcal{S}',i} \leqslant c_{i+1} - c_i$. Moreover, we want the coloring to correspond to the given preorder p. To achieve this, for each type C, we first ensure that no color $d < c_{p^-(C)}$ is assigned to C (by adding the constraint $\sum_{i=1}^{p^-(C)-1} \sum_{\mathcal{S}' \subseteq \mathcal{S}\colon C \in \mathcal{S}'} x_{\mathcal{S}',i} = 0$). Second, we ensure that enough colors are assigned to C in the interval formed by its two type endpoints (by adding the constraint $\sum_{i=p^-(C)}^{p^+(C)-1} \sum_{\mathcal{S}' \subseteq \mathcal{S}\colon C \in \mathcal{S}'} x_{\mathcal{S}',i} = |C|$). Third, we ensure that no color $d \geqslant c_{p^+(C)}$ is assigned to C (by adding the constraint $\sum_{i=p^+(C)}^{\ell-1} \sum_{\mathcal{S}' \subseteq \mathcal{S}\colon C \in \mathcal{S}'} x_{\mathcal{S}',i} = 0$). It remains to ensure that no two types connected by an edge share a color. To this end, for each $i \in [\ell - 1]$ and each set $\mathcal{S}' \subseteq \mathcal{S}$ that contains types C and C' with $\{C, C'\} \in E(G)$, we add the constraint $x_{\mathcal{S}',i} = 0$.

▷ Claim 30 (⋆). For a given proper preorder p, there is a feasible solution to the constructed ILP if and only if there is a proper k-coloring of G that corresponds to p.

It is well known [12, Theorem 9.19] that an ILP can be solved in FPT-time w.r.t. the number of variables. Hence, we can solve the ILP in FPT-time w.r.t. mixed neighborhood diversity as it uses $\mathcal{O}(2^{\mathsf{mnd}} \cdot \mathsf{mnd})$ variables. Therefore, by enumerating all proper preorders and checking for each preorder whether a corresponding proper k-coloring exists, we can decide in FPT-time w.r.t. mixed neighborhood diversity whether G is k-colorable. ◀

Furthermore, while MixedColoring is paraNP-hard w.r.t. neighborhood diversity, it becomes FPT if we use the chromatic number as an additional parameter.

▶ **Theorem 31.** *MixedColoring is* FPT *parameterized by neighborhood diversity plus chromatic number.*

Proof. It was shown by Christofides [3] that there exists an optimal coloring of an (undirected) graph where one color class forms a maximal independent set. However, this property does not hold for mixed graphs; see Figure 6. Fortunately, we can strengthen the property by only considering maximal independent sets of vertices with inrank 0, i.e., vertices without incoming arcs, resulting in the following formula for the chromatic number.

▷ Claim 32 (⋆). For every (non-empty) mixed graph G there exists an optimal coloring of G where the first color class forms a maximal independent set in the subgraph induced by the vertices of inrank 0. Thus, for every mixed graph G it holds that $\chi(G) = 0$ if G is empty and $\chi(G) = \min_I \chi(G - I) + 1$ otherwise, where the minimum is taken over all maximal independent sets I in the subgraph induced by the vertices of inrank 0.

Furthermore, we can bound the number of maximal independent sets in the subgraph induced by vertices of inrank 0 by neighborhood diversity and clique number, as the following shows.

▷ Claim 33 (⋆). For every (non-empty) mixed graph G it holds that the number of maximal independent sets in the subgraph induced by the vertices of inrank 0 is at most $(\omega(G)+1)^{\mathsf{nd}(G)}$.

We implement the recursive formula for the chromatic number as a branching algorithm for MixedColoring as follows: Given a mixed graph G and a number of colors k, we branch on all maximal independent sets I in the subgraph induced by the vertices of inrank 0 and recursively check whether $G - I$ is $(k-1)$-colorable until we either reach the empty graph or run out of colors. Thus, the recursion depth is bounded by k. As in each step we branch into at most $(\omega(G)+1)^{\mathsf{nd}(G)}$ subproblems, this results in an overall runtime of $\mathcal{O}^*((\omega(G)+1)^{\mathsf{nd}(G)\cdot k})$, which is FPT w.r.t. neighborhood diversity plus chromatic number, as $\chi(G) \geqslant \omega(G)$. ◀

4 Open Problems

While we defined mixed neighborhood diversity and mixed cliquewidth only for mixed graphs without directed cycles, these parameters can also be defined for mixed graphs with directed cycles. For mixed cliquewidth the definition remains the same, while for mixed neighborhood diversity the definition can be adapted by integrating the restrictions for opposite arcs introduced by Fernau et al. [11]. The relationships of these more general parameters should be analogous; most proofs can be generalized directly since they do not depend on the requirement that mixed graphs do not contain directed cycles. We remark that it seems likely that Courcelle's theorem extends to (general) mixed cliquewidth, which would facilitate its application to other problems on mixed graphs. Further, we think that it would be interesting to investigate how graph parameters and problem complexities change if we remove all transitive arcs from input graphs. Preliminary computational experiments indicate that MixedColoring algorithms benefit from such a data reduction step. Lastly, while we explored the parameterized complexity of MixedColoring, exact (non-parameterized) algorithms for MixedColoring remain largely unexplored (except for ILP-based approaches).

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Nicos Christofides. An algorithm for the chromatic number of a graph. *Comput. J.*, 14(1):38–39, 1971. doi:10.1093/COMJNL/14.1.38.

4 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

5 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2012. doi:10.1017/cbo9780511977619.

6 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1–3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Peter Damaschke. Parameterized mixed graph coloring. *J. Comb. Optim.*, 38(2):362–374, 2019. doi:10.1007/S10878-019-00388-Z.

9 Dominique de Werra. Restricted coloring models for timetabling. *Discrete Math.*, 165-166:161–170, 1997. doi:10.1016/S0012-365X(96)00208-7.

10 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/J.IC.2010.11.026.

11 Henning Fernau, Florent Foucaud, Kevin Mann, Utkarsh Padariya, and Rajath Rao K. N. Parameterizing path partitions. *Theor. Comput. Sci.*, 1028:115029, 2025. doi:10.1016/J.TCS.2024.115029.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

13 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

14 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004. doi:10.1016/J.APAL.2004.01.007.

15 Robert Ganian. Using neighborhood diversity to solve hard problems. ArXiv report, 2012. arXiv:1201.3091.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

17 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Foundat. Comput. Sci.*, 11(3):423–443. doi:10.1142/S0129054100000260.

18 Grzegorz Gutowski, Konstanty Junosza-Szaniawski, Felix Klesen, Paweł Rzążewski, Alexander Wolff, and Johannes Zink. Coloring and recognizing mixed interval graphs. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC)*, volume 283 of *LIPIcs*, pages 36:1–36:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ISAAC.2023.36.

19 Grzegorz Gutowski, Florian Mittelstädt, Ignaz Rutter, Joachim Spoerhase, Alexander Wolff, and Johannes Zink. Coloring mixed and directional interval graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *30th International Symposium on Graph Drawing and Network Visualization (GD)*, volume 13764 of *LNCS*, pages 418–431. Springer, 2022. doi:10.1007/978-3-031-22203-0_30.

20 Pierre Hansen, Julio Kuplinsky, and Dominique de Werra. Mixed graph colorings. *Math. Methods Oper. Res.*, 45(1):145–160, 1997. doi:10.1007/BF01194253.

21 Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Digraph Coloring and Distance to Acyclicity. *Theory Comput. Syst.*, 68(4):986–1013, 2024. doi:10.1007/s00224-022-10103-x.

22 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Appl. Math.*, 126(2–3):197–221, 2003. doi:10.1016/S0166-218X(02)00198-1.

23 Marek Kubale. Interval vertex-coloring of a graph with forbidden colors. *Discrete Math.*, 74(1–2):125–136, 1989. doi:10.1016/0012-365X(89)90204-5.

24 Marek Kubale, editor. *Graph Colorings*, volume 352 of *Contemporary Mathematics*. AMS, Providence, RI, 2004. doi:10.1090/conm/352.

25 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37. doi:10.1007/s00453-011-9554-x.

26 Antonio Lauerbach, Konstanty Junosza-Szaniawski, Marie Diana Sieper, and Alexander Wolff. The parameterized complexity of coloring mixed graphs. ArXiv report, 2026. arXiv:2604.15274.

27 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

28 Víctor Neumann-Lara. The dichromatic number of a digraph. *J. Combin. Theory Ser. B*, 33(3):265–270, 1982. doi:10.1016/0095-8956(82)90046-6.

29 Kari-Jouko Räihä and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theor. Comput. Sci.*, 16:187–198, 1981. doi:10.1016/0304-3975(81)90075-X.

30 Bernard Ries and Dominique de Werra. On two coloring problems in mixed graphs. *Eur. J. Comb.*, 29(3):712–725, 2008. doi:10.1016/J.EJC.2007.03.006.

31 Éric Sopena. Homomorphisms and colourings of oriented graphs: An updated survey. *Discrete Math.*, 339(7):1993–2005, 2016. doi:10.1016/j.disc.2015.03.018.

32 Yuri N. Sotskov. Mixed graph colorings: A historical review. *Math.*, 8(3):385:1–24, 2020. doi:10.3390/math8030385.

Bichromatic Classifications of Points Using Strips

Jaegun Lee
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Chaeyoon Chung
Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

Hee-Kap Ahn
Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Republic of Korea

Abstract

Given a set of n points in the plane, each colored either blue or red, we study the problem of finding a strip that separates the blue points from the red points. Specifically, we consider the following two variants: (1) locating a strip that contains no red points while maximizing the number of blue points within the strip, and (2) locating a strip that contains all blue points while minimizing the number of red points within the strip. For variant (1), we present an $O(n^2)$-time algorithm, improving upon the previously best $O(n^2 \log n)$-time result. We also show that this running time is optimal under the standard 3SUM conjecture. We also give an output-sensitive algorithm with running time $O(k_{\mathsf{opt}} n \log n)$ that returns a strip, where k_{opt} is the number of blue points not contained within the strip in an optimal solution. We extend our results to the case of up to t parallel strips, obtaining an $O(n^2 \log n)$-time algorithm. For variant (2), an optimal $\Theta(n \log n)$-time algorithm is known for $t = 1$. We show 3SUM-hardness for $t = 2$ and give an $O(n^2)$-time algorithm. For any $t \geqslant 3$, we present an $O(n^2 \log n)$-time algorithm.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Bichromatic Classification, Separation, Strip, Duality

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.29

Funding This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00219980), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2019-II191906, Artificial Intelligence Graduate School Program (POSTECH)) and (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)).

1 Introduction

Given a set of points in the plane, each colored either blue or red, we aim to identify a simple geometric region that separates the two colors as well as possible. In this paper, we focus on the case where the separation region is a *strip*, defined as a closed region bounded by two parallel lines of arbitrary orientation.

This model represents a fundamental classification task in which blue points represent positive instances, while red points represent negative instances. There is a fair amount of work for achieving a *perfect* bichromatic separation, wherein the two colors are distinctly separated without any errors. This is done using simple geometric shapes such as a line (or a collection of lines), a strip, a wedge, and a circle [20, 21, 27].

In many real-world datasets, however, it is often not possible to achieve a perfect separation by a low-complexity shape due to factors such as noise, overlapping distributions, or outliers. In such cases, one typically optimizes a trade-off between two types of classification errors:

© Jaegun Lee, Chaeyoon Chung, and Hee-Kap Ahn;
licensed under Creative Commons License CC-BY 4.0

false positives, where negative instances are incorrectly classified as positive, and *false negatives*, where positive instances are incorrectly classified as negative. The problems addressed in this paper correspond to two particularly important settings along this trade-off. In applications where admitting a negative instance is unacceptable, the objective is to enforce zero false positives and maximize the number of correctly classified positive instances. Conversely, in applications where missing a positive instance is unacceptable, the objective is to enforce zero false negatives and minimize the number of admitted negative instances.

We first study the following problem, which corresponds to the zero false positives setting.

▶ **Problem 1** (MaxBlue)**.** Given a set $P = B \cup R$ of n points in the plane, find a strip that contains the maximum number of points from set B while excluding any points from set R.

The MaxBlue problem enforces a strict feasibility constraint on red points. A complementary perspective is to require the separating region contains no red point while maximizing the number of covered blue points. This addresses settings where omitting a positive instance is unacceptable, leading to complete coverage of B and minimizing red points. This results in the MinRed variant, which corresponds to the zero false negatives setting.

▶ **Problem 2** (MinRed)**.** Given a set $P = B \cup R$ of n points in the plane, find a strip that contains all points of B while minimizing the number of points of R contained in the strip.

Motivation for using strips. The strip model is a natural and powerful geometric representation of separation. Unlike a single separating line, which represents a zero-width decision boundary, a strip provides a *thickened* boundary that captures a tolerance margin around a linear separator. This model represents situations in which the target set must lie within a bounded corridor that remains free of extraneous or adversarial points. Such a representation is essential when the target data corresponds to a physical object, a high-confidence cluster, or a region that must be certified safe within a specified geometric width [19]. Optimizing the placement of such a strip allows one to quantify the structural purity of a dataset and determine how much perturbation can be tolerated before separation becomes impossible. In this sense, strip separability serves as a robust geometric analogue of margin-based classification.

Applications. This geometric formulation arises naturally in several application domains requiring high-precision spatial reasoning. In metrology and industrial design, strip containment models tolerance-zone analysis: a manufactured component must lie within an allowable corridor while avoiding forbidden regions occupied by other parts [33]. In robotics and motion planning, strip separation captures narrow-passage navigation problems, where a robot must follow a collision-free linear trajectory that accounts for its physical width while avoiding obstacles. In pattern recognition and machine learning, strip separation corresponds to margin-based classification, where maximizing the width of a separating strip improves robustness and generalization [32]. Similarly, in geographic information systems and urban planning, strips model buffer zones that must fully contain protected regions while remaining disjoint from hazardous or restricted areas. These examples illustrate that strips provide a minimal yet expressive geometric primitive for modeling safety margins, tolerances, and robustness constraints.

Generalization to Multiple Strips. Furthermore, we consider natural generalizations of MaxBlue and MinRed, in which up to t parallel strips are allowed for the separation, for a fixed integer t. Extending from a single strip to multiple parallel strips is both natural and essential for modeling structured data in which target points occur in several roughly parallel

groups separated by gaps or forbidden regions. Such patterns arise in applications like crop-row detection [22], multi-lane traffic modeling [18], and circuit layout verification [23]. A single strip cannot capture these interleaved configurations, whereas multiple strips allow the model to represent periodicity and spacing, yielding a more expressive separability that maximizes covered target points while strictly excluding forbidden ones. This formulation is a discrete analogue of maximum-margin separation for nonconvex distributions, related to classical geometric separability and covering problems [26, 31].

▶ **Problem 3** (t-MaxBlue)**.** Given a set $P = B \cup R$ of n points in the plane and an integer t, find t parallel strips whose union contains the maximum number of points from set B while excluding any points from set R.

▶ **Problem 4** (t-MinRed)**.** Given a set $P = B \cup R$ of n points in the plane and an integer t, find t parallel strips whose union contains all points of B while minimizing the number of points from set R contained in the union.

See Figure 1 for an illustration of each problem.

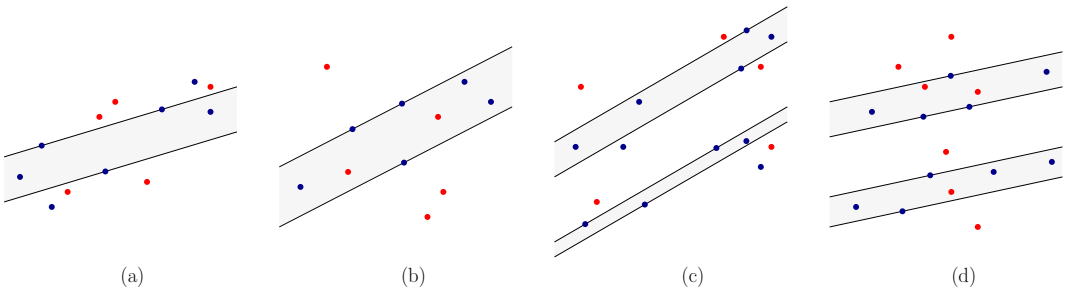


Figure 1 Optimal separation strips for (a) MaxBlue, (b) MinRed, (c) 2-MaxBlue, (d) 2-MinRed.

1.1 Related Work

In the classical approach to geometric bichromatic separation, the objective is to achieve perfect separation between the two classes. This involves finding a simple geometric shape that separates the two colors without any errors. The simplest case is perfect separation by a line, which can be decided in $O(n)$ time using linear programming [20]. Perfect separation by a circle can be decided in $O(n)$ time [27]. Hurtado et al. [21] gave $O(n \log n)$-time algorithms for perfect separation by regions bounded by two lines, including a strip, wedge, and double wedge. Moreover, several of these problems have $\Omega(n \log n)$ lower bounds [3].

The study of bichromatic separation with errors has been motivated in the context of situations where perfect separation does not exist. In the false negative setting, which minimizes the number of red points within a separation region that contains all blue points, Glazenburg et al. [17] gave $O(n \log n)$-time algorithms for finding an optimal separation by a halfplane, strip, or wedge, and an $O(n^2)$-time algorithm for finding an optimal separation by a double wedge. Bitner et al [6] studied optimal separation by a disk and gave an $O(n^2 \log n)$-time algorithm. The variant using an axis-parallel square or rectangle has also been considered [24].

Fewer results are known for the complementary false positive setting, which maximizes the number of blue points within a separation region that contains no red point. Eckstein et al. [13] showed NP-hardness for separation by a box for blue and red points in d-dimensional space when the dimension d is part of the input. For the planar case ($d = 2$), the problem

can be solved in $O(n \log^3 n)$ time [4]. Glazenburg et al. [17] gave algorithms for optimal separation of blue and red points in the plane using various geometric shapes, including halfplanes, strips, wedges, and double wedges. Their optimal strip separation algorithm, which is exactly MaxBlue, runs in $O(n^2 \log n)$ time.

Few results are known about natural generalizations concerning the union of multiple geometric objects. Seara [28] studied perfect separation using two strips, two parallel strips, and two wedges. Maji and Sadhu [24] studied MinRed using two interior-disjoint axis-parallel rectangles (or squares) and two interior-disjoint convex polygons.

Some works consider alternative objective functions for bichromatic separation. For instance, Everett et al. [15] and Glazenburg et al. [17] minimized the total number of misclassified points (uncovered blue points plus red points within the separation region). Dobkin et al. [12] maximized the discrepancy, the difference between the numbers of blue and red points within the region.

1.2 Our Results

We present exact algorithms for MaxBlue, t-MaxBlue and t-MinRed. We give an $O(n^2)$-time algorithm for MaxBlue, improving upon the previous best $O(n^2 \log n)$-time result by Glazenburg et al. [17]. We show that this running time is optimal under the standard 3SUM conjecture (Section 4).

The time complexity of our algorithm depends primarily on the total number of points n. However, a perfect separation strip can be computed in $\Theta(n \log n)$ time [28] if one exists, revealing a significant computational gap when the number k_{opt} of blue points not contained in an optimal strip for MaxBlue is small. Considering the practical motivations and applications of this problem, the excluded points typically represent outliers or noise within the dataset. In such contexts, k_{opt} is expected to be substantially smaller than n, making an output-sensitive approach highly advantageous. We present an output-sensitive algorithm for MaxBlue with running time $O(nk_{\mathsf{opt}} \log n)$. Consequently, our algorithm is expected to run in near-linear time for such practical inputs, making the output-sensitive approach particularly valuable.

To ensure robustness, we run the output-sensitive algorithm within a time budget of $O(n^2)$; if it does not terminate within this bound, we invoke the $O(n^2)$-time algorithm. This hybrid approach yields an overall complexity of $O(\min\{n^2, nk_{\mathsf{opt}} \log n\})$ for MaxBlue (Section 5), effectively bridging the gap between theoretical worst-case analysis and practical efficiency.

Finally, we consider the natural generalization for the MaxBlue problem and MinRed problem, in which up to t parallel strips are allowed for the separation. We present an $O(n^2 \log n)$-time algorithm for the t-MaxBlue problem, for any value of t (Section 6). For the t-MinRed, we present an $O(n^2 \log n)$-time algorithm for $t \geqslant 3$ (Section 7). When $t = 2$, we show that the problem is 3SUM-hard, and give an $O(n^2)$-time algorithm. A summary of our results and previous works is shown in Table 1.

Table 1 A summary of previous works and our results. † indicates that the problem is 3SUM-hard.

	MaxBlue	t-MaxBlue	t-MinRed
Previous works	$O(n^2 \log n)$ [17]	-	-
Ours	$\boldsymbol{O(n^2)}$† (Sec. 4) $\boldsymbol{O(nk_{\mathsf{opt}} \log n)}$ (Sec. 5)	$\boldsymbol{O(n^2 \log n)}$ (Sec. 6)	$\boldsymbol{O(n^2)}$† for $t = 2$ $\boldsymbol{O(n^2 \log n)}$ for $t \geqslant 3$ (Sec. 7)

2 Preliminaries

For any two points u, w in the plane, we denote the line segment connecting them by uw.

Duality

We employ the standard point-line duality transform [5]. For any point $p = (a, b) \in \mathbb{R}^2$, its dual is the line $p^* : y = ax - b$. Conversely, the dual of a nonvertical line $\ell : y = ax - b$ is the point $\ell^* = (a, b)$. This duality transform is order preserving: p lies on ℓ if and only if ℓ^* lies on p^*, and p lies above ℓ if and only if ℓ^* lies above p^*. For a set of points P, we define $P^* = \{p^* \mid p \in P\}$. A strip bounded by two parallel lines in the primal plane corresponds to the vertical line segment connecting their dual points in the dual plane (see Figure 2(a–b)).

▶ **Observation 5.** *Let σ be a strip bounded by parallel lines ℓ_1 and ℓ_2. A point $p \in \mathbb{R}^2$ lies in σ if and only if its dual line p^* intersects the vertical line segment $\ell_1^*\ell_2^*$ connecting ℓ_1^* and ℓ_2^*.*

Proof. Without loss of generality, assume ℓ_1 lies above ℓ_2. A point p lies in σ if and only if it is on or below ℓ_1 and on or above ℓ_2. Since ℓ_1 and ℓ_2 are parallel, they share the same slope, which implies that their dual points ℓ_1^* and ℓ_2^* have the same x-coordinate; thus, the segment $\ell_1^*\ell_2^*$ is vertical. By the order-preserving property, $p \in \sigma$ is equivalent to ℓ_1^* lying on or below p^* and ℓ_2^* lying on or above p^*, which means p^* intersects the segment $\ell_1^*\ell_2^*$. ◀

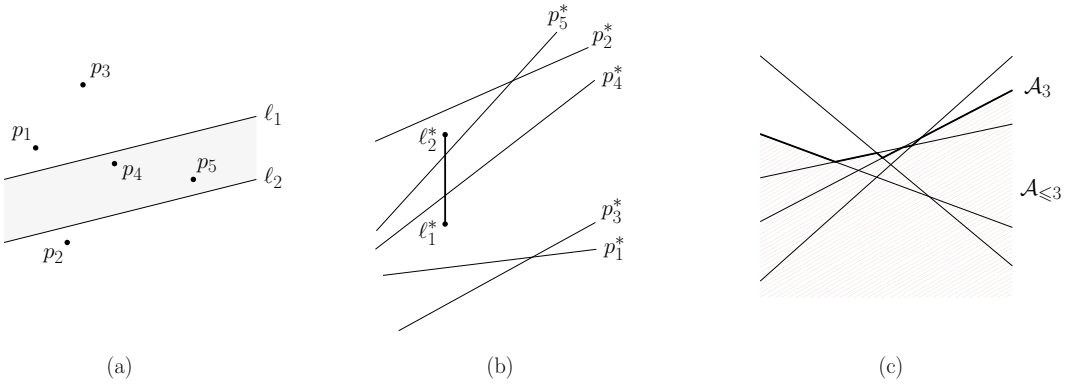


Figure 2 (a) Points $p_1, \ldots, p_5$ and a strip σ bounded by lines ℓ_1 and ℓ_2. (b) For $i = 1, \ldots, 5$, $p_i \in \sigma$ if and only if p_i^* intersects the vertical segment $\ell_1^*\ell_2^*$. (c) For an arrangement of a set L of lines, $\mathcal{A}_3(L)$ is shown in bold. The set of edges contained in the shaded region forms $\mathcal{A}_{\leqslant 3}(L)$.

Arrangement

Let L be a set of n lines in the plane. The arrangement $\mathcal{A}(L)$ is the subdivision of the plane induced by L. The *level* of a point $p \in \mathbb{R}^2$ is the number of lines in L strictly below p. Since the level is constant along any edge of $\mathcal{A}(L)$, the level of an edge is well-defined. The *k-level* of $\mathcal{A}(L)$, denoted by $\mathcal{A}_k(L)$, consists of the edges with level k; this set forms an x-monotone chain [25]. We denote the union of the i-levels for all $0 \leqslant i \leqslant k$ by $\mathcal{A}_{\leqslant k}(L)$ (see Figure 2(c) and [8, 29]).

Let $f_k(L)$ denote the complexity of $\mathcal{A}_k(L)$. It is known that $f_k(L) = O(nk^{1/3})$ [11] and $f_k(L) = n2^{\Omega(\sqrt{\log k})}$ [30]. $\mathcal{A}_k(L)$ can be computed in $O((n + f_k(L)) \log n)$ time [7] or in expected $O(n \log n + nk^{1/3})$ time [8]. In contrast, the complexity of $\mathcal{A}_{\leqslant k}(L)$ is $\Theta(nk)$ [2, 10, 14]. Everett et al. [15] computed $\mathcal{A}_{\leqslant k}(L)$ in $O(n \log n + kn)$ time. Furthermore, Chan [9] proposed an expected $O(n \log n + k^2 \log n)$-time algorithm to minimize a linear objective function over $\mathcal{A}_{\leqslant k}(L)$.

3 Problem Formulation in the Dual Plane

We reformulate the problems with the dual plane. For an input set $P = B \cup R$ of points in the primal plane, we consider its corresponding dual set $P^* = B^* \cup R^*$. As established in Observation 5, a strip in the primal plane maps to a vertical segment in the dual plane. By extension, a set of t parallel strips in the primal plane corresponds to t vertical segments sharing a common x-coordinate in the dual plane. Thus, each problem can be expressed in terms of dual-plane geometry as follows.

▶ **Problem 6** (MaxBlue*). Given a set $P^* = B^* \cup R^*$ of n lines in the plane, find a vertical segment that intersects the maximum number of lines in B^* while intersecting no lines in R^*.

▶ **Problem 7** (MinRed*). Given a set $P^* = B^* \cup R^*$ of n lines in the plane, find a vertical segment that intersects all lines in B^* while minimizing the number of lines in R^* that intersect it.

▶ **Problem 8** (t-MaxBlue*). Given a set $P^* = B^* \cup R^*$ of n lines in the plane and an integer t, find t vertical segments with a common x-coordinate whose union intersects the maximum number of lines in B^* while intersecting no lines in R^*.

▶ **Problem 9** (t-MinRed*). Given a set $P^* = B^* \cup R^*$ of n lines in the plane and an integer t, find t vertical segments with a common x-coordinate whose union intersects all lines in B^* while minimizing the number of lines in R^* intersected by the union.

4 $O(n^2)$-time algorithm for MaxBlue and 3SUM-hardness

We present an $O(n^2)$-time algorithm for MaxBlue, improving upon the previous best $O(n^2 \log n)$-time algorithm [17]. We also show that this running time is optimal under the standard 3SUM conjecture.

▶ **Theorem 10.** *We can solve MaxBlue and MaxBlue* in $O(n^2)$ time.*

Proof. We can solve MaxBlue by solving MaxBlue*. Let $P^* = B^* \cup R^*$ be a set of n lines in the plane. Our algorithm is rather simple. It computes a vertical segment in each cell C of $\mathcal{A}(R^*)$ that intersects the most lines in B^*. Among all such segments, it reports the one that intersects the most. An optimal vertical segment intersects no lines in R^*, which ensures the algorithm's correctness. However, achieving the optimal running time is challenging.

We describe the algorithm in detail and analyze its time complexity. Consider the arrangements $\mathcal{A}(R^*)$ and $\mathcal{A}(P^*)$. Note that once $\mathcal{A}(P^*)$ is computed, the arrangement $\mathcal{A}(R^*)$ is implicitly available. Let C be a cell of $\mathcal{A}(R^*)$. Imagine we sweep C with a vertical line ℓ from left to right, starting from the leftmost vertex of C. Let c_B be the number of lines in B^* whose intersection with ℓ lies in the interior of C. Observe that $c_B = 0$ at the leftmost vertex of C, and c_B changes only when ℓ hits an intersection point x between a line h in B^* and the boundary of C, excluding the leftmost and rightmost vertices: c_B increases by one if h enters C at x, and c_B decreases by one if h leaves C at x. Thus, we can maintain c_B by traversing the upper and lower boundary chains of C from their leftmost vertex within the arrangement $\mathcal{A}(P^*)$, and determine the position of ℓ at which c_B is maximized. We repeat this procedure for every cell in $\mathcal{A}(R^*)$.

The arrangement $\mathcal{A}(P^*)$ can be computed in $O(n^2)$ time. For each cell C in $\mathcal{A}(R^*)$, the procedure above takes $O(n_C)$ time, where n_C denotes the number of vertices of $\mathcal{A}(P^*)$ lying on the boundary of C. Let $d(v)$ be the degree of a vertex v in $\mathcal{A}(P^*)$, that is, number

of edges incident to v in $\mathcal{A}(P^*)$. Observe that each vertex v of $\mathcal{A}(P^*)$ appears on the boundaries of at most $d(v)$ cells of $\mathcal{A}(R^*)$. Thus, the total time complexity is bounded by $\sum_{C\in\mathcal{A}(R^*)} O(n_C) = \sum_{v\in\mathcal{A}(P^*)} O(d(v)) = O(n^2)$. ◀

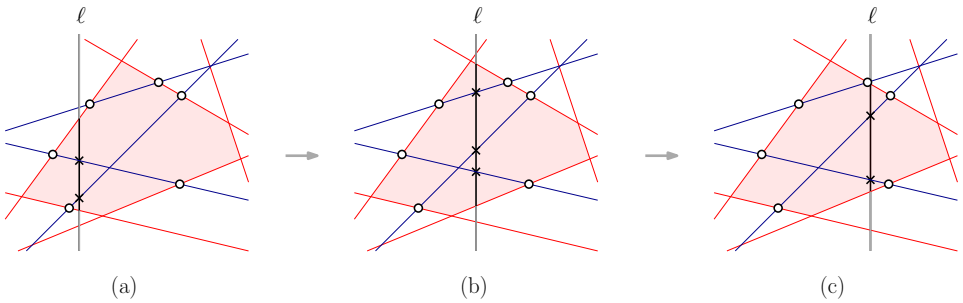


Figure 3 Sweeping a cell in $\mathcal{A}(R^*)$ with a vertical line ℓ. (a) $c_B = 2$, (b) $c_B = 3$, and (c) $c_B = 2$.

We show that MaxBlue is 3SUM-hard by a reduction from the GEOMBASE problem. Thus, there is no strongly sub-quadratic algorithm for MaxBlue unless the 3SUM problem can be solved in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$. The GEOMBASE problem is defined as follows: Given n points in the plane with integer coordinates on three horizontal lines $y = 0$, $y = 1$ and $y = 2$, determine whether there exists a non-horizontal line containing three of the points. It is known that the GEOMBASE problem is 3SUM-hard [16].

▶ **Theorem 11.** *MaxBlue and MaxBlue* are 3SUM-hard.*

Proof. We use a reduction from the GEOMBASE problem. Consider n blue points in the plane with integer x-coordinates larger than 0 and at most M on three horizontal lines $y = 0$, $y = 1$ and $y = 2$, for some integer M. For each blue point $p = (\alpha, 1)$, we place two red points $q = (\alpha + 1/(2\sqrt{4+M^2}), 1)$ and $q' = (\alpha - 1/(2\sqrt{4+M^2}), 1)$. We show that no line $\ell : (b-a)y - 2x + 2a = 0$ passing through any two blue points $(a, 0), (b, 2)$ intersects qq' unless it passes through p. If ℓ does not pass through p, $|2\alpha - a - b| \geqslant 1$ since α, a, and b are all integers. Then the distance between p and ℓ is $|2\alpha - a - b|/\sqrt{4+(a-b)^2} \geqslant |2\alpha - a - b|/\sqrt{4+M^2} \geqslant 1/\sqrt{4+M^2} > 1/(2\sqrt{4+M^2})$. See Figure 4(a).

Similarly, for each $i = 0, 2$, we place two red points q, q' on $y = i$ for each blue point (β, i) such that no line passing through any two blue points $(a, j), (b, k)$ for $\{j, k\} = \{0, 1, 2\} \setminus \{i\}$ intersects qq' unless it passes through (β, i). Thus, we have n blue points and $2n$ red points in total. The reduction takes $O(n)$ time.

If a non-horizontal line contains three blue points, there is a strip containing three blue points while excluding any red points. If no such line exists, no strip contains three blue points while excluding any red points. See Figure 4(b). ◀

5 $O(nk_{\text{opt}} \log n)$-Time Algorithm for MaxBlue

We present an $O(nk_{\text{opt}} \log n)$-time algorithm for MaxBlue, where k_{opt} denotes the number of blue points not covered by an optimal strip. In the dual plane, we are given a set $P^* = B^* \cup R^*$ of n lines in the plane, and our goal is to find a vertical segment that intersects the maximum number of lines in B^* while intersecting no lines in R^*. For convenience, we will refer to the lines in B^* as blue lines and the lines in R^* as red lines.

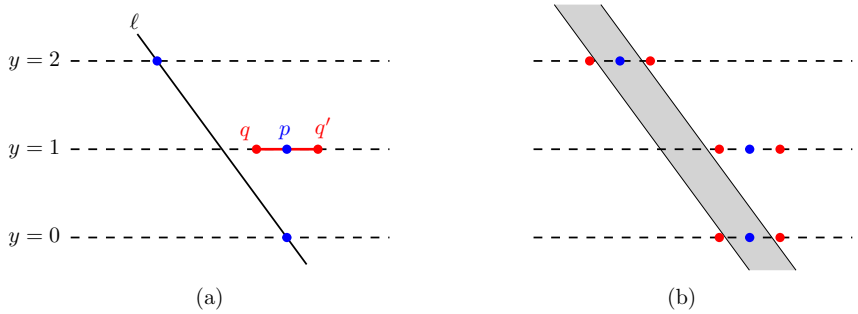


Figure 4 (a) We place two red points q, q' on $y = 1$ for each blue point p on $y = 1$ such that no line passing through any two blue points $(a, 0), (b, 2)$ intersects qq' unless it passes through p. (b) A non-horizontal line contains three blue points if and only if there is a strip containing three blue points while excluding any red points.

We focus on vertical segments whose endpoints lie on blue lines. If an endpoint does not lie on a blue line, we can shorten the segment until each endpoint does, without changing the set of blue lines it intersects.

In Section 5.1, we compute a value M such that $k_{\mathsf{opt}}/2 \leqslant M < 2k_{\mathsf{opt}}$. Then, in Section 5.2, we determine k_{opt} and compute an optimal vertical segment using binary search. We assume $k_{\mathsf{opt}} \neq 0$, as the case $k_{\mathsf{opt}} = 0$ can be detected in $O(n \log n)$ time [21].

5.1 Reducing a Search Space

We find a value M such that $k_{\mathsf{opt}}/2 \leqslant M < 2k_{\mathsf{opt}}$ using exponential search. For each $k = 2^0, 2^1, \ldots$, we check if there exists a vertical segment with exactly k blue lines above it and exactly k blue lines below it, and no red lines intersecting it. Let k' be the smallest value of k for which the exponential search succeeds. We have $k_{\mathsf{opt}} \leqslant 2k'$ because there exists a vertical segment that intersects no red lines and misses exactly $2k'$ blue lines. The search fails for $k'/2$. We show $k_{\mathsf{opt}} > k'/2$. Suppose $k_{\mathsf{opt}} \leqslant k'/2$. Then there exists a vertical segment with k_1 blue lines above and k_2 blue lines below, intersecting no red line, such that $\max\{k_1, k_2\} \leqslant k_{\mathsf{opt}} \leqslant k'/2$. Shortening it gives a vertical segment with $k'/2$ blue lines above it and $k'/2$ blue lines below it, intersecting no red lines. This contradicts that the search fails for $k'/2$. Thus, $k_{\mathsf{opt}} > k'/2$.

To perform this exponential search, we observe that a vertical segment with exactly k blue lines above it and exactly k blue lines below it has one endpoint on $\mathcal{A}_k(B^*)$ and the other on $\mathcal{A}_{|B|-k-1}(B^*)$. We denote such a segment by $\Upsilon_k(x)$, where x is its x-coordinate. Imagine moving $\Upsilon_k(x)$ from left to right while keeping its endpoints on these two x-monotone chains. As we increase x, we maintain the number of red lines intersected by $\Upsilon_k(x)$ and check if there exists a position x' such that $\Upsilon_k(x')$ intersects no red lines.

▶ **Lemma 12.** *For $0 \leqslant i < j < |B|$, suppose that the points in $\mathcal{A}_i(B^*) \cap R^*$ and $\mathcal{A}_j(B^*) \cap R^*$ are given in sorted order of their x-coordinates. We can decide whether there exists a vertical segment intersecting no red lines, with one endpoint on $\mathcal{A}_i(B^*)$ and the other on $\mathcal{A}_j(B^*)$ in $O(|R^*| + |\mathcal{A}_i(B^*) \cap R^*| + |\mathcal{A}_j(B^*) \cap R^*|)$ time.*

Proof. Let L be a value smaller than the smallest x-coordinate among the points in $(\mathcal{A}_i(B^*) \cap R^*) \cup (\mathcal{A}_j(B^*) \cap R^*)$. Any vertical segment lying on or to the left of $x = L$ with one endpoint on $\mathcal{A}_i(B^*)$ and the other on $\mathcal{A}_j(B^*)$ intersects the same set of red lines. See Figure 5.

We sweep the plane with a vertical line ℓ, starting at $x = L$ and moving to the right, and count the number N_r of red lines intersected by the vertical segment whose endpoints are $\mathcal{A}_i(B^*) \cap \ell$ and $\mathcal{A}_j(B^*) \cap \ell$. At $x = L$, we compute N_r in $O(|R^*|)$ time.

As ℓ moves to the right, N_r changes only when ℓ passes through a point in $(\mathcal{A}_i(B^*) \cap R^*) \cup (\mathcal{A}_j(B^*) \cap R^*)$. Therefore, given these intersection points in increasing order of their x-coordinates, we can update N_r while sweeping and determine whether there exists a position at which the segment intersects no red lines in $O(|\mathcal{A}_i(B^*) \cap R^*| + |\mathcal{A}_j(B^*) \cap R^*|)$ time. ◀

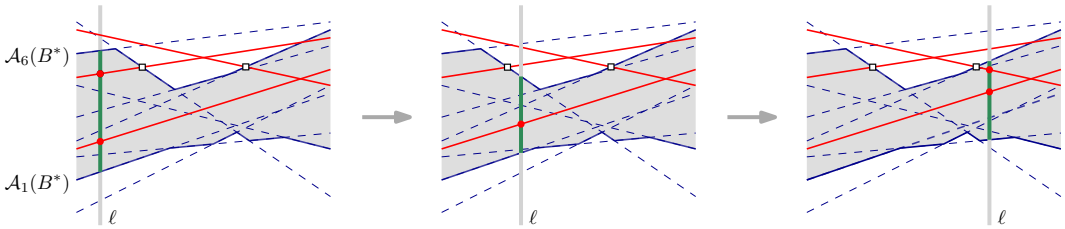


Figure 5 Three snapshots of the algorithm in Lemma 12 for $i = 1$ and $j = 6$.

Let k be an integer with $0 \leqslant k < |B|$. We show that $\mathcal{A}_k(B^*) \cap R^*$ and $\mathcal{A}_{|B|-k-1}(B^*) \cap R^*$ can be computed and sorted in $O(nk \log n)$ time.

▶ **Lemma 13.** *We can compute a collection of $k + 1$ edge-disjoint concave chains whose union is $\mathcal{A}_{\leqslant k}(B^*)$ in $O(nk^{1/3} \log n)$ time.*

Proof. We use a standard algorithm to decompose $\mathcal{A}_{\leqslant k}(B^*)$ into $O(k)$ concave chains as in [1, 9]. Chan [9] showed that this can be done in the same time we construct $\mathcal{A}_k(B^*)$. Thus, the chains can be constructed in $O(nk^{1/3} \log n)$ time [7, 11]. ◀

▶ **Lemma 14.** *$\mathcal{A}_{\leqslant k}(B^*) \cap R^*$ has size $O(nk)$ and can be computed in $O(nk \log n)$ time.*

Proof. We first decompose $\mathcal{A}_{\leqslant k}(B^*)$ into $k + 1$ concave chains in $O(nk^{1/3} \log n)$ time using Lemma 13. By construction, every vertex of each chain lies on $\mathcal{A}_k(B^*)$. Therefore, each chain has complexity $O(nk^{1/3})$. Then, for every pair consisting of a chain and a line in R^*, we compute all intersection points between them. Since each chain is concave, a line intersects it at most twice, and these intersections can be found in $O(\log n)$ time. As there are $O(k)$ chains and $O(n)$ lines, the total number of intersections is $O(nk)$, and all intersections can be computed in $O(nk \log n)$ time. See Figure 6. ◀

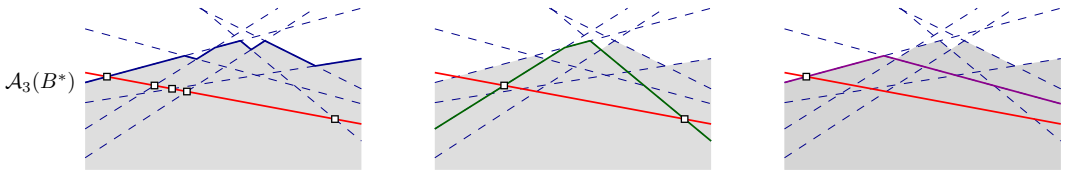


Figure 6 (a) For a set B^* of 7 blue lines, a red line intersects $\mathcal{A}_{\leqslant 3}(B^*)$ in five points. (b-c) After decomposing $\mathcal{A}_{\leqslant 3}(B^*)$ into concave chains, a red line intersects each concave chain in at most two points.

Let $Q = \mathcal{A}_{\leqslant k}(B^*) \cap R^*$, which can be computed in $O(nk \log n)$ time by Lemma 14. Observe that $\mathcal{A}_k(B^*) \cap R^* = \mathcal{A}_k(B^*) \cap Q$. We can compute $\mathcal{A}_k(B^*)$ in $O(nk^{1/3} \log n)$ time [7, 11]. After sorting the points of Q in increasing order of their x-coordinates, we traverse the chain $\mathcal{A}_k(B^*)$ from left to right and, for each point of Q, check whether it lies on $\mathcal{A}_k(B^*)$. Since both sequences are processed in increasing order of their x-coordinates, this can be done in linear time in the size of Q and $\mathcal{A}_k(B^*)$. Consequently, we can identify all points in $\mathcal{A}_k(B^*) \cap R^*$ within the same time bound.

▶ **Corollary 15.** *$\mathcal{A}_k(B^*) \cap R^*$ has size $O(nk)$ and can be computed in $O(nk \log n)$ time.*

Observe that $\mathcal{A}_{|B|-k-1}(B^*) \cap R^*$ has size $O(nk)$ and can be computed in $O(nk \log n)$ time by Corollary 15 and symmetry. Thus, by Lemma 12 and Corollary 15, we can decide whether there exists $\Upsilon_k(x)$ for some x intersecting no red lines in $O(nk \log n)$ time.

Recall that $k_{\mathsf{opt}}/2 \leqslant k' < 2k_{\mathsf{opt}}$, where k' is the smallest value of $k = 2^0, 2^1, \ldots$ for which the exponential search succeeds. Once the search succeeds, we terminate it and set $M = k'$. The total running time is $\sum_{k=2^0,2^1,\ldots,k'} O(nk \log n) = O(nk' \log n)$, which is $O(nk_{\mathsf{opt}} \log n)$.

▶ **Lemma 16.** *We can compute M such that $k_{opt}/2 \leqslant M < 2k_{opt}$ in $O(nk_{opt} \log n)$ time.*

5.2 Finding an Optimal Strip

By Lemma 16, we can compute M such that $k_{\mathsf{opt}} \in [M/2, 2M]$ in $O(nk_{\mathsf{opt}} \log n)$ time. Observe that for an integer $i \in [M/2, 2M]$, there exists a vertical segment that intersects no red lines and misses exactly i blue lines if and only if $i \geqslant k_{\mathsf{opt}}$. Thus, we find k_{opt} using binary search within $[M/2, 2M]$ as follows. At each binary step, let $[s, e]$ be the current range and m its median. Initially, $[s, e] = [M/2, 2M]$. We test if there exists a vertical segment intersecting no red lines and missing exactly m blue lines. If so, update $[s, e]$ to $[s, m]$. Otherwise, update $[s, e]$ to $[m + 1, e]$. Continue the binary search recursively.

Preprocessing

We compute, for every $i = 0, \ldots, 2M$, the sorted list of points in $\mathcal{A}_i(B^*) \cap R^*$ in increasing order of their x-coordinates. We also do this for every $i = |B| - 2M - 1, \ldots, |B| - 1$. We show that this can be done in $O(nM \log n)$ time in total.

▶ **Lemma 17.** *For every $i = 0, \ldots, 2M$, the list of points in $\mathcal{A}_i(B^*) \cap R^*$ sorted in increasing order of their x-coordinates can be computed in $O(nM \log n)$ total time.*

Proof. We first compute the arrangement $\mathcal{A}_{\leqslant 2M}(B^*)$ in $O(n \log n + nM)$ time [15]. This allows us to store each chain $\mathcal{A}_i(B^*)$ explicitly for every $i = 0, \ldots, 2M$ by tracing the arrangement. Note that the complexity of $\mathcal{A}_{\leqslant 2M}(B^*)$ is $O(nM)$.

Then we compute all intersection points $Q = \mathcal{A}_{\leqslant 2M}(B^*) \cap R^*$ and sort them in increasing order of their x-coordinates. By Lemma 14, this can be done in $O(nM \log n)$ time.

We determine which chain $\mathcal{A}_i(B^*)$ contains each point of Q by sweeping the plane with a vertical line ℓ. For each $i = 0, \ldots, 2M$, we maintain the edge of $\mathcal{A}_i(B^*)$ intersected by ℓ, sorted by increasing i. Since the points $\mathcal{A}_i(B^*) \cap \ell$ for $i = 0, \ldots, 2M$ are sorted by their y-coordinates, we can determine which chain $\mathcal{A}_i(B^*)$ contains the point of Q hit by ℓ by binary search in $O(\log M)$ time. Since $|Q| = O(nM)$, it takes $O(nM \log M)$ time in total. ◀

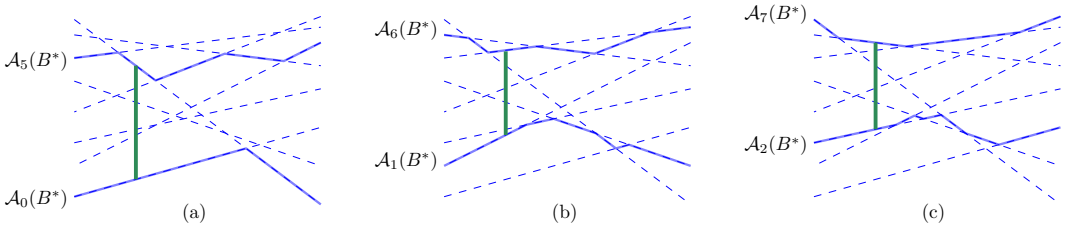


Figure 7 For 8 blue lines, a vertical segment missing two blue lines has endpoints in $\mathcal{A}_i(B^*)$ and $\mathcal{A}_j(B^*)$ for $(i,j) \in \{(0,5),(1,6),(2,7)\}$.

Binary search

At each binary step with range $[s,e]$, we test if there exists a vertical segment missing exactly $m = \lfloor (s+e)/2 \rfloor$ blue lines and intersecting no red lines. If a vertical segment with endpoints in $\mathcal{A}_i(B^*)$ and $\mathcal{A}_j(B^*)$ misses exactly m blue lines, then $0 \leqslant i \leqslant m$ and $j = |B| - m - 1 + i$. See Figure 7. Thus, for every $i = 0, \ldots, m$, we check if there exists a vertical segment with endpoints in $\mathcal{A}_i(B^*)$ and $\mathcal{A}_{|B|-m-1+i}(B^*)$ that intersects no red lines.

By preprocessing, we have the sorted list of points in $\mathcal{A}_i(B^*) \cap R^*$ and $\mathcal{A}_{|B|-m-1+i}(B^*) \cap R^*$ for every $i \in \{0,1,\ldots,m\}$. By Lemma 12, we can test in $O(|R^*| + |\mathcal{A}_i(B^*) \cap R^*| + |\mathcal{A}_{|B|-m-1+i}(B^*) \cap R^*|)$ time whether there exists a vertical segment with endpoints in $\mathcal{A}_i(B^*)$ and $\mathcal{A}_{|B|-m-1+i}(B^*)$ that intersects no red lines. Thus, the total time is

$$\sum_{i=0,\ldots,m} O(|R^*| + |\mathcal{A}_i(B^*) \cap R^*| + |\mathcal{A}_{|B|-m-1+i}(B^*) \cap R^*|),$$

which is $O(nm + |\mathcal{A}_{\leqslant m}(B^*)| + |\mathcal{A}_{\geqslant |B|-m-1}(B^*)|)$, where $\mathcal{A}_{\geqslant |B|-m-1}(B^*)$ denotes the union of i-levels of $\mathcal{A}(B^*)$ for all $i \geqslant |B| - m - 1$. By Lemma 14, $|\mathcal{A}_{\leqslant m}(B^*) \cap R^*| = O(nm)$, and $|\mathcal{A}_{\geqslant |B|-m-1}(B^*) \cap R^*| = O(nm)$ by symmetry. Thus, each binary-search step takes $O(nm) = O(nk_{\mathsf{opt}})$ time.

Since the binary search takes $O(\log M) = O(\log k_{\mathsf{opt}})$ steps, its total time is $O(nk_{\mathsf{opt}} \log k_{\mathsf{opt}})$. Thus, the overall running time is $O(nM \log n + nk_{\mathsf{opt}} \log k_{\mathsf{opt}}) = O(nk_{\mathsf{opt}} \log n)$.

▶ **Theorem 18.** *We can solve MaxBlue and MaxBlue* in $O(nk_{opt} \log n)$ time.*

Our output-sensitive algorithm is asymptotically faster than the $O(n^2)$-time algorithm for $k_{\mathsf{opt}} = o(n/\log n)$. We can guarantee the running time $O\left(\min\{n^2, nk_{\mathsf{opt}} \log n\}\right)$ by terminating the exponential search immediately when k exceeds $n/\log n$ and running the quadratic-time algorithm.

6 Algorithm for t-MaxBlue

We present an $O(n^2 \log n)$-time algorithm for t-MaxBlue. Note that t is part of the input, but it does not affect the asymptotic running time.

▶ **Theorem 19.** *We can solve t-MaxBlue and t-MaxBlue* in $O(n^2 \log n)$ time.*

Proof. In t-MaxBlue*, we are given a set $P^* = B^* \cup R^*$ of n lines in the plane and an integer t, and find t vertical segments with a common x-coordinate whose union intersects the maximum number of lines in B^* while intersecting no lines in R^*. For convenience, we will refer to the lines in B^* as blue lines and the lines in R^* as red lines.

We first compute the arrangement $\mathcal{A}(R^* \cup B^*)$ in $O(n^2)$ time. Then, we sweep the plane with a vertical line ℓ from left to right. For any fixed position, ℓ is subdivided by $\ell \cap R^*$ into $|R| + 1$ segments (possibly open or degenerate to a point). By shrinking those segments, we consider the maximal vertical segments whose endpoints lie on $\ell \cap B^*$ and that do not intersect any lines of R^*. See Figure 8. Let $s_1, s_2, \ldots, s_{|R|+1}$ denote these segments, ordered by decreasing y-coordinate, and let n_i denote the number of blue lines intersected by s_i.

During the sweep, we maintain two lists, L_1 and L_2. List L_1 contains pairs (s_i, n_i), sorted by decreasing y-coordinate of the segments. List L_2 contains the sorted numbers n_i. Any combinatorial changes to the vertical segments in ℓ occur only when ℓ hits a vertex of $\mathcal{A}(R^* \cup B^*)$. Any changes to the numbers occur only when ℓ hits an intersection of a blue line and a red line, which is a vertex of $\mathcal{A}(R^* \cup B^*)$. At such an intersection, only two consecutive numbers n_i and n_{i+1} change, each by at most one. In Figure 8, L_1 has $(s_2, 3)$ and $(s_3, 2)$ at ℓ_1, but they change to $(s_2, 2)$ and $(s_3, 3)$ at ℓ_2 due to the intersection u. After ℓ_4, s_2 intersects no blue lines, and L_1 has $(s_2, 0)$. Let *an event* refer to a moment when ℓ hits a vertex of $\mathcal{A}(R^* \cup B^*)$.

By maintaining a balanced binary search tree that stores $R^* \cup B^*$ in sorted order of their intersections with ℓ, as in a standard plane sweep algorithm, we can compute all events sorted in their x-coordinates in $O(n^2 \log n)$ total time. At an event with a blue line and a red line defining the event, we can update both L_1 and L_2 in $O(\log n)$ time. Thus, the overall running time is $O(n^2 \log n)$.

To find an optimal solution, we maintain an additional value, the sum of the largest t values in L_2. The sum can be updated in $O(\log n)$ time per event by maintaining L_2. Let $N \subseteq \{n_1, \ldots, n_{|R|+1}\}$ be the set of the t largest values in L_2, and let $S \subseteq \{s_1, \ldots, s_{|R|+1}\}$ be the corresponding segments at a position of ℓ where the sum is maximized. The vertical segments in S intersect $\sum_{m \in N} m$ blue lines but no red lines, and S is an optimal solution to t-MaxBlue*. By the dual transform, the dual of the vertical segments in S are t parallel strips that contain $\sum_{m \in N} m$ blue points but no red points. Thus, the set of t parallel strips, dual to S, is an optimal solution to t-MaxBlue. ◀

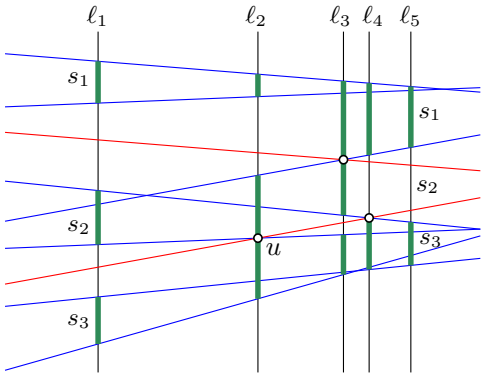


Figure 8 ℓ moves from ℓ_1 to ℓ_5. Any changes to L_2 occur only when ℓ hits an intersection of a blue line and a red line, marked by a small disk.

7 Algorithm and 3SUM-hardness for t-MinRed

We first show that t-MinRed is 3SUM-hard even when $t = 2$. Then we show that t-MinRed can be solved in $O(n^2)$ time for $t = 2$ and in $O(n^2 \log n)$ time for $t \geqslant 3$.

▶ **Lemma 20.** *2-MinRed is 3SUM-hard.*

Proof. We use a reduction from the GEOMBASE problem. Consider a set of n red points on the horizontal lines $y = 0$, $y = 1$, and $y = 2$, each with an integer x-coordinate satisfying $0 < x \leqslant M$ for some integer M. For each $i = 0, 1, 2$, and for each red point $p = (\beta, i)$, we place two blue points q, q' on $y = i$ as in Theorem 11 in total $O(n)$ time such that no line passing through any two red points (a, j), (b, k) for $\{j, k\} = \{0, 1, 2\} \setminus \{i\}$ intersects qq' unless it passes through p. We have n red points and $2n$ blue points in total. The reduction takes $O(n)$ time.

Suppose that there exists a non-horizontal line ℓ containing three red points. Then there are two strips parallel to ℓ such that their union contains all blue points while containing exactly $n - 3$ red points. If no non-horizontal line contains three red points, no two parallel strips can cover all blue points while excluding three or more red points. In particular, every pair of parallel strips whose union contains all blue points must contain at least $n - 2$ red points. ◀

In the dual plane, we are given a set $P^* = B^* \cup R^*$ of n lines in the plane and an integer t. Our goal is to find t vertical segments with a common x-coordinate whose union intersects all lines in B^* while minimizing the number of lines in R^* intersected by the union.

▶ **Theorem 21.** *We can solve 2-MinRed in $O(n^2)$ time.*

Proof. For a cell C of $\mathcal{A}(B^*)$, let $\mathsf{H_U}(C)$ and $\mathsf{H_L}(C)$ denote the upper hull and the lower hull of C, respectively. Let s_1 and s_2 be an optimal pair of vertical segments where s_1 lies above s_2. Since $s_1 \cup s_2$ intersects all lines in B^*, s_1 has its upper endpoint on $\mathcal{A}_{|B|-1}(B^*)$, s_2 has its lower endpoint on $\mathcal{A}_0(B^*)$, and both s_1 and s_2 are incident to a common cell in $\mathcal{A}(B^*)$. See Figure 9(a).

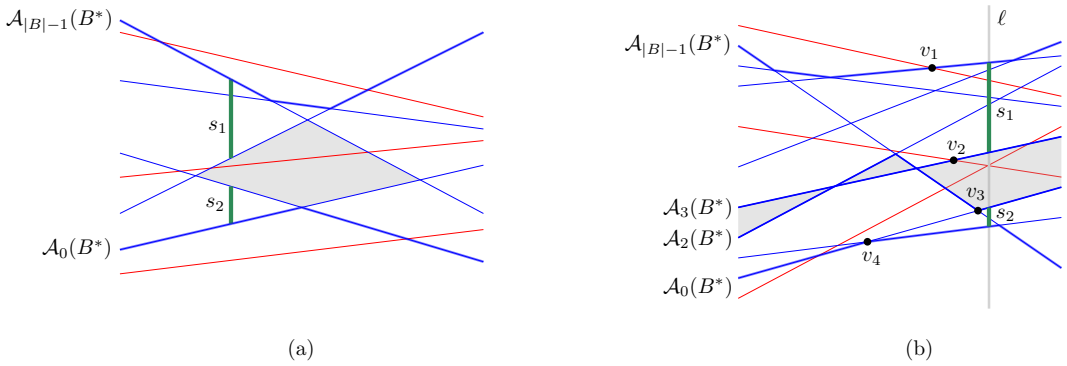


Figure 9 (a) s_1 has its upper endpoint on $\mathcal{A}_{|B|-1}(B^*)$, s_2 has its lower endpoint on $\mathcal{A}_0(B^*)$, and both s_1 and s_2 are incident to a common cell in $\mathcal{A}(B^*)$. (b) The gray region represents the cells in $\mathcal{C}_2$. The number of red lines intersected by $s_1 \cup s_2$ is $n(v_2) - n(v_1) + n(v_4) - n(v_3) = 1 - 0 + 3 - 3 = 1$.

Therefore, for each cell C in $\mathcal{A}(B^*)$, except the cell below $\mathcal{A}_0(B^*)$ and the cell above $\mathcal{A}_{|B|-1}(B^*)$, we find two vertical segments s_1 and s_2 such that their endpoints are the intersections between a vertical line and $\{\mathcal{A}_{|B|-1}(B^*), \mathsf{H_U}(C), \mathsf{H_L}(C), \mathcal{A}_0(B^*)\}$, and the number of lines in R^* intersected by $s_1 \cup s_2$ is minimized.

To do this efficiently, we first compute $\mathcal{A}(B^* \cup R^*)$ in $O(n^2)$ time, which also gives $\mathcal{A}(B^*)$. For a vertex v in $\mathcal{A}(B^* \cup R^*)$, let $n(v)$ denote the number of lines in R^* lying above v. We can compute $n(v)$ for every vertex v in $\mathcal{A}(B^* \cup R^*)$ in $O(n^2)$ time by traversing the arrangement $\mathcal{A}(B^* \cup R^*)$.

For a cell C in $\mathcal{A}(B^*)$, all points in $\mathsf{H_L}(C)$ have the same level in $\mathcal{A}(B^*)$, and all points in $\mathsf{H_U}(C)$ have the same level in $\mathcal{A}(B^*)$, if they are defined. Their levels differ by one in $\mathcal{A}(B^*)$. Let $\mathcal{C}_i$ denote the set of cells in $\mathcal{A}(B^*)$ whose lower hull lies on $\mathcal{A}_i(B^*)$ for $i = 0, \ldots, |B| - 2$. Then, $\mathcal{C}_i$ is the set of cells in $\mathcal{A}(B^*)$ whose upper hull lies on $\mathcal{A}_{i+1}(B^*)$. Note that $\bigcup_{i=0}^{|B|-2} \mathcal{C}_i$ is the set of all cells in $\mathcal{A}(B^*)$, except the cell below $\mathcal{A}_0(B^*)$ and the cell above $\mathcal{A}_{|B|-1}(B^*)$.

Consider $\mathcal{C}_i$ for a fixed $0 \leqslant i \leqslant |B| - 2$. We find two optimal vertical segments lying on a vertical line ℓ, restricted to the cells in $\mathcal{C}_i$ as follows. Imagine we sweep the plane with a vertical line ℓ from left to right. At a fixed position of ℓ, let s_1 and s_2 be the vertical segments lying on ℓ whose endpoints are the intersections of ℓ and each of the four chains in $\{\mathcal{A}_{|B|-1}(B^*), \mathcal{A}_{i+1}(B^*), \mathcal{A}_i(B^*), \mathcal{A}_0(B^*)\}$. See Figure 9(b).

Let $v_1 \in \mathcal{A}_{|B|-1}(B^*)$, $v_2 \in \mathcal{A}_{i+1}(B^*)$, $v_3 \in \mathcal{A}_i(B^*)$, and $v_4 \in \mathcal{A}_0(B^*)$ be the rightmost vertices in $\mathcal{A}(B^* \cup R^*)$ among the ones on or left to ℓ. Then, the number of lines in R^* intersected by $s_1 \cup s_2$ is $n(v_2) - n(v_1) + n(v_4) - n(v_3)$. Thus, while we slide ℓ, the number changes only when ℓ encounters a vertex of $\mathcal{A}(B^* \cup R^*)$ contained in the four chains.

Since $n(v)$ is computed for every vertex v in $\mathcal{A}(B^* \cup R^*)$ in the preprocessing and the total number of vertices in the four chains is $O(n)$, this procedure can be done in $O(n)$ time by traversing the four chains in the arrangement $\mathcal{A}(B^* \cup R^*)$. We repeat this for every $0 \leqslant i \leqslant |B| - 2$, which takes $O(n^2)$ time in total. ◀

▶ **Theorem 22.** *We can solve t-MinRed in $O(n^2 \log n)$ time for $t \geqslant 3$.*

Proof. We give a reduction from t-MinRed to $(t+1)$-MaxBlue in $O(n)$ time, which completes the proof by Theorem 19. Let $P = B \cup R$ be a set of n points in the plane. Assume that there is an optimal solution for t-MinRed whose strips have a positive slope. The other case can be handled symmetrically.

We first compute an axis-parallel square Q that contains P in its interior. Let v_a and v_b be the upper-left and lower-right vertices of Q, respectively. Consider two translates Q_a and Q_b of Q, one with its lower-right corner at v_a and one with its upper-left corner at v_b. Let $v_1, \ldots, v_4$ and $v_5, \ldots, v_8$ denote the corners of Q_a and Q_b, respectively. For each v_i, we place n points on it. Let V be the set of $8n$ points placed on $v_1, \ldots, v_8$. Observe that the convex hull of V contains the convex hull of $B \cup R$. See Figure 10.

Let S_{opt} be an optimal solution for t-MinRed whose strips have positive slope and contain no point in V. Let $1 \leqslant k_{\mathsf{opt}} < |R|$ denote the number of points in R covered by S_{opt}. Let $\overline{S_{\mathsf{opt}}}$ be a set of $t+1$ strips that are parallel and disjoint to the strips of S_{opt} and whose union covers all points in $(P \cup V) \setminus S_{\mathsf{opt}}$. Since S_{opt} covers all points in B, $\overline{S_{\mathsf{opt}}}$ covers no points in B, and it covers $8n + |R| - k_{\mathsf{opt}}$ points in total, $8n$ from V and $|R| - k_{\mathsf{opt}}$ from R.

We now construct an instance of $(t+1)$-MaxBlue with two point sets $E = V \cup R$ and $F = B$. Here, an optimal solution consists of $t+1$ parallel strips whose union contains the maximum number of points of E while containing no points of F. Let S be an optimal solution for the $(t+1)$-MaxBlue instance. Let $\overline{S}$ be a set of t strips such that each strip is the region between two consecutive strips of S.

We show that $\overline{S}$ is an optimal solution for t-MinRed for the instance. We first show three observations on S: (1) every point in V is covered by S, (2) every point in $F = B$ is covered by $\overline{S}$, and (3) every point in R is covered by either S or $\overline{S}$.

For (1), suppose there is a point of V not covered by S. Then S covers at most $7n$ points of V, and hence at most $7n + |R|$ points of $E = V \cup R$. This contradicts the optimality of S, because $\overline{S_{\mathsf{opt}}}$ covers $8n + |R| - k_{\mathsf{opt}} > 7n + |R|$ points of E while covering no point of $F = B$.

For (2), suppose there is a point $p \in F = B$ not covered by $\overline{S}$. By definition, p is not covered by S, so p lies outside $S \cup \overline{S}$. This implies that some vertex in V is not covered by S, contradicting (1). See Figure 10(b).

For (3), suppose there is a point $q \in R$ covered by neither S nor $\overline{S}$. Then we could enlarge an outermost strip of S so that it contains q without including any point of $F = B$, contradicting the optimality of S.

By (2), $\overline{S}$ covers all points of $F = B$. Let k be the number of points of R covered by $\overline{S}$. Suppose $\overline{S}$ is not optimal for t-MinRed, that is, $k > k_{\mathsf{opt}}$. By (3), S covers $|R| - k$ points of R, and by (1), it covers all $8n$ points of V. Thus S covers $8n + |R| - k$ points of $E = V \cup R$, which is less than $8n + |R| - k_{\mathsf{opt}}$, contradicting the optimality of S_{opt}. ◀

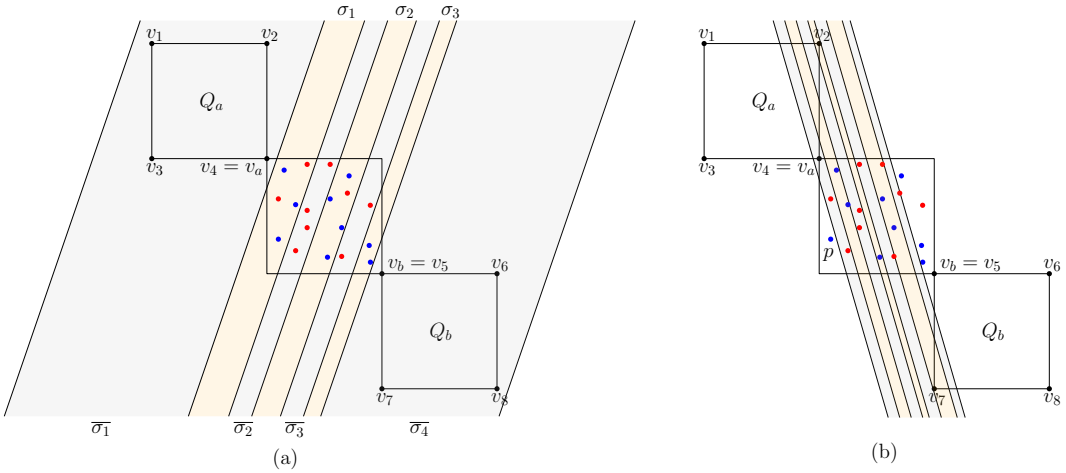


Figure 10 (a) $S_{\mathsf{opt}} = \{\sigma_1, \sigma_2, \sigma_3\}$ and $\overline{S_{\mathsf{opt}}} = \{\overline{\sigma_1}, \overline{\sigma_2}, \overline{\sigma_3}, \overline{\sigma_4}\}$ (b) S consists of the gray strips, and $\overline{S}$ consists of the yellow strips. If $S \cup \overline{S}$ does not cover p, there is a vertex in V not covered by S.

References

1. Pankaj K. Agarwal, Boris Aronov, Timothy M. Chan, and Micha Sharir. On levels in arrangements of lines, segments, planes, and triangles. *Discrete & Computational Geometry*, 19(3):315–331, 1998. doi:10.1007/PL00009348.
2. Noga Alon and E Györi. The number of small semispaces of a finite set of points in the plane. *Journal of Combinatorial Theory, Series A*, 41(1):154–157, 1986. doi:10.1016/0097-3165(86)90122-6.
3. Esther M. Arkin, Ferran Hurtado, Joseph S. B. Mitchell, Carlos Seara, and Steven S. Skiena. Some lower bounds on geometric separability problems. *International Journal of Computational Geometry & Applications*, 16(01):1–26, 2006. doi:10.1142/S0218195906001902.
4. Jonathan Backer and J. Mark Keil. The mono-and bichromatic empty rectangle and square problems in all dimensions. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics (LATIN 2010)*, pages 14–25. Springer, 2010. doi:10.1007/978-3-642-12200-2_3.
5. Mark de Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.
6. Steven Bitner, Yam Cheung, and Ovidiu Daescu. Minimum separating circle for bichromatic points in the plane. In *Proceedings of the 7th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2010)*, pages 50–55. IEEE, 2010.
7. Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS 2002)*, pages 617–626. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181985.
8. Timothy M. Chan. Remarks on k-level algorithms in the plane. Manuscript, 1999.

9 Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM Journal on Computing*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.

10 Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989. doi:10.1007/BF02187740.

11 T. K. Dey. Improved bounds for planar k -sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, March 1998. doi:10.1007/PL00009354.

12 David P. Dobkin, Dimitrios Gunopulos, and Wolfgang Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *Journal of Computer and System Sciences*, 52(3):453–470, 1996. doi:10.1006/JCSS.1996.0034.

13 Jonathan Eckstein, Peter L. Hammer, Ying Liu, Mikhail Nediak, and Bruno Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002. doi:10.1023/A:1020546910706.

14 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10. Springer Science & Business Media, 1987. doi:10.1007/978-3-642-61568-9.

15 Hazel Everett, Jean-Marc Robert, and Marc Van Kreveld. An optimal algorithm for computing ($\leqslant K$)-levels, with applications. *International Journal of Computational Geometry & Applications*, 6(03):247–261, 1996.

16 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

17 Erwin Glazenburg, Thijs van der Horst, Tom Peters, Bettina Speckmann, and Frank Staals. Robust bichromatic classification using two lines. In *Proceedings of the 35th International Symposium on Algorithms and Computation (ISAAC 2024)*, volume 322, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ISAAC.2024.33.

18 Dirk Helbing. Modeling multi-lane traffic flow with queuing effects. *Physica A: Statistical Mechanics and its Applications*, 242(1-2):175–194, 1997.

19 Michael E Houle and Godfried T Toussaint. Computing the width of a set. In *Proceedings of the first annual symposium on Computational geometry*, pages 1–7, 1985.

20 Michael F. Houle. Algorithms for weak and wide separation of sets. *Discrete Applied Mathematics*, 45(2):139–159, 1993. doi:10.1016/0166-218X(93)90057-U.

21 Ferran Hurtado, Marc Noy, Pedro A. Ramos, and Carlos Seara. Separating objects in the plane by wedges and strips. *Discrete Applied Mathematics*, 109(1-2):109–138, 2001. doi:10.1016/S0166-218X(00)00230-4.

22 Md Nazmuzzaman Khan, Adibuzzaman Rahi, Veera P Rajendran, Mohammad Al Hasan, and Sohel Anwar. Real-time crop row detection using computer vision-application in agricultural robots. *Frontiers in Artificial Intelligence*, 7:1435686, 2024. doi:10.3389/FRAI.2024.1435686.

23 Adam Kimura, Jon Scholl, James Schaffranek, Matthew Sutter, Andrew Elliott, Mike Strizich, and Glen David Via. A decomposition workflow for integrated circuit verification and validation. *Journal of Hardware and Systems Security*, 4(1):34–43, 2020. doi:10.1007/S41635-019-00086-6.

24 Sukanya Maji and Sanjib Sadhu. Geometric objects covering all red points and minimum blue points. *Turkish Journal of Engineering*, 9(1):47–55, 2025.

25 Jiri Matousek. *Lectures on Discrete Geometry*, volume 212. Springer Science & Business Media, 2013.

26 Nimrod Megiddo. On the complexity of polyhedral separability. *Discrete & Computational Geometry*, 3(4):325–337, 1988. doi:10.1007/BF02187916.

27 Joseph O'rourke, S. Rao Kosaraju, and Nimrod Megiddo. Computing circular separability. *Discrete & Computational Geometry*, 1(2):105–113, 1986.

28 Carlos Seara. *On geometric separability.* PhD thesis, Univ. Politecnica de Catalunya, 2002.

29 Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman. *Handbook of Discrete and Computational Geometry.* CRC press, 2017.

30 Géza Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001. doi:10.1007/S004540010022.

31 Godfried T Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, page A10, 1983.

32 Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

33 Chee-Keng Yap. *Fundamental problems of algorithmic algebra*, volume 49. Oxford University Press Oxford, 2000.

Parallel Algorithms for Group Isomorphism via Code Equivalence

Michael Levet
Department of Computer Science, College of Charleston, SC, USA

Abstract

In this paper, we exhibit AC^3 isomorphism tests for coprime extensions $H \ltimes N$ where H is elementary Abelian and N is Abelian; and groups where $\mathrm{Rad}(G) = Z(G)$ is elementary Abelian and $G = \mathrm{Soc}^*(G)$. The fact that isomorphism testing for these families is in P was established respectively by Qiao, Sarma, and Tang (STACS 2011), and Grochow and Qiao (CCC 2014, *SIAM J. Comput.* 2017).

The polynomial-time isomorphism tests for both of these families crucially leveraged *small* (size $O(\log |G|)$) instances of Linear Code Equivalence (Babai, SODA 2011). Here, we combine Luks' group-theoretic method for Graph Isomorphism (FOCS 1980, *J. Comput. Syst. Sci.* 1982) with the fact that G is given by its multiplication table, to implement the corresponding instances of Linear Code Equivalence in AC^3.

As a byproduct of our work, we show that isomorphism testing of arbitrary central-radical groups is decidable using AC circuits of depth $O(\log^3 n)$ and size $n^{O(\log \log n)}$. This improves upon the previous bound of $n^{O(\log \log n)}$-time due to Grochow and Qiao (*ibid.*).

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Parallel algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Group Isomorphism, Circuit Complexity, Code Equivalence

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.30

Related Version *Full Version*: `https://arxiv.org/abs/2604.13953`

Funding This work was partially supported by CRC 358 Integral Structures in Geometry and Number Theory at Bielefeld and Paderborn, Germany; the Department of Mathematics at Colorado State University; James B. Wilson's NSF grant DMS-2319370; and travel support from the Department of Computer Science at the College of Charleston.

Acknowledgements I wish to thank Peter Brooksbank, Edinah Gnang, Joshua A. Grochow, Takunari Miyazaki, and James B. Wilson for helpful discussions, as well as the anonymous referees for their helpful feedback. Parts of this work began at Tensors Algebra-Geometry-Applications (TAGA) 2024. I wish to thank Elina Robeva, Christopher Voll, and James B. Wilson for organizing this conference.

1 Introduction

The Group Isomorphism problem (GpI) takes as input two finite groups G and H, and asks if there exists an isomorphism $\varphi : G \to H$. When the groups are given by their multiplication (Cayley) tables, it is known that GpI belongs to $\mathsf{NP} \cap \mathsf{coAM}$. The generator-enumeration strategy has time complexity $n^{\log(n)+O(1)}$, where n is the order of the group [24, 44]. The parallel complexity of generator-enumeration has been gradually improved – see [20]. Algorithmically, the best known bound for GpI is $n^{(1/4)\log(n)+O(1)}$ [47, 41] (see [37, Sec. 2.2]). Even the substantial work on practical algorithms in more succinct input models (e.g., [13, 23, 12, 18]) still results in an $n^{\Theta(\log n)}$-time algorithm in the general case [55].

The Cayley model is impractical when working with computer algebra software. Instead, the groups are given by generators as permutations or matrices, or as black-boxes. For permutation groups, GpI belongs to NP [39]. For matrix groups or black-box groups, GpI

© Michael Levet;
licensed under Creative Commons License CC-BY 4.0

belongs to $\mathsf{Promise}\Sigma_2^p$ [9]; it is open whether GpI belongs to NP or coNP in such succinct models. In the past several years, there have been significant advances on algorithms with worst-case guarantees on the serial runtime for special cases of this problem – see [30, 21, 28] for a survey.

Key motivation for GpI comes from its relationship to the Graph Isomorphism problem (GI). When the groups are given verbosely by their Cayley tables, GpI $\leq_m^{\mathsf{AC}^0}$ GI [56]. There is no AC^0-Treduction from GI to GpI [19]. In light of Babai's breakthrough result that GI $\in$ QuasiP [4], GpI is a key barrier to improving the complexity of GI. There is considerable evidence suggesting that GI, and hence GpI, is not NP-complete [48, 17, 32, 4, 35, 1].

In this paper, we will investigate the parallel complexity of GpI. There are two key motivations for this. The first comes from GI, whose best known lower bound is DET [53], which is a subclass of TC^1. There is a large body of work spanning almost 40 years on NC algorithms for GI – see [38] for a survey. In contrast, the work on NC algorithms for GpI is very new (see [26] for a survey), and GpI is strictly easier than GI under AC^0-reductions [19]. So it is surprising that we know more about parallelizing subclasses of GI than subclasses of GpI. The second key motivation comes from the fact that complexity-theoretic lower bounds for GpI remain open, even against depth-2 AC circuits. In contrast, GpI is solvable using depth-4 $\mathsf{quasiAC}^0$ circuits [20]. Isomorphism testing is in P for a number of families of groups, and closing the gap between P and AC^0 for special cases is a key stepping stone in trying to characterize the complexity of the general GpI problem. We now turn to stating our main results.

Isomorphism Testing of Coprime Extensions. We will first recall the following notation [46]. Let $\mathcal{H}(\mathcal{A}, \mathcal{E})$ be the class of coprime extensions $H \ltimes N$, where N is Abelian and H is elementary Abelian.

▶ **Theorem 1** (cf. [46]). *There exists a uniform AC^3 algorithm that, given groups G_1, G_2 by their multiplication tables, decides if $G_1, G_2 \in \mathcal{H}(\mathcal{A}, \mathcal{E})$; and if so, decides if $G_1 \cong G_2$.*

There has been considerable work on polynomial-time isomorphism tests for coprime [36, 46, 8, 28, 14] and tame extensions [29]. In [46], polynomial-time isomorphism tests were given for coprime extensions $H \ltimes N$ where (i) H was $O(1)$-generated and N was Abelian, and (ii) $\mathcal{H}(\mathcal{A}, \mathcal{E})$. The parallel complexity for isomorphism testing of coprime extensions was first investigated by Grochow and Levet [28], who showed that some constant-dimensional Weisfeiler–Leman (WL) algorithm identifies, in $O(1)$-rounds, all coprime extensions $H \ltimes N$, where N is Abelian and H is $O(1)$-generated. Consequently, they obtained that the isomorphism problem for this family belongs to L. In his thesis, Brachter [14] showed that the more general class of groups with Abelian Sylow towers has WL-dimension $O(\log \log n)$. Consequently, Brachter's work yields $n^{O(\log \log n)}$-time bounds for isomorphism testing of this class. It remains open whether even the class $\mathcal{H}(\mathcal{A}, \mathcal{E})$ has bounded Weisfeiler–Leman dimension. Prior to this paper, it was open whether isomorphism testing of groups in $\mathcal{H}(\mathcal{A}, \mathcal{E})$ was in NC.

Methods. The polynomial-time isomorphism test for $\mathcal{H}(\mathcal{A}, \mathcal{E})$ [46] crucially leveraged *small* (size $O(\log |G|)$) instances of Linear Code Equivalence (CodeEq). Thus, in order to obtain our parallel bounds, we will establish the following.

▶ **Theorem 2.** *Let $m \in O(\log n)$, and let C_1, C_2 be linear codes of length m given by their generator matrices. We can decide equivalence between C_1 and C_2, as well as compute the coset of equivalences, using an AC circuit of depth $O((\log^2 n) \cdot \mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$.*

In establishing the bound of $(2+o(1))^m$-time for CodeEq, Babai reduces to isomorphism testing of graphs with $O(m)$ vertices [6]. When $m \in O(\log n)$, applying Babai's quasipolynomial GI procedure [4] would only yield bounds of $m^{O(\log^2 m)} = (\log n)^{O((\log\log n)^2)}$ runtime, which is not sufficient to obtain bounds of NC. Instead, we utilize Luks' group-theoretic method [40] to handle these small graphs. While Luks' claims of polynomial-time isomorphism testing are for graphs of bounded valence, the graphs we will consider need not satisfy this condition. However, our graphs are *small*, having only $O(\log n)$ many vertices. Thus, we bring to bear a standard suite of NC algorithms for permutation groups [3, 43]. As the group G is specified by its multiplication table and our graphs have $O(\log n) = O(\log |G|)$ many vertices, we can implement solutions for each subroutine of [3, 43] using an AC circuit of depth $\text{poly}(\log\log n)$ and size $\text{poly}(n)$. Grochow, Johnson, and Levet [26] introduced the notation $\mathsf{FOLL}^{O(1)}$ to refer to the class of languages decidable by uniform AC circuits of depth $\text{poly}(\log\log n)$ and size $\text{poly}(n)$. We combine this suite of $\mathsf{FOLL}^{O(1)}$permutation group algorithms, as well as the $\mathsf{FOLL}^{O(1)}$procedure for *small* instances of Coset Intersection (CosetInt) from [26].

Isomorphism Testing of Central-Radical Groups. We now turn to our second main result. Recall that the *solvable radical* is the unique maximal solvable normal subgroup of G. We will denote the solvable radical of G as $\text{Rad}(G)$. Here, we will consider *central-radical groups*, which are precisely those groups where the solvable radical $\text{Rad}(G) = Z(G)$.

▶ **Theorem 3** (cf. [30, Thm. C]). *Let G_1, G_2 be groups given by their multiplication tables. Suppose that G_1, G_2 have central, elementary Abelian radicals. Then isomorphism can be decided in AC^3, if either:*

(a) *$G_1/\text{Rad}(G_1)$ is a direct product of non-Abelian simple groups; or*

(b) *$G_1/\text{Rad}(G_1)$ is a direct product of perfect groups, each of order $O(1)$.*

This improves upon the previous polynomial-time bounds due to Grochow and Qiao [30].

Key motivation for isomorphism testing of central-radical groups comes from the following series of characteristic subgroups, known as the *Babai–Beals filtration* [5]: $1 \leq \text{Rad}(G) \leq \text{Soc}^*(G) \leq \text{PKer}(G) \leq G$. We will now explain the terms on this chain. Let $\pi : G \to G/\text{Rad}(G)$ be the natural projection map. $\text{Soc}^*(G)$ is the preimage of the socle[1] $\text{Soc}(G/\text{Rad}(G))$, under π. The group $\text{Soc}^*(G)/\text{Rad}(G) = \text{Soc}(G/\text{Rad}(G))$ is the direct product of non-Abelian simple groups $T_1, \ldots, T_k$. The conjugation action of G on $\text{Soc}(G/\text{Rad}(G))$ induces a permutation action φ on $T_1, \ldots, T_k$. Now $\text{PKer}(G) := \text{Ker}(\varphi)$. Note that Thm. 3(a) considers precisely those groups where $\text{Rad}(G) = Z(G)$ is elementary Abelian, and $G = \text{Soc}^*(G)$. While these assumptions might feel restrictive, Grochow and Qiao essentially leveraged the full weight of [30] in order to obtain a polynomial-time isomorphism test, combining deep mathematical insights (group cohomology) with demanding algorithmic techniques (see under Methods, below).

The class of *Fitting-free* groups consists precisely of those groups G where $\text{Rad}(G)$ is trivial. There has been considerable work on isomorphism testing of Fitting-free groups. In the Cayley table model, it took a series of two papers [6, 7] to obtain a polynomial-time isomorphism test, and this bound was only recently improved this bound from P to AC^3 [26]. Fitting-free groups have also received significant attention, from the perspective of the Weisfeiler–Leman algorithm [15, 14, 28, 27, 26].

[1] The *socle* of a group H is the direct product of the minimal normal subgroups of H.

In light of the efficient isomorphism tests for Fitting-free groups [6, 7, 26], the next step is to consider groups where $\mathrm{Rad}(G)$ is non-trivial. In [6], the authors posed the problem of handling isomorphism for central-radical groups. Grochow and Qiao [30] exhibited an $n^{O(\log\log n)}$-time algorithm to decide isomorphism of arbitrary central-radical groups, as well as groups where $\mathrm{Rad}(G)$ was elementary Abelian (not necessarily central). For additional complexity-theoretic work on isomorphism testing for groups where $\mathrm{Rad}(G)$ is non-trivial, see [29, 16, 37, 14].

Methods. We will now outline the key ideas and techniques involved in proving Thm. 3. As a conceptual starting point, let us first consider the setting of direct products. The Remak–Krull–Schmidt theorem provides that two direct products $G = G_1 \times \cdots \times G_k$ and $H = H_1 \times \cdots \times H_\ell$ are isomorpic if and only if $k = \ell$, and for some permutation $\sigma \in \mathrm{Sym}(k)$, $G_i \cong H_{\sigma(i)}$ for all $i \in [k]$. Grochow and Qiao established that for the groups in Thm. 3, an analogous result holds [30, Lem. 7.4] (recalled as Lem. 27).

Let G_1, G_2 be two groups under consideration in Thm. 3. For $i = 1, 2$, write $G_i/\mathrm{Rad}(G_i) = \prod_{j=1}^{k} T_{i,j}$. Let $\pi_i : G_i \to G_i/\mathrm{Rad}(G_i)$ be the natural projection map. Rather than considering the isomorphism type of the $T_{i,j}$, we will need to consider the isomorphism type of each $\pi_i^{-1}(T_{i,j})$. It is then necessary to determine whether there exists a permutation $\sigma \in \mathrm{Sym}(k)$ such that $\pi_1^{-1}(T_{1,j})$ and $\pi_2^{-1}(T_{2,\sigma(j)})$ are isomorphic. However, there is an additional complication, in that we require a single isomorphism $\alpha : Z(G_1) \cong Z(G_2)$, such that for all $j \in [k]$, α can be extended to an isomorphism of $\pi_1^{-1}(T_{1,j})$ and $\pi_2^{-1}(T_{2,\sigma(j)})$. Grochow and Qiao handled this step using *small* (size $O(\log |G|)$) instances of CodeEq and CosetInt. In these places, we will use our parallel implementation for CodeEq (Thm. 2), as well as the $\mathsf{FOLL}^{O(1)}$procedure for *small* instances of CosetInt from [26].

Still, we will need an additional ingredient. In order to determine the isomorphism types of the $\pi_i^{-1}(T_{i,j})$ ($i = 1, 2$; $j \in [k]$), Grochow and Qiao utilized [30, Thm. 6.1]. Note that [30, Thm. 6.1] is quite powerful. Grochow and Qiao obtained one of their main results [30, Thm. A/Cor. 6.2] as a corollary: a bound of $n^{O(\log\log n)}$-time for isomorphism testing of central-radical groups, as well as the same bound for computing the coset of isomorphisms when $\mathrm{Rad}(G_i) = Z(G_i)$ was elementary Abelian. We will parallelize [30, Thm. 6.1]; for readability, we refer to Thm. 22, rather than stating the exact theorem here. As a byproduct, we obtain the following complexity-theoretic improvement in isomorphism testing of arbitrary central-radical groups.

▶ **Theorem 4** (cf. [30, Thm. A]). *Isomorphism of central-radical groups of order n, given by their Cayley tables, can be decided by AC circuits of depth $O(\log^3 n)$ and size $n^{O(\log\log n)}$.*

2 Preliminaries

2.1 Groups and Codes

Groups. All groups will be assumed to be finite. A *Hall* subgroup of a group G is a subgroup N such that $|N|$ and $|G/N|$ are coprime. When a Hall subgroup is normal, we refer to G as a coprime extension. We will consider the following classes of finite groups, using the notation of Qiao, Sarma, and Tang [46]. Let $\mathcal{E}$ be the class of elementary Abelian groups, and $\mathcal{A}$ the class of Abelian groups. For classes of finite groups $\mathcal{X}, \mathcal{Y}$, let $\mathcal{H}(\mathcal{X}, \mathcal{Y})$ denote the class of coprime extensions $H \ltimes N$, where $N \in \mathcal{X}$ and $H \in \mathcal{Y}$.

Let G be a group. Given $g, h \in G$, the *commutator* $[g, h] := ghg^{-1}h^{-1}$. For sets $X, Y \subseteq G$, the *commutator subgroup* $[X, Y] := \langle\{[g, h] : g \in X, h \in Y\}\rangle$. We say that G is *perfect* if $G = [G, G]$, and that G is *centerless* if $Z(G) = 1$. Let $d(G)$ be the minimum size taken over all generating sets of G. A *basis* of an Abelian group A is a set of generators $\{a_1, \ldots, a_k\}$ such that $A = \langle a_1\rangle \times \cdots \times \langle a_k\rangle$.

Codes. Let $\mathbb{F}$ be a field. Let $\mathrm{Mat}_{n\times m}(\mathbb{F})$ denote the set of $n \times m$ matrices over the field $\mathbb{F}$. $\mathrm{GL}_m(\mathbb{F})$ denotes the set of $m \times m$ invertible matrices over $\mathbb{F}$. A *linear code* of length m is a subspace $U \leq \mathbb{F}^m$. A $d \times m$ matrix A over $\mathbb{F}$ *generates* the code U if the rows of A span U. Let U, W be d–dimensional codes of length m over $\mathbb{F}$, generated by $d \times m$ matrices A, B, respectively. Then U and V are *equivalent* if and only there exists a permutation matrix $P \in \mathrm{GL}_m(\mathbb{F})$ and a matrix $T \in \mathrm{GL}_d(\mathbb{F})$ such that $B = TAP$. If A_1, A_2 are generator matrices for two codes U_1, U_2 respectively, we write $\mathrm{CodeEq}(A_1, A_2)$ for the coset of code equivalences taking U_1 to U_2.

2.2 Group Extensions and Cohomology

We recall preliminaries concerning group extensions and Abelian cohomology from [30]. Given a finite group G, we will consider an Abelian normal subgroup $A \trianglelefteq G$ when considering G as an extension of A by $Q := G/A$. Here, we denote this as $A \overset{\iota}{\hookrightarrow} G \overset{\pi}{\twoheadrightarrow} Q$, where $\iota : A \to G$ is an injection and $\mathrm{Im}(\iota) = \ker(\pi)$. As A is Abelian, we write the group operation of A additively, even though we denote the group operation of G multiplicatively – this is standard in this area. Despite that $A \leq G$, we will use these notations in different contexts, and so it should not cause confusion. We refer to G as the *total group*.

Let $\pi : G \to G/A \cong Q$ be the natural projection map. Any function $s : Q \to G$ such that $\pi(s(q)) = q$ for all $q \in Q$ is called a *section* of π. Any such section gives rise to a function $f_s : Q \times Q \to A$ defined by $f_s(p, q) = s(p)s(q) \cdot s(pq)^{-1}$. We are free to choose $s(1) = \mathrm{id}_G$, and then $f(1, q) = f(q, 1) = 0$ for all $q \in Q$. Such sections are called *normalized*. We will assume that all sections are normalized, unless otherwise stated.

For $g \in G$, the conjugation action $\theta_g : G \to G$ is defined by $\theta_g(x) = gxg^{-1}$. As the operation of G is associative, we have that for all $p, q, r \in Q$: $f_s(p, q) + f_s(pq, r) = \theta_p(f_s(q, r)) + f_s(p, qr)$. This is the 2-*cocycle identity*. Any function $f : Q \times Q \to A$ is called a 2-*cochain*. If f satisfies the 2-cocycle identity with respect to θ, then f is called a 2-*cocycle* with respect to θ. Given any homomorphism $\theta : Q \to \mathrm{Aut}(A)$, every 2-cocycle with respect to θ arises as f_s for some section s of some extension $A \hookrightarrow G \twoheadrightarrow Q$ with respect to θ. Given any function $u : Q \to A$, the 2-*coboundary* associated to u is the function $b_u : Q \times Q \to A$ defined by $b_u(p, q) = u(p) + \theta_p(u(q)) - u(pq)$. Any two 2-cocycles associated to the same extension differ by a coboundary. The 2-cochains form an Abelian group $C^2(Q, A)$ defined by pointwise addition: $(f + g)(p, q) = f(p, q) + g(p, q)$. Observe that the 2-cocycle identity is $\mathbb{Z}$-linear. Thus, the 2-cocycles form a subgroup of $C^2(Q, A)$, denoted by $Z^2(Q, A, \theta)$. Similarly, the 2-coboundaries form a subgroup of $Z^2(Q, A, \theta)$, denoted by $B^2(Q, A, \theta)$. A 2-*cohomology class* is a coset of $H^2(Q, A, \theta) := Z^2(Q, A, \theta)/B^2(Q, A, \theta)$. If $f \in Z^2(Q, A, \theta)$, we denote the corresponding cohomology class by $[f]$. It follows from the above discussion that each extension $A \hookrightarrow G \twoheadrightarrow Q$ determines a single cohomology class $[f] \in H^2(Q, A, \theta)$.

▶ **Definition 5** ([30, Def. 2.1]). *Let A be an Abelian group, and Q an arbitrary group. Let $\theta : Q \to Aut(A)$ be an action, and $f : Q \times Q \to A$ a 2-cocycle ($f \in Z^2(Q, A, \theta)$). The pair (θ, f) is called* extension data. *Two extension data for the pair (Q, A) are equivalent if they have the exact same action and if the two 2-cocycles are cohomologous.*

Given an extension $A \hookrightarrow G \twoheadrightarrow Q$, the associated extension data are the action θ as defined above, and any 2-cocycle f_s for any section $s : Q \to G$. The extension data are in general not unique – we may choose any representative of the corresponding 2-cohomology class. Furthermore, if the action is trivial, then this extension is called *central*. If the 2-cohomology class is trivial, then this extension is called *split*, and $G = P \ltimes A$ for some subgroup $P \leq G$ isomorphic to Q.

We now turn to discussing when two different extension data $(\theta_1, f_1), (\theta_2, f_2)$ yield isomorphic total groups. We first recall additional notation and terminology. Recall that a subgroup $N \leq G$ is *characteristic* if for all $\varphi \in \mathrm{Aut}(G)$, $\varphi(N) = N$. The analogous notion for isomorphisms (rather than automorphisms) is a function $\mathcal{S}$ that assigns to each group G a subgroup $\mathcal{S}(G)$ such that any isomorphism $\varphi : G_1 \to G_2$ restricts to an isomorphism $\varphi|_{\mathcal{S}(G_1)} : \mathcal{S}(G_1) \to \mathcal{S}(G_2)$. We call such a function $\mathcal{S}$ a *characteristic subgroup function*. Most natural characteristic subgroups encountered are characteristic subgroup functions – for instance, the center $Z(G)$, the commutator $[G, G]$, and the solvable radical $\mathrm{Rad}(G)$.

▶ **Definition 6** ([30, Definition 2.2]). *Let A be an Abelian group, and let Q be an arbitrary group. Let $(\theta_1, f_1), (\theta_2, f_2)$ be extension data for A-by-Q. Then the extension data are* pseudo-congruent *if there exists $(\alpha, \beta) \in Aut(A) \times Aut(Q)$ such that:*

$$\theta_1(q)(a) = \alpha^{-1}\Big(\theta_2(\beta(q))(\alpha(a))\Big) =: \theta_2^{\alpha,\beta}(q)(a). \tag{1}$$

for all $q \in Q, a \in A$; and $f_1(p, q) = \alpha^{-1}(f_2(\beta(p), \beta(q))) + b_u(p, q)$, for all $p, q \in Q$ and for some coboundary b_u. In this case, we write $(\theta_1, f_1) \cong (\theta_2, f_2)$.

▶ **Lemma 7** (see e.g., [30, Lem. 2.3] and [31, Sec. 2.7.4]). *Let $\mathcal{S}$ be a characteristic subgroup function. Let G_1, G_2 be groups, such that $\mathcal{S}(G_1)$ and $\mathcal{S}(G_2)$ are both Abelian. Then $G_1 \cong G_2$ if and only if both of the following conditions hold:*
1. *$\mathcal{S}(G_1) \cong \mathcal{S}(G_2)$ and $G_1/\mathcal{S}(G_1) \cong G_2/\mathcal{S}(G_2)$.*
2. *$(\theta_1, f_1) \cong (\theta_2, f_2)$, where (θ_i, f_i) is the extension data of $A \hookrightarrow G_i \twoheadrightarrow Q$ $(i = 1, 2)$.*

Note that if the 2-cohomolohy class is trivial (in which case, the extension splits), Lem. 7 yields Taunt's Lemma as a corollary:

▶ **Lemma 8** (Taunt [52]). *Let $G = H \ltimes_\theta N$ and $\widehat{G} = \hat{H} \ltimes_{\hat{\theta}} \widehat{N}$. If $\alpha : H \to \hat{H}$ and $\beta : N \to \widehat{N}$ are isomorphisms such that for all $h \in H$ and all $n \in N$, $\widehat{\theta}_{\alpha(h)}(n) = (\beta \circ \theta_h \circ \beta^{-1})(n)$, then the map $(h, n) \mapsto (\alpha(h), \beta(n))$ is an isomorphism of $G \cong \widehat{G}$. Conversely, if G and $\widehat{G}$ are isomorphic and $|H|$ and $|N|$ are coprime, then there exists an isomorphism of this form.*

2.3 Computational Complexity

We assume that the reader is familiar with standard complexity classes such as $\mathsf{P}, \mathsf{NP}, \mathsf{L}$, and NL. Denote by FL the class of logspace computable functions. For a standard reference on circuit complexity, see [54]. We consider Boolean circuits using the gates AND, OR, NOT, and $\mathsf{Majority}$, where $\mathsf{Majority}(x_1, \ldots, x_n) = 1$ if and only if $\geq n/2$ of the inputs are 1. Otherwise, $\mathsf{Majority}(x_1, \ldots, x_n) = 0$. In this paper, we will consider $\mathsf{DTIME}(\log^c n)$-uniform circuit families $(C_n)_{n \in \mathbb{N}}$, for some fixed $c \geq 1$. For this, one encodes the gates of each circuit C_n by bit strings of length $O(\log^c n)$. Then the circuit family $(C_n)_{n \geq 0}$ is called $\mathsf{DTIME}(\log^c n)$-*uniform* if (i) there exists a deterministic Turing machine that computes for a given gate $u \in \{0, 1\}^*$ of C_n ($|u| \in O(\log^c n)$) in time $O(\log^c n)$ the type of gate u, where the types are $x_1, \ldots, x_n$, NOT, AND, OR, or $\mathsf{Majority}$ gates, and (ii) there exists a deterministic Turing

machine that decides for two given gates $u, v \in \{0,1\}^*$ of C_n ($|u|, |v| \in O(\log^c n)$) and a binary encoded integer i with $O(\log^c n)$ many bits in time $O(\log^c n)$ whether u is the i-th input gate for v. When $c = 1$, this notion of uniformity is referred to as DLOGTIME-uniformity. For circuit families of size $\text{poly}(n)$, we will use DLOGTIME-uniformity.

▶ **Definition 9.** *Fix $k \geq 0$. We say that a language L belongs to (uniform) NC^k if there exist a (uniform) family of circuits $(C_n)_{n\in\mathbb{N}}$ over the AND, OR, NOT gates such that the following hold: (i) The AND and OR gates take exactly 2 inputs. That is, they have fan-in 2; (ii) C_n has depth $O(\log^k n)$ and uses (has size) $n^{O(1)}$ gates. Here, the implicit constants in the circuit depth and size depend only on L; and (iii) $x \in L$ if and only if $C_{|x|}(x) = 1$.*

The complexity class AC^k is defined analogously as NC^k, except that the AND, OR gates are permitted to have unbounded fan-in. That is, a single AND gate can compute an arbitrary conjunction, and a single OR gate can compute an arbitrary disjunction. The complexity class TC^k is defined analogously as AC^k, except that our circuits are now permitted Majority gates of unbounded fan-in. We also allow circuits to compute functions by using multiple output gates. For every k, the following containments are well-known: $\mathsf{NC}^k \subseteq \mathsf{AC}^k \subseteq \mathsf{TC}^k \subseteq \mathsf{NC}^{k+1}$. We also have that $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1$.

The complexity class FOLL is the set of languages decidable by uniform AC circuits of depth $O(\log \log n)$ and polynomial-size [11]. We will use a slight generalization of this: we use $\mathsf{FOLL}^{O(1)}$ to denote the class of languages L decidable by uniform AC circuits of depth $O((\log \log n)^c)$ for some c that depends only on L [26]. It is known that $\mathsf{AC}^0 \subsetneq \mathsf{FOLL}^{O(1)} \subsetneq \mathsf{AC}^1$, the former by a simple diagonalization argument on top of Sipser's result [50], and the latter because the Parity function is in AC^1 but not $\mathsf{FOLL}^{O(1)}$ (nor any depth $o(\log n/\log \log n)$). $\mathsf{FOLL}^{O(1)}$ cannot contain any complexity class that can compute Parity, such as $\mathsf{TC}^0, \mathsf{NC}^1, \mathsf{L}$, or NL, and it remains open whether any of these classes contain $\mathsf{FOLL}^{O(1)}$.

We will also be interested in NC circuits of quasipolynomial size (i. e., $2^{O(\log^k n)}$ for some constant k). For a circuit class $\mathcal{C} \subseteq \mathsf{NC}$, the analogous class permitting a quasipolynomial number of gates is denoted quasi$\mathcal{C}$. Note that DLOGTIME uniformity does not make sense for quasiNC, as we cannot encode gate indices using $O(\log n)$ bits. Instead, we will use DPOLYLOGTIME-uniformity for quasiNC [10, 25].

2.4 Permutation Group Algorithms

We will consider the permutation group model, in which groups are specified succinctly by a sequence of permutations from $\text{Sym}(m)$. The computational complexity for this model will be measured in terms of m. We will first recall some key concepts concerning permutation groups – see [22] for a general reference. A permutation group $G \leq \text{Sym}(m)$ is *transitive* if the permutation domain $[m]$ is a single G-orbit. An equivalence relation $\sim$ on $[m]$ is G-invariant if $x \sim y \Leftrightarrow x^g \sim y^g$ for all $x, y \in [m], g \in G$. The discrete equivalence – in which all elements are pairwise inequivalent – and indiscrete equivalence – in which all elements are equivalent – are considered trivial. The equivalence classes of any G-invariant equivalence relation are called *blocks* of G. The singleton sets and the whole set $[m]$ are considered trivial blocks; any other blocks, if they exist, are non-trivial. G is *primitive* if it has no non-trivial blocks, or equivalently, has no non-trivial G-invariant equivalence relations.

Equivalently, a block for G is a subset $\Delta \subseteq [m]$ such that for every $g \in G$, Δ^g is either disjoint from Δ or equal to Δ. A *system of imprimitivity* for G is a collection of blocks that partition $[m]$ (equivalently, the equivalence classes of a G-invariant equivalence relation). If

G is transitive, every system of imprimitivity arises as $\{\Delta^g : g \in G\}$ for some block Δ. A block is *minimal* if it is non-trivial and contains no proper subset that is also a non-trivial block. Two distinct minimal blocks can intersect in at most one point.

We will now recall a standard suite of problems with known NC solutions in the setting of permutation groups. In our setting, m will be *small* ($m \in O(\text{polylog}(n))$) relative to the overall input size n, which will yield $\mathsf{FOLL}^{O(1)}$ bounds for these problems.

▶ **Lemma 10.** *Let $c \in \mathbb{Z}^+$, and let $m \in O(\log^c n)$. Let $G \leq Sym(m)$ be given by a sequence S of generators. The following problems are in $\mathsf{FOLL}^{O(1)}$ relative to n, that is, they have uniform AC circuits of depth $\text{poly}(\log\log n)$ and $\text{poly}(n)$ size:*

(a) *Compute the order of G.*
(b) *Decide whether a given permutation σ is in G; and if so, exhibit a word ω such that $\sigma = \omega(S)$.*
(c) *Find the kernel of any action of G.*
(d) *Find the pointwise stabilizer of $B \subseteq [m]$.*
(e) *Given a list of generators S for G, compute a minimal (non-redundant) of generators S' for G of size $\text{poly}(m)$.*
(f) *Computing a non-trivial, minimal block system for G, if one exists; otherwise, a report that G is primitive.*
(g) *Computing the orbits of G.*
(h) *CosetInt: Given $G, H \leq Sym(m)$ and $x, y \in Sym(m)$, compute $Gx \cap Hy$.*

Proof. Each of these problems in (a)–(g) is known to be solvable using a circuit of depth $O(\text{polylog}(m))$ and size $O(\text{poly}(m))$: see [3] for (a)–(e), [43] for (f)–(g). The $\mathsf{FOLL}^{O(1)}$ bound follows from the fact that $m \in O(\log^c n)$. For (h), the $\mathsf{FOLL}^{O(1)}$bound was established in [26]. ◀

Finally, we consider the Transversal problem, which takes as input $H < G \leq \text{Sym}(m)$ such that $[G : H] \in m^{O(1)}$ and asks for a left transversal for H in G. An NC algorithm was given when $|G| \in m^{O(\log m)}$ [34, Prop. 3.13]. We will instead consider Transversal when $m \in O(\log n)$ and $[G : H] \in (\log n)^{O(\log\log n)}$. A careful analysis of [34, Prop. 3.13] yields the following:

▶ **Lemma 11** (cf. [34, Prop. 3.13])**.** *Let $m \in O(\log n)$, and let $H < G \leq Sym(m)$. Suppose that $[G : H] \in (\log n)^{O(\log\log n)}$. We can solve Transversal using an AC circuit of depth $O((\log n) \cdot \text{poly}(\log\log n))$.*

2.5 Representation Theory of Finite Groups

We recall key preliminaries concerning the representation theory of finite groups. For a general reference, see [49]. Let G be a finite group, and let V be a vector space. A *representation* of G over V is a group homomorphism $\varphi : G \to \text{GL}(V)$. The *trivial representation* maps every group element to the identity matrix in $\text{GL}(V)$. If $V \cong \mathbb{F}^d$, for some field $\mathbb{F}$ and some $d \in \mathbb{N}$, a homomorphism $\varphi : G \to \text{GL}_d(\mathbb{F})$ is called a *representation* of G over $\mathbb{F}$ of dimension d. Let $\varphi : G \to \text{GL}(V)$ be a representation. A subspace $L \leq V$ is *invariant* with respect to φ if for all $g \in G$, $\varphi_g(L) = L$. Note that $0, V$ are the *trivial invariant subspaces*. A representation without any non-trivial invariant subspaces is called an *irreducible representation*.

Fix a field $\mathbb{F}$. For the rest of this section, we consider representations over $\mathbb{F}$. Let $\varphi : G \to \text{GL}(V)$ and $\rho : G \to \text{GL}(W)$ be representations. The *direct sum* $\varphi \oplus \rho$ is a representation of G over $V \oplus W$, defined as $(\varphi \oplus \rho)_g(u + v) = \varphi_g(u) + \rho_g(v)$, for all $g \in G$.

A representation is *completely reducible* if it is the direct sum of irreducible representations. Maschke's theorem states that if the characteristic of $\mathbb{F}$ is 0 or coprime to $|G|$, then any representation of G over $\mathbb{F}$ is completely reducible.

Two representations $\varphi, \psi : G \to \mathrm{GL}(V)$ are *equivalent* if there exists an invertible linear transformation $T : V \to V$ such that $\varphi(g) = T\psi(g)T^{-1}$ for all $g \in G$. Suppose that φ, ψ are completely reducible. Write $\varphi = \iota_1^{d_1} \oplus \cdots \oplus \iota_\ell^{d_\ell}$ and $\psi = \iota_1^{e_1} \oplus \cdots \oplus \iota_\ell^{e_\ell}$, where the ι_i's are irreducible and pairwise inequivalent, and each $d_i, e_i \geq 0$. We have that φ and ψ are equivalent if and only if $d_i = e_i$ for each $i \in [\ell]$.

▶ **Proposition 12.** *Let p, q be distinct primes. Let $G = \mathbb{Z}_q^\ell \ltimes_\varphi \mathbb{Z}_p^k$ be given by its multiplication table. In AC^1, we can list the irreducible components of φ and group them by equivalence type. In particular, each irreducible component is returned as a function of the form $\tau : \mathbb{Z}_q^\ell \to GL_{k_\tau}(\mathbb{Z}_p)$, where k_τ is the dimension of τ.*

3 Isomorphism Testing of Small Graphs in Parallel

In this section, we will establish the following.

▶ **Theorem 13** (cf. [40]). *Fix $n \in \mathbb{N}$. Let X, Y be graphs with $m \in O(\log n)$ vertices. We can decide whether X and Y are isomorphic, using an AC circuit of depth $O((\log n)^2 \cdot \mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$.*

In order to establish Thm. 13, we essentially parallelize the previous work of Luks for isomorphism testing for graphs of bounded valence [40]. While our graphs here need not have bounded valence, they are *small*. We will crucially take advantage of this, in tandem with a suite of efficient parallel algorithms for permutation groups (see Section 2.4).

Fix an edge e of X. We compute $\mathrm{Aut}_e(X)$, the automorphism group of X that fixes e. Following [40], define X_r to be the subgraph of X consisting of all vertices and all edges of X, which appear in paths of length $\leq r$ through e. So $X_1 = e$ and $X_{m-1} = X$. We will compute $\mathrm{Aut}_e(X_r)$ for each $r = 1, \ldots, m-1$. The groups are related via the homomorphisms: $\pi_r : \mathrm{Aut}_e(X_{r+1}) \to \mathrm{Aut}_e(X_r)$, where $\pi_r(\sigma)$ is the restriction of σ to X_r. Thus, given (generators for) $\mathrm{Aut}_e(X_r)$, determining generators for $\mathrm{Aut}_e(X_{r+1})$ reduces to two problems: find a set $\mathcal{R}$ of generators for $\mathrm{Ker}(\pi_r)$, and find a set $\mathcal{S}$ of generators for $\mathrm{Im}(\pi_r)$. A careful analysis of [40] shows that we can compute $\mathcal{R}$ in AC^0. It remains to compute $\mathcal{S}$.

▶ **Proposition 14.** *Take the same assumptions as Thm. 13 and fix $r \in [m-1]$. Suppose we are given generators for $Aut_e(X_r)$. We can compute $\mathcal{S}$ using an AC circuit of depth $O((\log n) \cdot \mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$.*

Prop. 14 immediately yields Thm. 13.

Proof of Thm. 13. We proceed for $m - 1 \in O(\log n)$ times iterations. At each iteration, we compute generators for the corresponding kernel ($\mathcal{R}$) and use Prop. 14 to compute generators for the corresponding image ($\mathcal{S}$). The result now follows. ◀

The remainder of this section will be devoted to the proof of Prop. 14. Let $\Delta(X)$ be the maximum degree of X. Let $A \subseteq 2^{V(X_r)} \setminus \{\emptyset\}$ contain those subsets of size at most Δ. Define $f : V(X_{r+1}) \setminus V(X_r) \to A$ by: $f(v) := \{w \in V(X_r) : vw \in E(X)\}$. Luks [40] previously established that $\sigma \in \mathrm{Aut}_e(X_r)$ is in $\mathrm{Im}(\pi_r)$ if and only if, for each $0 \leq s \leq m-1$, σ stabilizes the set of fathers of s-tuples: $A_s := \{a \in A : |f^{-1}(a)| = s\}$, as well as the set A' of new edges. Color A accordingly with $2m$ colors (see e.g., [40,

p. 49]). We need to now find the automorphisms in $G = \mathrm{Aut}_e(X_r)$ acting on A, that preserve the edge colors. We now recall some notions from Luks [40]. Let A be a colored set, with coloring χ. Let $B \subseteq A$ and $K \leq \mathrm{Sym}(A)$. The set of permutations in K that preserve the colors in B is: $\mathcal{C}_B(K) := \{\sigma \in K : \text{for all } b \in B, \chi(\sigma(b)) = \chi(b)\}$. Observe that: $\mathcal{C}_B(K \cup K') = \mathcal{C}_B(K) \cup \mathcal{C}_B(K')$, and $\mathcal{C}_{B \cup B'}(K) = \mathcal{C}_B(K) \cap \mathcal{C}_{B'}(K)$. Note that if $\mathcal{C}_B(\sigma B)$ is non-empty, then it is a left-coset of the subgroup $\mathcal{C}_B(G) \leq K$ [40, Lem. 2.4]. We now turn to solving the Color Automorphism problem. For $k \in \mathbb{N}$, let Γ_k be the class of groups G where all the non-Abelian composition factors of G are subgroups of $\mathrm{Sym}(k)$.

The Color Automorphism takes as input generators for a subgroup $G \leq \mathrm{Sym}(A)$ with $G \in \Gamma_k$, a G-stable subset B, and $\sigma \in \mathrm{Sym}(A)$. The solution is $\mathcal{C}_B(\sigma G)$. We will consider the Color Automorphism problem in the case when $k = m \in O(\log n)$. In order to solve the Color Automorphism problem, we proceed via a divide-and-conquer procedure. We have the following cases in the recursion:

Case 1 (Base Case). The recursion bottoms out when $|B| = 1$. In this case, we compute the Pointwise Stabilizer of $\mathcal{C}_{B^{\sigma^{-1}}}(G)$ in $\mathsf{FOLL}^{O(1)}$using Lem. 10(d). Now $\sigma\mathcal{C}_{B^{\sigma^{-1}}}(G) = \mathcal{C}_B(\sigma G)$.

Case 2 (Intransitive Case). Suppose that B is the disjoint union of G-stable subsets $Z_1, \ldots, Z_k$. We use Lem. 10(g) to, in $\mathsf{FOLL}^{O(1)}$, break B up into orbits $Z_1, \ldots, Z_k$. We now have that: $\mathcal{C}_B(\sigma G) = \bigcap_{i=1}^{k} \mathcal{C}_{Z_i}(\sigma G)$. We may thus compute each $\mathcal{C}_{Z_i}(\sigma G)$ in parallel. As G acts transitively on each Z_i, the recursive calls for $\mathcal{C}_{Z_i}(\sigma G)$ fall under Cases 1 or 3. In order to compute the intersection, we use a binary tree circuit of depth $O(\log k) \leq O(\log |X|) \leq O(\log \log n)$. At each node of the binary tree, we utilize the $\mathsf{FOLL}^{O(1)}$ algorithm for CosetInt (Lem. 10h). Thus, the total non-recursive work in this case is $\mathsf{FOLL}^{O(1)}$.

Case 3 (Transitive Case). If $|B| > 1$ and B is not the disjoint union of at least two G-stable subsets, then we find a minimal G-block system in B, which we call Φ. As $|X| \in O(\log n)$, such a block system is $\mathsf{FOLL}^{O(1)}$-computable (Lem. 10(f)). Furthermore, by Lem. 10(c), we may compute the kernel K of the G-action on Φ in $\mathsf{FOLL}^{O(1)}$. From [2], we have that $[G : K] \in (\log n)^{O(\log \log n)}$. Thus, we may write G as a union of $(\log n)^{O(\log \log n)}$ cosets: $G = \bigcup_{i=1}^{[G:K]} \tau_i K$, using an AC circuit of depth $O((\log n)\mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$ (Lem. 11). Now the problem breaks up as follows: $\mathcal{C}_B(\sigma G) = \bigcup_{i=1}^{[G:K]} \mathcal{C}_B(\sigma \tau_i K)$.

In parallel, we recursively compute $\mathcal{C}_B(\sigma \tau_i K)$ for each $1 \leq i \leq [G : K]$. As K fixes (setwise) each block of Φ, the computation for each $\mathcal{C}_B(\sigma \tau_i K)$ falls under Case 1 or Case 2. We may, in parallel, recursively compute each $\mathcal{C}_B(\sigma \tau_i K)$. Each recursive call deals with subsets of size $|B|/\ell$. When we recombine the cosets, we must take care to ensure that we only have $\mathrm{poly}(n)$ generators. We accomplish this in $\mathsf{FOLL}^{O(1)}$using Lem. 10(e). We use a binary tree circuit of depth $\log[G : K] \in O((\log \log n)^2)$ to compute $\mathcal{C}_B(\sigma G)$ from the $\mathcal{C}_B(\sigma \tau_i K)$ $(1 \leq i \leq [G : K])$. This yields a circuit of depth $\mathrm{poly}(\log \log n)$ and size $\mathrm{poly}(n)$ to compute $\mathcal{C}_B(\sigma G)$.

Complexity Analysis. A recursive call at the intransitive case (Case 2) results in either the base case (Case 1) or the transitive case (Case 3). Similarly, a recursive call at the transitive case (Case 3) results in either the base case (Case 1) or the intransitive case (Case 3). The recursive calls in the transitive case reduce the size of the problem by at least 1/2. Thus, the recursion tree thus has height $O(\log m) = O(\log \log n)$. Each level of the recursion tree is computable using an AC circuit of depth $O((\log n)\mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$. Thus, in total, we require an AC circuit of depth $O((\log n)\mathrm{poly}(\log \log n))$ and size $\mathrm{poly}(n)$, as desired. This completes the proof of Prop. 14.

4 Linear Code Equivalence for Small Codes in Parallel

In this section, we establish Thm. 2.

Proof Sketch of Thm. 2. Babai [6] exhibited an algorithm that tests the equivalence of two linear codes of length m in time $(2+o(1))^m$. He accomplished this by reducing to $\binom{m}{d}$ instances of Graph Isomorphism – precisely, isomorphism testing of $d \times (m-d)$ bipartite graphs with colored edges. A careful analysis shows that Babai's reduction is computable using an NC circuit of depth $O(\log^2 m)$ and size $\mathrm{poly}(m)$. As $m \in O(\log n)$, we obtain that this reduction is $\mathsf{FOLL}^{O(1)}$-computable. By Thm. 13, we can decide isomorphism of such graphs using an AC circuit of depth $O((\log^2 n) \cdot \mathrm{poly}(\log\log n))$ and size $\mathrm{poly}(n)$. The result now follows. ◀

5 Isomorphism Testing of Coprime Extensions in Parallel

In this section, we will establish Thm. 1. We begin by recalling some additional preliminaries concerning coprime extensions. We note that coprime extensions are determined entirely by the isomorphism types of N, H and their actions (Lem. 8). We will now show how to compute a decomposition $G = H \ltimes N$, where N is Abelian, H is elementary Abelian, and $\gcd(|H|, |N|) = 1$, if such a decomposition exists.

▶ **Lemma 15.** *Let G be a group given by its multiplication table. We can, in FL, decide if there exists an Abelian normal Hall subgroup $N \trianglelefteq G$ such that G/N is elementary Abelian and $\gcd(|N|, |G/N|) = 1$, as well as compute such an N, if one exists, and G/N in FL.*

Let p, q be distinct primes. We recall key facts regarding representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$, as well as additional background from [46, Section 5.2]. For $n \in \mathbb{N}$, the *cyclotomic polynomial* $\Phi_n(x)$ is the unique irreducible polynomial that is a divisor of $x^n - 1$, but for all $k < n$, is not a divisor of $x^k - 1$. Suppose that $\Phi_q(x)$ factors as $g_1(x) \cdots g_r(x)$, where $g_1(x), \ldots, g_r(x) \in \mathbb{Z}_p[x]$ are monic polynomials of degree $d = (q-1)/r$. Note that $d = |\mathbb{Z}_q^\times|$. Let $M \in \mathrm{GL}_d(\mathbb{Z}_p)$ be the companion matrix of $g_1(x)$. For $v \in \mathbb{Z}_q^\ell$ with $v \neq 0$, define $v^* : \mathbb{Z}_q^\ell \to \mathbb{Z}_q$ by mapping $v^*(u) = (v, u)$ (the inner product of v and u).

For $u \in \mathbb{Z}_q^\ell$, let $0 \leq h_u < q$ such that $h_u \equiv v^*(u) \pmod q$. Now define $f_v : \mathbb{Z}_q^\ell \to \mathrm{GL}_d(\mathbb{Z}_p)$ by sending $u \mapsto M^{h_u}$. We write $M^{v^*(u)}$ in place of M^{h_u}. Let $f_0 : \mathbb{Z}_q^\ell \to \mathrm{GL}_d(\mathbb{Z}_p)$ be the trivial representation. Now $\{f_v : v \in \mathbb{Z}_q^\ell\}$ forms the set of all irreducible representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$. Note that for distinct and non-zero $u, v \in \mathbb{Z}_q^\ell$, f_u and f_v might be equivalent.

▶ **Lemma 16** ([46, Claim 1])**.** *Let $u, v \in \mathbb{Z}_q^\ell$ be distinct and both non-zero. Let f_u, f_v be the corresponding irreducible representations. We have that f_u and f_v are equivalent if and only if there exists some $s \in \mathbb{Z}_q$ such that $u = sv$, and M^s and M are conjugate (by abuse of notation, we associate s with the least non-negative integer belonging to the equivalence class $s \in \mathbb{Z}_q$).*

Let $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}_k(\mathbb{Z}_p)$. As p, q are distinct primes, we have by Maschke's theorem that τ is completely reducible. Write $\tau = f_{v_1}^{k_1} \oplus \cdots \oplus f_{v_t}^{k_t}$, where each $v_i \in \mathbb{Z}_q^\ell$, and $k_1 \geq k_2 \geq \cdots \geq k_t \geq 1$. For a given multiplicity $w \in [k]$, define $L_\tau(w)$ to be the set of irreducible representations with multiplicity w appearing in τ. Now define $L_\tau := (L_\tau(w))_{w \in [k]}$. We have that L_τ determines τ up to equivalence. In order to deal with the concrete form of the representations, Qiao, Sarma, and Tang introduced the following.

▶ **Definition 17** ([46, Definition 5.4]). *Let $\tau : \mathbb{Z}_q^\ell \to GL_k(\mathbb{Z}_p)$, and let $w \in [k]$. Define $\mathcal{L}_\tau(w)$ to be a set of vectors such that for every irreducible representation $f \in L_\tau(w)$, there exists a unique $v \in \mathcal{L}_\tau(w)$ such that f and f_v are equivalent. Now define $\mathcal{L}_\tau := (\mathcal{L}_\tau(w))_{w\in[k]}$. We refer to such a tuple $\mathcal{L}_\tau$ as an* indexing tuple *of L_τ.*

A representation $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}_k(\mathbb{Z}_p)$ has at most 2^k indexing tuples [46].

▶ **Proposition 18** ([46, Claim 2]). *Let $\tau, \gamma : \mathbb{Z}_q^\ell \to GL_k(\mathbb{Z}_p)$ be two representations. We have that τ and γ are equivalent if and only if there exist indexing tuples of τ and γ, $\mathcal{L}_\tau$ and $\mathcal{L}_\gamma$, such that $\mathcal{L}_\tau = \mathcal{L}_\gamma$.*

▶ **Lemma 19.** *Let p, q be distinct primes, and let $G = \mathbb{Z}_q^\ell \ltimes_\varphi \mathbb{Z}_p^k$ be given by its multiplication table. We can list all indexing tuples of φ in NC^3.*

▶ **Theorem 20** (cf. [46, Thm. 1.3]). *Given groups G_1, G_2 by their multiplication tables, there exists a uniform AC^3 algorithm that decides if $G_1, G_2 \in \mathcal{H}(\mathcal{E}, \mathcal{E})$; and if so, decides if $G_1 \cong G_2$.*

Proof. We first use Lem. 15 to, in FL, find decompose $G_i = H_i \ltimes_{\tau_i} N_i$ $(i = 1, 2)$. We may then, in L, decide whether: (i) the N_i and H_i are elementary Abelian groups of coprime order, (ii) $N_1 \cong N_2$, and (iii) $H_1 \cong H_2$ [11]. Suppose that $N_1 \cong N_2 \cong \mathbb{Z}_p^k$ and $H_1 \cong H_2 \cong \mathbb{Z}_q^\ell$. It remains to decide whether $\tau_1, \tau_2 : \mathbb{Z}_q^\ell \to \mathrm{GL}_k(\mathbb{Z}_p)$ are equivalent. By Prop. 12, we may in AC^1, decompose τ_1, τ_2 into their irreducible components and group them by equivalence type. Write $\tau_1 = f_{v_1}^{k_1} \oplus \cdots \oplus f_{v_t}^{k_t}$ and $\tau_2 = f_{u_1}^{\ell_1} \oplus \cdots \oplus f_{u_{t'}}^{\ell_{t'}}$. Now in NC^3 (using Lem. 19), we may obtain indexing tuples $\mathcal{L}_{\tau_1}, \mathcal{L}_{\tau_2}$. We may check in L that $t = t'$; as well as that for all $w \in [k]$, whether $|\mathcal{L}_{\tau_1}(w)| = |\mathcal{L}_{\tau_2}(w)|$. If these conditions are not satisfied, then τ_1 and τ_2 are not equivalent; in which case, G_1 and G_2 are not isomorphic.

We now consider our fixed indexing tuple $\mathcal{L}_{\tau_1}$ for τ_1. By Prop. 18, it suffices to decide if there exists an indexing tuple $\mathcal{L}_{\tau_2}$ of τ_2 and $\varphi \in \mathrm{GL}_\ell(\mathbb{Z}_p)$ such that $\mathcal{L}_{\tau_1}^\varphi = \mathcal{L}_{\tau_2}$. In parallel, we will consider all indexing tuples of τ_2. Note that we can list all such indexing tuples of τ_2 in NC^3 (using Lem. 19). For clarity, fix such an indexing tuple $\mathcal{L}_{\tau_2}$. Deciding whether such a φ exists reduces to CodeEq for a code of length $m \in O(\log n)$ followed by taking an intersection of the code equivalences with $\prod_{w=1}^k \mathrm{Sym}(|\mathcal{L}_{\tau_1}(w)|)$ [46, Prop. 5]. We can write down standard 2-element generating sets for $\mathrm{Sym}(|\mathcal{L}_{\tau_1}(w)|)$ in AC^0. By Thm. 2, we can solve this instance of CodeEq using an AC circuit of depth $O((\log^2 n) \cdot \mathrm{poly}(\log\log n))$ and size $\mathrm{poly}(n)$. The requisite instance of CosetInt is $\mathsf{FOLL}^{O(1)}$-computable (Lem. 10(h)). ◀

In order to handle the case of $\mathcal{H}(\mathcal{A}, \mathcal{E})$, we use the following to reduce to the case of $\mathcal{H}(\mathcal{E}, \mathcal{E})$.

▶ **Proposition 21** (cf. [36, 8]). *Let A be an Abelian p-group given by its multiplication table. Let $S = (g_1, \ldots, g_s)$ be a basis for A. Let $\varphi_1, \varphi_2 \in Aut(A)$ be given as matrices with respect to S. Suppose that $|\varphi_1| = |\varphi_2|$. If p does not divide $|\varphi_1| = |\varphi_2|$, then there exists an FL-computable map $\Psi_p : Aut(A) \to GL_s(\mathbb{Z}_p)$ such that φ_1 and φ_2 are conjugate if and only if $\Psi_p(\varphi_1)$ and $\Psi_p(\varphi_2)$ are conjugate.*

Following the strategy of [36, 46, 8], we will later use Prop. 21 to reduce isomorphism testing of $\mathcal{H}(\mathcal{A}, \mathcal{E})$ to the case of $\mathcal{H}(\mathcal{E}, \mathcal{E})$. Le Gall [36] established Prop. 21 in the case when the complement was cyclic. He also showed that this reduction is NC-computable relative to $|A|$. Babai and Qiao [8] subsequently showed that Le Gall's reduction holds for arbitrary complements. In the full version, we will carefully analyze this reduction, to show that Ψ_p is FL-computable. This is a critical step in order to establish the AC^3 bounds in Thm. 1. We will now prove Thm. 1.

Proof of Thm. 1. Using Lem. 15, we write $G_i = H_i \ltimes_{\varphi_i} N_i$. Now in L, we test whether H_i is elementary Abelian, $N_1 \cong N_2$, and $H_1 \cong H_2$ [11, 19]. Otherwise, we reject. For $i \in [2]$, let $A_{i,p} \leq N_i$ be the Sylow p-subgroup of N_i (and hence, G_i). Let $\varphi_{i,p} : H_i \to \mathrm{Aut}(A_{i,p})$ be the projection of φ_i to $A_{i,p}$. Let $G_{i,p} = H_i \ltimes \varphi_{i,p} A_{i,p}$. By [46, Section 5.3], we have that $G_1 \cong G_2$ if and only if, for all primes p dividing $|N_1| = |N_2|$, $G_{1,p} := H_1 \ltimes_{\varphi_{1,p}} A_{1,p}$ and $G_{2,p} := H_2 \ltimes_{\varphi_{2,p}} A_{2,p}$ are isomorphic. We have shown that, in FL, we can construct each $G_{1,p}, G_{2,p}$. By [33, Prop. 4.5], we may in AC^1 compute a basis β for $A_{1,p}$, as well as $A_{2,p}$, and thus fix an isomorphism $\phi_p : A_{1,p} \cong A_{2,p}$. Let Ψ_p be as defined in Prop. 21. By Prop. 21 (applied with β), we have that $\varphi_{1,p}, \varphi_{2,p}$ are conjugate if and only if $\Psi_p(\varphi_{1,p})$ and $\Psi_p(\varphi_{2,p})$ are conjugate. Furthermore, Ψ_p is FL-computable. Thus, given $G_{1,p}$ and $G_{2,p}$, we may in AC^1 write down $G'_{i,p} := H_1 \ltimes_{\Psi_p(\varphi_{i,p})} \mathbb{Z}_p^{s_p}$ $(i = 1, 2)$. As $H_1 \cong H_2$ and $A_{1,p} \cong A_{2,p}$, we have that the following are equivalent: (i) $G_{1,p} \cong G_{2,p}$, (ii) $G'_{1,p} \cong G'_{2,p}$, and (iii) $\Psi_p(\varphi_1)$ and $\Psi_p(\varphi_2)$ are conjugate. Thus, it suffices to test for isomorphism between $G'_{1,p} \cong G'_{2,p}$, which is AC^3-computable using Thm. 20. The result now follows. ◀

6 Isomorphism Testing of Central-Radical Groups in Parallel

6.1 When Enumerating Aut(Q) is Allowed

In this section, we will establish the following.

▶ **Theorem 22** (cf. [30, Thm. 6.1]). *Let $\mathcal{S}$ be a logspace-computable characteristic subgroup function. Fix functions $d(n) \in \Omega(\log^2 n)$ and $s(n)$. Let G_1, G_2 be two groups of order n given by their multiplication tables, and suppose that (i) $\mathcal{S}(G_1) \leq Z(G_1)$ and (ii) $Aut(G_1/\mathcal{S}(G_1))$ can be listed using an AC circuit of depth $d(n)$ and size $s(n)$. Then we can decide isomorphism between G_1 and G_2 using an AC circuit of depth $\max\{d(n), \log^3(n)\}$ and size $s(n) \cdot n^{O(1)}$.*

Note that $\mathrm{Aut}(G/\mathrm{Rad}(G))$ can be listed using an an AC circuit of depth $O(\log \log n)$ and size $n^{O(\log \log n)}$ [26], which yields:

▶ **Corollary 23** (cf. [30, Thm. A]). *Isomorphism of central-radical groups, given by their multiplication tables, can be decided by AC circuits of depth $O(\log^3 n)$ and size $n^{O(\log \log n)}$.*

We recall from [30] details on how to work with 2-cohomology classes algorithmically. First, as the action is trivial in central extensions, we will drop it from $Z^2(Q, A), B^2(Q, A)$, and $H^2(Q, A)$. Write $A := \mathcal{S}(G)$, and let $A = \prod_{i=1}^{k} \mathbb{Z}/p_i^{\mu_i}\mathbb{Z}$ be the decomposition of A into cyclic subgroups. By choosing an arbitrary section s, we get a cocycle $f : Q \times Q \to A$. We may view f as a $k \times |Q|^2$-size integer matrix, which we denote M_f. The rows are indexed by $[k]$, and the columns are indexed by $Q \times Q$. For $i \in [k]$ and $(q, q') \in Q \times Q$, the entry $M_f[i, (q, q')]$ is the ith coordinate of $f(q, q')$ relative to the basis $(e_1, \ldots, e_k)$ modulo $p_i^{\mu_i}$.

Under this identification, the set $C^2(Q, A)$ is the set of all such matrices. Now $Z^2(Q, A)$ is a subgroup of $C^2(Q, A)$, under matrix addition. Similarly, $B^2(Q, A)$ is a subgroup of $Z^2(Q, A)$ under matrix addition. We will use U_i to denote the subgroup of $C^2(Q, A)$ consisting of matrices whose only non-zero entries are in the ith row (so $U_i \cong (\mathbb{Z}/p_i^{\mu_i}\mathbb{Z})^{|Q|^2}$). $\mathrm{Aut}(A)$ acts on $C^2(Q, A)$ by left-multiplication, and $\mathrm{Aut}(Q)$ acts on $C^2(Q, A)$ by permuting the columns according to the diagonal action of $\mathrm{Aut}(Q)$. Note that the actions of $\mathrm{Aut}(A)$ and $\mathrm{Aut}(Q)$ commute.

▶ **Proposition 24** (cf. [30, Prop. 6.8]). *For any $\mu \geq 1$, a $\mathbb{Z}$-basis of $B^2(Q, \mathbb{Z}/p^\mu\mathbb{Z})$ can be computed in AC^0. Furthermore, a $\mathbb{Z}_p$-basis of $B^2(Q, \mathbb{Z}_p)$ can be computed in the same bound.*

Now for a 2-cochain $f \in C^2(Q, A)$ with corresponding $k \times |Q|^2$ matrix M, let $R_i \leq (\mathbb{Z}/p_i^{\mu_i}\mathbb{Z})^{|Q|^2}$ be the subgroup generated by the ith row of M. For $\mu < \mu_i$, let $R_i^{(\mu)}$ denote the subgroup of $(\mathbb{Z}/p_i^{\mu_i}\mathbb{Z})^{|Q|^2}$ that is given by taking R_i modulo p^μ. For $\mu > \mu_i$, let $R_i^{(\mu)}$ denote the subgroup of $(\mathbb{Z}/p_i^{\mu_i}\mathbb{Z})^{|Q|^2}$ that is given by multiplying every element of R_i by $p^{\mu-\mu_i}$. For any prime $q \neq p$, let $R_i^{(q,\mu)}$ denote the trivial subgroup. If $q = p$, let $R_i^{(q,\mu)} := R_i^{(\mu)}$. Let $R^{(p,\mu)} = \langle R_1^{(p,\mu)}, \ldots, R_k^{(p,\mu)} \rangle$.

▶ **Proposition 25** ([30, Prop. 6.9]). *Let $A = \prod_{i=1}^k \mathbb{Z}/p_i^{\mu_i}\mathbb{Z}$ be an Abelian group (the p_i are primes, not necessarily distinct). Let $f_1, f_2 \in C^2(Q, A)$. With the notation as above, there exists $\alpha \in Aut(A)$ such that f_1 and f_2^α are cohomologous if and only if $\langle R_1^{(p_i,\mu_i)}, B^2(Q, \mathbb{Z}/p_i^{\mu_i}\mathbb{Z})\rangle = \langle R_2^{(p_i,\mu_i)}, B^2(Q, \mathbb{Z}/p_i^{\mu_i}\mathbb{Z})\rangle$, for each $i \in [k]$. Here, $\langle\cdot\rangle$ denotes the $\mathbb{Z}$-span (=group generated by).*

Proof of Thm. 22. We follow the strategy of [30, Thm. 6.1]. We begin by listing $\mathrm{Aut}(Q)$ using an AC circuit of size $s(n)$ and depth $d(n)$. Choose arbitrary sections of G, H, to get a 2-cocycle f_1 of G and a 2-cocycle f_2 of H. By Lem. 7, it is sufficient and necessary to test whether there exist $(\alpha, \beta) \in \mathrm{Aut}(A) \times \mathrm{Aut}(Q)$ such that f_1 and $f_2^{(\alpha,\beta)}$ are cohomologous.

For each $\beta \in \mathrm{Aut}(Q)$, we get $f_2' := f_2^{(\mathrm{id},\beta)}$. We first use Prop. 24 to, in AC^0, get a basis V for $B^2(Q, A)$. Let M_1 be the matrix representation of f_1, and M_2 be the matrix representation for f_2'. By Prop. 25, it suffices to determine whether the $\mathbb{Z}$-span of the rows in M_1 with V, is the same as the $\mathbb{Z}$-linear span of the rows of M_2 with V. Checking this condition reduces to solving a system of linear equations over the Abelian group A, which is NC^3-computable [42, 45]. The result now follows. ◀

6.2 When $\mathrm{Aut}(Q)$ is too big

We will establish Thm. 3. We first recall additional preliminaries from [30, Sec. 7]. Throughout this section, we will consider central extensions $A \hookrightarrow G_j \twoheadrightarrow Q$ $(j = 1, 2)$, with $A = Z(G_j)$, and $Q = \prod_{i=1}^\ell T_i$ with each T_i perfect, centerless, and indecomposable.

▶ **Proposition 26** (cf. [30, Prop. 6.10]). *For $A = \mathbb{Z}_p^k$ and a group Q, let $n = |A| \cdot |Q|$. In NC^2, we can compute an $Aut(A)$-invariant complement W of $B^2(Q, A)$ in $C^2(Q, A)$, as well as a $\mathbb{Z}_p$-linear projection $\pi : C^2(Q, A) \to W$ that commutes with every $\alpha \in Aut(A)$.*

The following lemma from [30] establishes conditions in which the *cohomology splits*, allowing us to restrict attention to the extensions corresponding to the individual direct factors of the quotient.

▶ **Lemma 27** ([30, Lem. 7.4]). *Given two central extensions $A \hookrightarrow G_j \twoheadrightarrow Q$ $(j = 1, 2)$, with $A = Z(G_j)$, and $Q = \prod_{i=1}^\ell T_i$ with each T_i perfect, centerless, and indecomposable. Let $U_{j,i}$ be the preimage of T_i under the natural projection map $G_j \to G_j/A$. The extensions $A \hookrightarrow G_j \twoheadrightarrow Q$ $(j = 1, 2)$ are equivalent if and only if for all $i \in [\ell]$, the extensions $A \hookrightarrow U_{j,i} \twoheadrightarrow T_i$ $(j = 1, 2)$ are equivalent.*

▶ **Lemma 28.** *Let G be a group of order n. We can decide if (i) $Rad(G) = Z(G)$ is elementary Abelian; and if so, (ii) compute the decomposition of $G/Z(G)$ into a direct product of either non-Abelian simple groups or $O(1)$-size perfect, indecomposable groups in FL. Furthermore, we can group the direct factors by isomorphism type in FL.*

Let $Q = \prod_{i=1}^\ell T_i$. Classify the T_i's together and group them by their isomorphism types, identifying $Q = \prod_{i=1}^r Q_i^{\ell_i}$. Then $\mathrm{Aut}(Q) \cong \prod_{i=1}^r \mathrm{Aut}(Q_i) \wr \mathrm{Sym}(\ell_i)$. A *diagonal* of $\mathrm{Aut}(Q)$ is an element in $\prod_{i=1}^r \mathrm{Aut}(Q_i)$. We can efficiently enumerate the diagonals:

▶ **Lemma 29.** *Given the decomposition $Q = \prod_{i=1}^{r} T_i^{\ell}$ as in the preceding paragraph, we can enumerate all diagonals of Q in* FL.

▶ **Lemma 30** ([30, Lem. 7.5])**.** *Let $A' \times A'' \overset{\iota}{\hookrightarrow} G \overset{\pi}{\twoheadrightarrow} Q$ be a central extension of $A' \times A''$ by Q. Let $p_{A'} : A' \times A'' \to A'$ be the projection onto A' along A''. If there is a 2-cocycle $f : Q \times Q \to A' \times A''$ such that $p_{A'} \circ f$ is a 2-coboundary, then G is isomorphic (even equivalent) to $A' \times (G/A')$. Furthermore, if $Z(G)$ is elementary Abelian, then A' can be computed in* NC^2 *using linear algebra over Abelian groups.*

We recall some notions from [30, Proof of Thm. 7.1]. Let G be a group under consideration in Thm. 3. We say that a 2-cocycle $f_j : Q \times Q \to A$ respects the direct factors if there exist $f_{j,i} : T_i \times T_i \to A$ $(i \in [\ell])$ such that the following condition holds: $f_j((p_1, \ldots, p_\ell), (q_1, \ldots, q_\ell)) = \sum_{i \in [\ell]} f_{j,i}(p_i, q_i)$. Let $Z^2_{\text{prod}}(Q, A)$ denote the set of 2-cocycles respecting the direct factors. Grochow and Qiao [30, Lem. 7.4] showed that $Z^2_{\text{prod}}(Q, A) \neq \emptyset$. Similarly, we say that a 2-coboundary $b_j : Q \times Q \to A$ respects the direct factors if there exist $b_{j,i} : T_i \times T_i \to A$ $(i \in [\ell])$ such that the following condition holds: $b_j((p_1, \ldots, p_\ell), (q_1, \ldots, q_\ell)) = \sum_{i \in [\ell]} b_{j,i}(p_i, q_i)$.

Let $B^2_{\text{prod}}(Q, A)$ be the set of 2-coboundaries respecting the direct factors. The difference of two cohomologous 2-cocycles in $Z^2_{\text{prod}}(Q, A)$ belongs to $B^2_{\text{prod}}(Q, A)$. Define $C^2_{\text{prod}}(Q, A)$ as the set of 2-cochains respecting the direct factors. We may view the elements of $C^2_{\text{prod}}(Q, A)$ as $k \times \left(\sum_{i \in [\ell]} |T_i|^2\right)$ matrices, whose rows are indexed by $[k]$ and whose columns are indexed by triples $(i; p, q)$ with $p, q \in T_i$. That is, $C^2_{\text{prod}}(Q, A) = \bigoplus_{i \in [k]} C^2(T_i, A)$.

Proof of Thm. 3. We follow the strategy of [30, Sec. 7.3]. We decompose G_j $(j = 1, 2)$ as an extension of $A = \text{Rad}(G_i) = \mathbb{Z}_p^k$ by $Q = \prod_{i=1}^{\ell} T_i$. Fix arbitrary sections $s_j : Q \to G_j$ $(j = 1, 2)$, and let f_j be the corresponding 2-cocycles. We then classify the T_i's according to their isomorphism types. Grouping them together by isomorphism type, we have $Q = \prod_{i=1}^{r} Q_i^{\ell_i}$. By Lem. 28, this step is FL-computable. We enumerate all diagonals of $\text{Aut}(Q)$ in FL using Lem. 29. By Lem. 7, $G_1 \cong G_2$ if and only if they are pseudo-congruent extensions of A by Q. The extensions are pseudo-congruent if and only if there exists $(\alpha, \beta) \in \text{Aut}(A) \times \text{Aut}(Q)$ such that, after twisting by (α, β), the resulting extensions are equivalent. Once we fix such an $(\alpha, \beta) \in \text{Aut}(A) \times \text{Aut}(Q)$, Lem. 27 provides that the problem is reduced to determining the equivalence of $G_1|_{T_i}$ and $G_2|_{T_i}$ $(i \in [\ell])$.

(a) Consider the case when each T_i is non-Abelian simple. In this case, each T_i is 2-generated, we can list $\text{Aut}(T_i)$ in FL [51]. By Thm. 22, we can determine the equivalence type of $G|_{T_i}$ in AC^3. Note that every $\beta \in \text{Aut}(Q)$ can be represented as a pair $(\delta, \sigma) \in (\prod_{i=1}^{r} \text{Aut}(Q_i)^{\ell_i}) \rtimes (\prod_{i=1}^{r} \text{Sym}(\ell_i))$. Thus, $Z^2_{\text{prod}}(Q, A)$ is an invariant subset of $Z^2(Q, A)$ under the actions of both $\text{Aut}(A)$ and $\text{Aut}(Q)$. Similarly, $B^2_{\text{prod}}(Q, A)$ is an invariant subset of $B^2(Q, A)$ under the actions of both $\text{Aut}(A)$ and $\text{Aut}(Q)$. By Prop. 26, we can in NC^2, compute for each $i \in [\ell]$ an $\text{Aut}(A)$-invariant W_i such that $C^2(T_i, A) = B^2(T_i, A) \oplus W_i$, along with a projection $\pi_i : C^2(T_i, A) \to W_i$ such that π_i commutes with the action of $\text{Aut}(A)$.
From [30, Sec. 7.3], we have the following properties. Each element of W_i $(i \in [\ell])$ can be written as a $k \times 17$ matrix, in a manner that is still $\text{Aut}(A)$-equivariant. For the remainder of this proof, we will denote π_i as the composition of the previous π_i, followed by the mapping onto $k \times 17$ matrices. Furthermore, for each choice of diagonal $\delta = \prod_{i=1}^{\ell} \text{Aut}(T_i)$, we may choose the complements W_i, W_h such that whenever $T_i \cong T_h$, δ identifies W_i and W_h. With this choice, we can construct $\pi_\delta = \bigoplus_{i=1}^{\ell} \pi_i : C^2(Q, A) \to W \leq \text{Mat}_{k \times 17\ell}(\mathbb{Z}_p)$, for some $\text{Aut}(A)$-invariant $W \leq C^2_{\text{prod}}(Q, A)$ such that $C^2_{\text{prod}}(Q, A) = B^2_{\text{prod}}(Q, A) \oplus W$ and π_δ commutes with the action of $\text{Aut}(A) \times \prod_{i=1}^{r} \text{Sym}(\ell_i)$. For each diagonal δ,

it remains to decide whether there exists $(\alpha, \sigma) \in \mathrm{Aut}(A) \times \prod_{i=1}^{r} \mathrm{Sym}(\ell_i)$ such that $\pi(f_1) = \pi(f_2^{(\mathrm{id},\delta,\mathrm{id})})^{(\alpha,\mathrm{id},\sigma)}$. Note that as α, δ commute and σ is already on the right, we have that $(\alpha, \delta, \sigma) = (\mathrm{id}, \delta, \mathrm{id})(\alpha, \mathrm{id}, \sigma)$.
Let $M_1 := M_{\pi(f_1)}$. Without loss of generality, we may assume that M_1 has rank k. Otherwise, we may in NC^2 (using Lem. 30), split out a direct factor of the center as $A' \times G_j/A'$ $(j = 1, 2)$. By the Remak–Krull–Schmidt theorem, we then reduce to testing isomorphism between G_1/A' and G_2/A', where the desired rank condition holds. Now in parallel, for each diagonal δ of $\prod_{i=1}^{\ell} T_i$, we compute $\pi(f_1)$ and $\pi(f_2^{(\mathrm{id},\delta,\mathrm{id})})$. Let $M_1 = M_{\pi(f_1)}$ and $M_2 = M_{\pi(f_2^{(\mathrm{id},\delta,\mathrm{id})})}$. Again note that we can enumerate all such diagonals in FL (Lem. 29). Furthermore, we have established above that π can be constructed in NC^2.
As the action of $\mathrm{Aut}(A)$ on M_1, M_2 is by left multiplication, and the action of $\prod_{i=1}^{r} \mathrm{Sym}(\ell_i)$ is on the blocks of columns, we treat M_1, M_2 as generators of two $\mathbb{Z}_p$-codes of dimension k and length 17ℓ. As $\ell \leq \log n$, we compute the coset of equivalences $\mathrm{CodeEq}(M_1, M_2) \subseteq \mathrm{Sym}(17\ell)$ using an AC circuit of depth $O((\log^2 n)(\mathrm{poly}(\log \log n)))$ and size $\mathrm{poly}(n)$ (Thm. 2). We then compute $\mathrm{CodeEq}(M_1, M_2) \cap \prod_{i=1}^{r} \mathrm{Sym}(\ell_i)$ in $\mathsf{FOLL}^{O(1)}$using Lem. 10h. If this intersection is non-empty, then we report isomorphic. Using a single OR gate, we return true (report isomorphic) if and only if at least one of these intersections is non-empty (where the OR is taken over all diagonals of Q). The total work is computable in AC^3.

(b) Consider now the case in which each T_i has size at most $D \in O(1)$. This case is handled almost identically as in (a). We sketch the differences. In this case, we can determine the isomorphism type of a given T_i in AC^0, by trying all of the $D! \in O(1)$ permutations. The corresponding linear codes have length $\ell \cdot D^2$, rather than 17ℓ [30]. The total work is computable in AC^3. ◀

References

1 V. Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. *Information and Computation*, 204(5):835–852, 2006. doi:10.1016/j.ic.2006.02.002.

2 L Babai, P.J Cameron, and P.P Pálfy. On the orders of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79(1):161–168, 1982. doi:10.1016/0021-8693(82)90323-4.

3 L. Babai, E. Luks, and A. Seress. Permutation groups in NC. In *STOC 1987*, STOC '87, pages 409–420, New York, NY, USA, 1987. Association for Computing Machinery. doi:10.1145/28395.28439.

4 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 684–697. ACM, New York, 2016. Preprint of full version at arXiv:1512.03547v2 [cs.DS]. doi:10.1145/2897518.2897542.

5 László Babai and Robert Beals. A polynomial-time theory of black box groups I. In *Groups St Andrews 1997 in Bath, I*, 1999.

6 László Babai, Paolo Codenotti, Joshua A. Grochow, and Youming Qiao. Code equivalence and group isomorphism. In *Proceedings of the Twenty-Second Annual ACM–SIAM Symposium on Discrete Algorithms (SODA11)*, pages 1395–1408, Philadelphia, PA, 2011. SIAM. doi:10.1137/1.9781611973082.107.

7 László Babai, Paolo Codenotti, and Youming Qiao. Polynomial-time isomorphism test for groups with no abelian normal subgroups - (extended abstract). In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 51–62, 2012. doi:10.1007/978-3-642-31594-7_5.

8 László Babai and Youming Qiao. Polynomial-time isomorphism test for groups with Abelian Sylow towers. In *29th STACS*, pages 453–464. Springer LNCS 6651, 2012. doi:10.4230/LIPIcs.STACS.2012.453.

9 László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 229–240. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715919.

10 D.A.M. Barrington. Quasipolynomial size circuit classes. In *[1992] Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 86–93, 1992. doi:10.1109/SCT.1992.215383.

11 David A. Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. *J. Comput. Syst. Sci.*, 63(2):186–200, 2001. doi:10.1006/jcss.2001.1764.

12 Hans Ulrich Besche and Bettina Eick. Construction of finite groups. *J. Symb. Comput.*, 27(4):387–404, 1999. doi:10.1006/jsco.1998.0258.

13 Hans Ulrich Besche, Bettina Eick, and E.A. O'Brien. A millennium project: Constructing small groups. *Intern. J. Alg. and Comput*, 12:623–644, 2002. doi:10.1142/S0218196702001115.

14 Jendrik Brachter. *Combinatorial approaches to the group isomorphism problem*. PhD thesis, TU Darmstadt, September 2023. URL: https://tuprints.ulb.tu-darmstadt.de/26387/1/thesis_brachter.pdf.

15 Jendrik Brachter and Pascal Schweitzer. A systematic study of isomorphism invariants of finite groups via the Weisfeiler–Leman dimension, 2022. doi:10.4230/LIPIcs.ESA.2022.27.

16 Peter A. Brooksbank, Joshua A. Grochow, Yinan Li, Youming Qiao, and James B. Wilson. Incorporating Weisfeiler–Leman into algorithms for group isomorphism. arXiv:1905.02518 [cs.CC], 2019.

17 Harry Buhrman and Steven Homer. Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 1992. doi:10.1007/3-540-56287-7_99.

18 John J. Cannon and Derek F. Holt. Automorphism group computation and isomorphism testing in finite groups. *J. Symb. Comput.*, 35:241–267, March 2003. doi:10.1016/S0747-7171(02)00133-5.

19 Arkadev Chattopadhyay, Jacobo Torán, and Fabian Wagner. Graph isomorphism is not AC^0-reducible to group isomorphism. *ACM Trans. Comput. Theory*, 5(4):Art. 13, 13, 2013. Preliminary version appeared in FSTTCS '10; ECCC Tech. Report TR10-117. doi:10.1145/2540088.

20 Nathaniel A. Collins, Joshua A. Grochow, Michael Levet, and Armin Weiß. On the constant-depth circuit complexity of generating quasigroups. *TheoretiCS*, Volume 4, August 2025. Preliminary version in Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation, ISSAC 2024 (DOI:10.1145/3666000.3669691). doi:10.46298/theoretics.25.19.

21 Heiko Dietrich and James B. Wilson. Group isomorphism is nearly-linear time for most orders. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–467, 2022. doi:10.1109/FOCS52979.2021.00053.

22 John D. Dixon and Brian Mortimer. *Permutation groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996. doi:10.1007/978-1-4612-0731-3.

23 Bettina Eick, C. R. Leedham-Green, and E. A. O'Brien. Constructing automorphism groups of p-groups. *Comm. Algebra*, 30(5):2271–2295, 2002. doi:10.1081/AGB-120003468.

24 V. Felsch and J. Neubüser. On a programme for the determination of the automorphism group of a finite group. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 59–60. 1970.

25 Flavio Ferrarotti, Senén González, Klaus-Dieter Schewe, and José María Turull-Torres. Proper hierarchies in polylogarithmic time and absence of complete problems. In Andreas Herzig and Juha Kontinen, editors, *Foundations of Information and Knowledge Systems*, pages 90–105, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-39951-1_6.

26 Joshua A. Grochow, Dan Johnson, and Michael Levet. Complexity of identifying Fitting-free groups. In Artur Jez and Jan Otop, editors, *Fundamentals of Computation Theory - 25th International Symposium, FCT 2025, Wrocław, Poland, September 15-17, 2025, Proceedings*, volume 16106 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2025. Preprint of full version at arXiv:2504.19777 [cs.CC]. doi:10.1007/978-3-032-04700-7_16.

27 Joshua A. Grochow and Michael Levet. On the descriptive complexity of groups without abelian normal subgroups. *Logical Methods in Computer Science*, Volume 21, Issue 4, November 2025. Preliminary version appeared in the proceedings of GandALF 2023 (DOI:10.4204/EPTCS.390.12). doi:10.46298/lmcs-21(4:20)2025.

28 Joshua A. Grochow and Michael Levet. On the parallel complexity of group isomorphism via Weisfeiler–Leman. *Journal of Computer and System Sciences*, 156:103703, 2026. Preliminary version appeared in the proceedings of FCT 2023, DOI:10.1007/978-3-031-43587-4_17. doi:10.1016/j.jcss.2025.103703.

29 Joshua A. Grochow and Youming Qiao. Polynomial-time isomorphism test of groups that are tame extensions - (extended abstract). In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 578–589, 2015. doi:10.1007/978-3-662-48971-0_49.

30 Joshua A. Grochow and Youming Qiao. Algorithms for group isomorphism via group extensions and cohomology. *SIAM J. Comput.*, 46(4):1153–1216, 2017. Preliminary version in IEEE Conference on Computational Complexity (CCC) 2014 (DOI:10.1109/CCC.2014.19). Also available as arXiv:1309.1776 [cs.DS] and ECCC Technical Report TR13-123. doi:10.1137/15M1009767.

31 D. Holt, B. Eick, and E. O'Brien. *Handbook of Computational Group Theory*. Chapman and Hall/CRC, 2005.

32 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

33 Dan Johnson, Michael Levet, Petr Vojtěchovský, and Brett Widholm. Parallel complexity of identifying groups and quasigroups via decompositions. In Uli Fahrenberg, Wesley Fussner, and Luigi Santocanale, editors, *Relational and Algebraic Methods in Computer Science*, pages 188–207, Cham, 2026. Springer Nature Switzerland. Preprint of full version at arXiv:2508.06478 [cs.DS]. doi:10.1007/978-3-032-22469-9_11.

34 William M. Kantor, Eugene M. Luks, and Peter D. Mark. Sylow subgroups in parallel. *J. Algorithms*, 31(1):132–195, 1999. doi:10.1006/JAGM.1999.0982.

35 Johannes Köbler, Uwe Schöning, and Jacobo Torán. Graph isomorphism is low for PP. *Comput. Complex.*, 2:301–330, 1992. doi:10.1007/BF01200427.

36 François Le Gall. Efficient isomorphism testing for a class of group extensions. In *Proc. 26th STACS*, pages 625–636, 2009. doi:10.4230/LIPIcs.STACS.2009.1830.

37 François Le Gall and David J. Rosenbaum. On the group and color isomorphism problems. arXiv:1609.08253 [cs.CC], 2016.

38 Michael Levet, Puck Rombach, and Nicholas Sieger. Canonizing Graphs of Bounded Rank-Width in Parallel via Weisfeiler-Leman. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2024.32.

39 Eugene Luks. Permutation groups and polynomial-time computation. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 11, September 1993. doi:10.1090/dimacs/011/11.

40 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

41 Eugene M. Luks. Group isomorphism with fixed subnormal chains, 2015. arXiv:1511.00151.

42 Pierre McKenzie and Stephen A. Cook. The parallel complexity of the abelian permutation group membership problem. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 154–161, 1983. doi:10.1109/SFCS.1983.74.

43 Pierre Murdock McKenzie. *Parallel complexity and permutation groups*. PhD thesis, University of Toronto, 1984. AAI0556312.

44 Gary L. Miller. On the $n^{\log n}$ isomorphism technique (a preliminary report). In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 51–58, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804331.

45 Ketan Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Comb.*, 7(1):101–104, 1987. doi:10.1007/BF02579205.

46 Youming Qiao, Jayalal M. N. Sarma, and Bangsheng Tang. On isomorphism testing of groups with normal Hall subgroups. In *Proc. 28th STACS*, pages 567–578, 2011. doi:10.4230/LIPIcs.STACS.2011.567.

47 David J. Rosenbaum. Bidirectional collision detection and faster deterministic isomorphism testing. arXiv:1304.3935 [cs.DS], 2013.

48 Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988. doi:10.1016/0022-0000(88)90010-4.

49 Jean-Pierre Serre. *Linear representations of finite groups*, volume Vol. 42 of *Graduate Texts in Mathematics*. Springer-Verlag, New York-Heidelberg, french edition, 1977. doi:10.1007/978-1-4684-9458-7.

50 Michael Sipser. Borel sets and circuit complexity. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 61–69. ACM, 1983. doi:10.1145/800061.808733.

51 Bangsheng Tang. *Towards Understanding Satisfiability, Group Isomorphism and Their Connections*. PhD thesis, Tsinghua University, 2013.

52 D. R. Taunt. Remarks on the isomorphism problem in theories of construction of finite groups. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(1):16–24, 1955. doi:10.1017/S030500410002987X.

53 Jacobo Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.

54 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

55 James B. Wilson. The threshold for subgroup profiles to agree is logarithmic. *Theory of Computing*, 15(19):1–25, 2019. doi:10.4086/toc.2019.v015a019.

56 V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *J. Soviet Math.*, 29(4):1426–1481, May 1985. doi:10.1007/BF02104746.

Improved and Parameterized Algorithms for Online Multi-Level Aggregation

Young-San Lin
Melbourne Business School, University of Melbourne, Australia

Alex Turoczy
Department of Computer Science, University of Oxford, UK

Abstract

We study the online multi-level aggregation problem with deadlines (MLAP-D) introduced by Bienkowski, Böhm, Byrka, Chrobak, Dürr, Folwarczný, Jeż, Sgall, Thang, and Veselý (ESA 2016, OR 2020). In this problem, requests arrive over time at the vertices of a given vertex-weighted tree, and each request has a deadline that it must be served by. The cost of serving a request equals the cost of a path from the root to the vertex where the request resides. Instead of serving each request individually, requests can be *aggregated* and served by transmitting a subtree from the root that spans the vertices on which the requests reside, to potentially be more cost-effective. The aggregated cost is the weight of the transmission subtree. The goal of MLAP-D is to find an aggregation solution that minimizes the total cost while serving all requests. MLAP-D generalizes some well-studied problems including the TCP acknowledgment problem and the joint replenishment problem, and arises in natural scenarios such as multi-casting, sensor networks, and supply chain management.

We present improved and parameterized algorithms for MLAP-D. Our result is twofold. First, we present an $e(D+1)$-competitive algorithm where D is the depth of the tree. Second, we present an $e(4H+2)$-competitive algorithm where H is the *caterpillar dimension* of the tree. Here, $H \leq D$ and $H \leq \log_2 |V|$ where $|V|$ is the number of vertices in the given tree. The caterpillar dimension remains constant for rich but simple classes of trees, such as line graphs ($H = 1$), caterpillar graphs ($H = 2$), and lobster graphs ($H = 3$). To the best of our knowledge, this is the first online algorithm parameterized on a measure better than depth. The state-of-the-art online algorithms are $6(D+1)$-competitive by Buchbinder, Feldman, Naor, and Talmon (SODA 2017) and $O(\log |V|)$-competitive by Azar and Touitou (FOCS 2020). Our framework outperforms the state-of-the-art ratios when $H = o(\min\{D, \log_2 |V|\})$. Our *memory-based* algorithms extend transmission subtrees with a cost comparable to transmission subtrees used to serve previous requests. Our simple framework directly applies to trees with *any* structure and differs from the previous frameworks that reduce the problem to trees with specific structures.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online Algorithms, Approximation Algorithms, Graph Problems

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.31

Related Version *Full Version*: `https://arxiv.org/abs/2511.23211`

1 Introduction

Aggregation optimization problems have a wide range of applications including supply chain management [2, 18, 22, 31], communication networks [19, 20], and multicasting [5, 13]. The common scenario of these applications is that individually resolving each request can be costly over time. Requests can be aggregated and served as batches under some aggregation constraints to lower service costs.

© Young-San Lin and Alex Turoczy;
licensed under Creative Commons License CC-BY 4.0

This work investigates the *online multi-level aggregation problem with deadlines* (MLAP-D) introduced in [7],[1] which captures two characteristics of aggregation optimization problems. First, the service network is hierarchical and has a rooted tree structure. Second, the requests arrive in an online and dynamic fashion, and the decision-making is irrevocable in serving the requests.

1.1 Online Multi-level Aggregation with Deadlines

In MLAP-D, we are given an undirected tree $\mathcal{T} = (V, E)$ rooted at $r \in V$. Each vertex $v \in V$ is associated with a cost defined by $c : V \to \mathbb{R}_{\geq 0}$. The tree $\mathcal{T}$ and the vertex costs are given offline, in advance. A *request* ρ is associated with a vertex $v_\rho \in V(\mathcal{T})$ and a time interval $[a_\rho, d_\rho]$. Here, v_ρ is the location of the request, and $a_\rho, d_\rho \in \mathbb{R}_{\geq 0}$ are the arrival time and the deadline of the request ρ, respectively. The time starts at zero, and the continuous time model is used. At time a_ρ, request ρ arrives and the information of ρ (v_ρ and d_ρ) is revealed. The requests appear in an online fashion, so at time t, any information about requests ρ' with $a_{\rho'} > t$ is not revealed. The output of an algorithm for MLAP-D is a collection $T_1, ..., T_\ell$ of subtrees of $\mathcal{T}$ rooted at r at different times $t_1, t_2, ...t_\ell$ where $t_1 < t_2 < ... < t_\ell$. A request ρ is served if during the time interval $[a_\rho, d_\rho]$, a rooted subtree T_i contains v_ρ. It is possible that more than one request is served at the same vertex and the same time. Every request ρ must be served before the deadline d_ρ. It is noteworthy that we need not serve requests one at a time, as requests with overlapping time intervals can be served simultaneously as a batch, which can be more cost-effective. Let $\mathcal{S} = \{(T_i, t_i)\}_{i=1}^{\ell}$ denote a solution for MLAP-D. The cost of $\mathcal{S}$ is the total cost of the subtrees at different times. More specifically, let

$$c(\mathcal{S}) = \sum_{i=1}^{\ell} c(T_i) \text{ be the cost of } \mathcal{S} \text{ where } c(T_i) = \sum_{v \in V(T_i)} c(v) \text{ is the cost of } T_i.$$

Without loss of generality we assume that the arrival times and deadlines of all requests are distinct, as any instance can be converted to an instance of this form via an infinitesimal perturbation.

We note following [15] that the vertex-weighted formulation captures the edge-weighted problem as follows. First, we can assume that in the edge-weighted instance, r only has one adjacent vertex. If r has multiple adjacent vertices, we can separate the problem into multiple instances in which r has one adjacent vertex. Combining the solution of these separated instances gives a solution of the original instance without any extra cost. Next, we can move the cost of any edge into the endpoint vertex further away from r and finally remove r and set the vertex adjacent to r as the new root.

Given a problem instance $\mathcal{I}$, and a deterministic online algorithm ALG, we use $\text{ALG}(\mathcal{I})$ to denote the cost of the solution to $\mathcal{I}$ outputted by ALG. Using the classic online algorithm framework, we compare this cost to that of the offline optimal solution OPT on this instance, which we denote $\text{OPT}(\mathcal{I})$. An algorithm is k-competitive if $\text{ALG}(\mathcal{I}) \leq k \cdot \text{OPT}(\mathcal{I})$ for all problem instances $\mathcal{I}$. For a fixed problem instance, we will simply use ALG and OPT to denote the costs.

MLAP-D was presented as a unification of several aggregation problems on trees [7]. We defer the discussions about these special case problems in Section 1.4. The first non-trivial online algorithm for MLAP-D was $D^2 2^D$-competitive [7], which was later improved to the

[1] See also [8] for the journal version.

state-of-the-art $6(D+1)$-competitive algorithm [15]. Here, D is the depth of the tree. There is a paper claiming a D-competitive algorithm [26] which we cannot confirm the correctness of. When $D = 2$, the competitive ratio is 2 [10, 11], which remains the highest known lower bound on the competitive ratio of MLAP-D for trees of fixed depth. The problem is NP-hard [2] and APX-hard [10, 27]. Closing the competitive ratio gap between 2 and $O(D)$ has been an intriguing open problem.

1.2 Our Contributions

In this work, we present improved competitiveness for MLAP-D. Our result is twofold. First, we present an improved $O(D)$-competitive algorithm. Second, we present an $O(H)$-competitive algorithm where H is the *caterpillar dimension* of the tree. Our competitive algorithms are based on a generic *memory-based* framework which judiciously constructs an online solution according to historical information.

1.2.1 Our $O(D)$-competitive algorithm

In Section 2, we show the following theorem by our memory-based framework.

▶ **Theorem 1.** *There exists a* $(1+\frac{1}{D})^D(D+1) \leq e(D+1)$*-competitive algorithm for MLAP-D.*

We emphasize that our algorithm can be directly applied to *any* tree structure. This makes the analysis direct, and differs from some previous work that relies on reductions to trees with specific structures, such as L-decreasing trees [9, 15].

1.2.2 $O(H)$-competitiveness

To potentially improve the competitive ratio for MLAP-D, we consider a preferable measure over depth, namely, the *caterpillar dimension* introduced in [25]. The notion of *heavy path decomposition* was introduced in [30]. A heavy path decomposition of $\mathcal{T}$ is a vertex-disjoint partition $\mathcal{P}$ such that (1) each element in $\mathcal{P}$ is a set of vertices that contains exactly one leaf vertex and forms a path in $\mathcal{T}$ and (2) any two vertices in an element in $\mathcal{P}$ must have an ancestor-descendant relationship. Recall that r is the root vertex of $\mathcal{T}$. Consider a leaf vertex u and the r-u path. Let $d(\mathcal{P}, u)$ denote the number of elements in $\mathcal{P}$ intersecting the r-u path. The *dimension* of the decomposition $\mathcal{P}$ is $dim(\mathcal{P}) := \max_{u \in L(\mathcal{T})}\{d(\mathcal{P}, u)\}$ where $L(\mathcal{T})$ denotes the set of leaf vertices in $\mathcal{T}$. The caterpillar dimension H of $\mathcal{T}$ is the minimum dimension among all heavy path decompositions. See Figure 1 for an example.

The caterpillar dimension remains constant for certain classes of trees, such as line graphs ($H = 1$), caterpillar graphs ($H = 2$), and lobster graphs ($H = 3$). The caterpillar dimension is at most D and $\log_2 |V|$, and can be computed in polynomial time, as well as the corresponding heavy path decomposition [25].

In Section 3, we show the following theorem using our memory-based framework.

▶ **Theorem 2.** *There exists a* $(4H+2)(1+\frac{1}{2H+1})^{H+1}(1+\frac{1}{2H})^H \leq e(4H+2)$*-competitive algorithm for MLAP-D.*

To the best of our knowledge, this is the first online algorithm parameterized on a measure better than depth. Interestingly, our framework outperforms the state-of-the-art ratios of $O(D)$ [16] and $O(\log |V|)$ [4] if $H = o(\min\{D, \log_2 |V|\})$. For infinite line graphs ($H = 1$), the best competitive ratio is known to be exactly 4 [7].

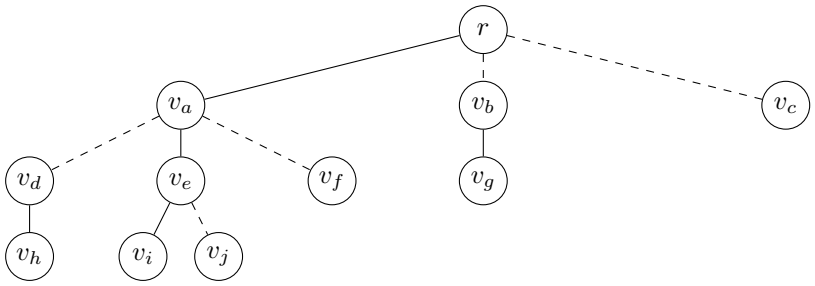


Figure 1 $\mathcal{T}$ consists of the solid and dashed edges. Components connected by the solid edges define the elements of $\mathcal{P}$. The heavy path decomposition $\mathcal{P}$ of $\mathcal{T}$ has six elements: $\{v_d, v_h\}$, $\{r, v_a, v_e, v_i\}$, $\{v_j\}$, $\{v_f\}$, $\{v_b, v_g\}$, and $\{v_c\}$. When the leaf vertex $u = v_h, v_j, v_f, v_g$, or v_c, the r-u path intersects two elements in $\mathcal{P}$ which maximizes $d(\mathcal{P}, u)$, so the dimension of $\mathcal{P}$ is two. A decomposition with a lower dimension does not exist, so the caterpillar dimension of $\mathcal{T}$ is two.

1.3 High-level Technical Overview

1.3.1 The $e(D+1)$-competitive algorithm

In both the offline and online settings, the fundamental question when designing an algorithm for MLAP-D is to decide how many extra requests to aggregate into the service subtree, beyond the bare minimum required for a feasible solution. The biggest challenge one faces when developing a competitive algorithm is finding a way to lower-bound the performance of the optimum solution for a given instance. Most online algorithms for MLAP-D [9, 15] have been *memoryless*, in the sense that the algorithm's decision making is only a function of the current unserved requests and locations, and the relative order of their deadlines. This discards information which can potentially be used to justify further aggregation of the service subtree at a given time t_i. Our main algorithmic contribution is to incorporate memory naturally to inform the algorithm's decision-making process. From a mathematical standpoint, memory is used in the amortized analysis to ensure that we can attribute the cost of our algorithm ALG to the cost of the optimal solution OPT without double-counting, which is crucial for establishing a competitive ratio.

Leaving out some technical details, we describe the motivations behind our algorithms as follows. When an active request ρ reaches its deadline at time t_i, we are forced to transmit a service subtree which contains v_ρ, and so our algorithm begins by initializing the service subtree as the path r-v_ρ. Each v in the r-v_ρ path adds a cost of $c(v)$ to the algorithm's solution, and therefore we want to delay making another service which includes v until we are sure that OPT itself must have outputted services with costs comparable to $c(v)$. Therefore, it is natural that each node receives a budget proportional to $c(v)$ which is allocated towards adding extra services to the service subtree, and we call the nodes that the budget is spent on *investments*. The investment procedure is handled by a recursive function, where deeper nodes are the first to make their investments. The difficulty with this approach is that at different times, several different vertices may invest in the same node, which could lead to over-counting the costs associated with OPT.

We use memory to tackle this issue. Before vertices on the path from r-v_ρ start spending their investment budget, they may also add their *previous* investments to the service subtree and recursively expand the transmission subtree, in a stage of the algorithm we call the *expansion stage*. This prevents nodes from reinvesting onto previous investments. The expansion stage is handled by a separate recursive process which occurs before the investment stage. It is then for these *expansion* vertices that we make investments. We decide whether

to add those previous investments as follows. After a vertex has made its investment, we set a timer in memory to record when all of the investments we have made are anticipated to reach their deadline. If a vertex v at time t_i is an expansion vertex again after the timer has elapsed, then we can make inferences about comparable costs OPT must have experienced based on our investments. In this case, we call v an *anticipated* vertex at time t_i. Otherwise, we can infer that there must have been a request that had appeared deeper than v and expired before our timer elapsed, which we can also associate with a cost by OPT. We call such vertices *unanticipated* at time t_i.

We briefly provide a broad overview of the analysis. In Section 2.1, we express ALG in terms of only the unanticipated expansion nodes. We use the standard amortized analysis techniques, in particular, an accounting argument to redistribute the costs in a top-down fashion based on proof by induction. Then in Section 2.2, we show that the unanticipated expansion nodes serve as a way to lower bound OPT. This requires using a charging scheme that defines a one-to-one mapping between the cost accounted for the unanticipated expansion vertices and OPT. As a consequence of these two results, we can bound the competitive ratio of the algorithm. When we first define the algorithm, we leave a parameter which controls the extent of aggregation unspecified. Selecting the optimal parameter results in an $e(D+1)$-competitive algorithm for MLAP-D.

1.3.2 The $e(4H+2)$-competitive algorithm

Our $e(4H+2)$-competitive algorithm utilizes the framework developed by the $e(D+1)$-competitive algorithm. The main difference is the use of the heavy path decomposition structure while assigning memory-based investment budgets to vertices. Instead of assigning the budget for every vertex v to be proportional to its cost $c(v)$, we allow one vertex per path in $\mathcal{P}$ to have a potentially much larger budget.

Following the same strategy as before, to prove the competitiveness of our algorithm, we show that (1) the cost ALG is comparable with the cost of the unanticipated expansion vertices and (2) we can relate the costs of unanticipated expansion vertices to OPT directly via a charging scheme. The charging scheme is identical to our $O(D)$-competitive algorithm. The main change to the analysis is to judiciously modify the redistribution accounting argument for the cost based on a proof by induction to show (1). Ultimately, by selecting the optimal parameters for the two different investment budgets, we obtain an $e(4H+2)$-competitive algorithm for MLAP-D.

1.4 Related Work

1.4.1 Online multi-level aggregation with waiting cost

The most related variant of MLAP-D is the setting that each request has a *waiting cost* instead of a strict deadline. The objective is to minimize the sum of the total service and waiting costs. This problem is termed the online multi-level aggregation problem (MLAP). For general MLAP, an algorithm with competitive ratio $D^4 2^D$ was presented in [7] and later improved to $O(D^2)$ [3]. When $D=1$, MLAP is also called the *TCP acknowledgment problem* (TCP-AP). The optimal competitive ratio for TCP-AP is 2 for deterministic algorithms [19] and $e/(e-1) \approx 1.582$ for randomized algorithms [16,20]. Interestingly, TCP-AP is equivalent to the *lot sizing problem* [31], which has been extensively studied in the operations research community. When $D=2$, MLAP is also called the *joint replenishment problem* (JRP). The state-of-the-art competitive ratio for JRP is 3 [17] and the current best lower bound is 2.754 [11] which improves an earlier bound of 2.64 [17]. When $H=1$, i.e., the input

tree is a line, the upper bound for the competitive ratio is 5 [12] which improves an earlier bound of 8 [14], and the lower bound is 4 [7] which improves an earlier bound of $(5+\sqrt{5})/2 \approx 3.618$ [12]. [4] introduced a unifying framework for problems with delay and deadlines admitting a competitive ratio of $O(\log |V|)$ for MLAP. A slight variant of MLAP was presented in [21] where the competitive ratio is logarithmic in the cost of the aggregation tree. We refer the reader to [9] for a concise summary for MLAP and MLAP-D.

1.4.2 Offline multi-level aggregation

In offline MLAP-D, the information of all requests, including the location, the arrival time, and the deadline (also the waiting cost for offline MLAP), is given in advance. The offline problem currently has significantly better approximation results than the online problem. For offline MLAP-D, the current best approximation ratio is 2 [6, 7]. For offline MLAP, the current best approximation ratio is $2+\varepsilon$ [28] by adapting an algorithm in [24] for the multi-stage assembly problem. For offline TCP-AP, there is an optimal dynamic programming algorithm [1]. Offline JRP is NP-hard [2] and APX-hard [10, 27] and the current best approximation ratio is 1.791 [11] improving the previous ratios of 1.8 [23] and 2 [24].

1.4.3 Approximation algorithms parameterized on caterpillar dimension

Quite surprisingly, the only algorithm parameterized on the caterpillar dimension that we are aware of is the advice complexity of the online k-server problem on trees [29]. More specifically, to obtain optimal solutions for online k-server on trees, it is sufficient to use $2+2\lceil \log_2(H+1) \rceil$ bits of advice per request. We believe that the caterpillar dimension is a natural measure to evaluate the performance of approximation and online algorithms for challenging problems on trees, especially for problems like MLAP-D where the state-of-the-art competitive ratio is linear in the tree depth.

2 The $e(D+1)$-competitive Algorithm

In this section, we prove Theorem 1.

We describe the components of Algorithm 1. Once a request ρ reaches its deadline, the procedure OnDeadline is called in order to construct a service subtree to serve ρ. During this procedure, vertices are added to a service subtree T in two separate stages, which we call the *expansion stage* and the *investment stage*. Vertices added during the procedure ExpansionStage are added to the set V^E and those added during InvestmentStage are added to V^I, and the final transmission tree is taken as the subtree induced on the disjoint union $V^E \cup V^I$. For each node v, the algorithm maintains three pieces of information which are used in both stages: $\ell_v \geq 0$, $next_v \geq 0$, and $I_v \subseteq V(\mathcal{T}_v) \setminus \{v\}$ where $\mathcal{T}_v$ is the subtree of $\mathcal{T}$ rooted at v which includes all the vertices and edges under v. These variables are initialized in the Initialization procedure at the beginning of the problem instance as $\ell_v = c(v)$, $next_v = \infty$ and $I_v = \varnothing$. We later describe how this stored information is used.

We first describe the expansion stage. When an unserved request ρ on v_ρ reaches its deadline at time t, we are forced to output a service subtree containing v_ρ. In this case, we say that the request ρ is *critical*. We start by adding the vertices on the path from r to v_ρ to V^E. From here, the algorithm recursively expands the transmission subtree by adding the vertices which are in I_v to V^E if $next_v \leq t$, starting with $v = r$ and then repeating this on its children in V^E. Observe for the first service occurring at t_1 that none of the sets I_v are added to V^E, as $next_v = \infty$ initially.

Algorithm 1 $e(D+1)$-competitive algorithm for MLAP-D.

procedure INITIALIZATION
  for $v \in V(\mathcal{T})$ **do**
    $\ell_v \leftarrow c(v)$
    $next_v \leftarrow \infty$
    $I_v \leftarrow \varnothing$ ▷ Stores set of nodes which v invested in most recently

procedure ONDEADLINE ▷ Unserved request ρ reaches deadline at time $t = d_\rho$
  Initialize V^E as the set of vertices on path from r to v_ρ ▷ V^E is a global variable
  $V^E \leftarrow$EXPANSIONSTAGE(V^E, r)
  Initialize $V^I \leftarrow \varnothing$ ▷ V^I is a global variable
  $V^I \leftarrow$INVESTMENTSTAGE(V^I, V^E, r)
  return $T \leftarrow \mathcal{T}[V^E \cup V^I]$ ▷ Transmit T

procedure EXPANSIONSTAGE(V^E, v)
  if $t \geq next_v$ **then** ▷ t denotes current time
    for $v' \in I_v$ **do**
      Add all vertices on the path from v to v' to V^E
  for child nodes v' of v in V^E **do**
    $V^E \leftarrow$EXPANSIONSTAGE(V^E, v')
  return V^E

procedure INVESTMENTSTAGE(V^I, V^E, v)
  for child nodes v' of v in V^E **do** ▷ Invest for deeper nodes first
    $V^I \leftarrow$INVESTMENTSTAGE(V^I, V^E, v')
  $I_v \leftarrow \varnothing$ ▷ Begin investing into new set of nodes
  $b_v \leftarrow \theta \cdot c(v)$ ▷ $\theta > 0$ is fixed parameter, for best competitive ratio set $\theta = D$
  while $b_v > 0$ and there are pending requests on vertices in $V(\mathcal{T}_v) \setminus (V^E \cup V^I)$ **do**
    Let ρ' be the pending request in $V(\mathcal{T}_v) \setminus (V^E \cup V^I)$ with the earliest deadline
    Let v' be first vertex on path from v to $v_{\rho'}$ not in $V^E \cup V^I$ ▷ $v' \in V(\mathcal{T}_v)$
    $V^I \leftarrow$ INVEST(v, v', V^I)
  $next_v \leftarrow \min\left(\{d_\rho : v_\rho \in V(\mathcal{T}_v) \setminus (V^E \cup V^I)\} \cup \{\infty\}\right)$
  return V^I

procedure INVEST(v, v', V^I) ▷ Invest in v' using b_v and update V^I if v' is served
  $\Delta b_v \leftarrow \min\{b_v, \ell_{v'}\}$ ▷ Put as much of budget as possible towards buying v'
  $b_v \leftarrow b_v - \Delta b_v$
  $\ell_{v'} \leftarrow \ell_{v'} - \Delta b_v$
  $I_v \leftarrow I_v \cup \{v'\}$ ▷ Record that v has invested in v'
  if $\ell_{v'} = 0$ **then** ▷ If v' is served
    $V^I \leftarrow V^I \cup \{v'\}$
    $\ell_{v'} \leftarrow c(v')$
  return V^I

We now describe the investment stage. Figuratively speaking, each vertex v' has a price $c(v')$ which the algorithm must invest into v' before v' can be added to V^I. We keep track of the remaining cost for a vertex v' using the variable $\ell_{v'}$; $\ell_{v'}$ starts at $c(v')$, is decreased with every investment, and is then reset to $c(v')$ after v' has been added to V^I. For each node $v \in V^E$ (starting with the deepest nodes and working upwards), we allocate a budget proportional to $c(v)$ for the algorithm to put towards investing into vertices, and once $\ell_{v'} = 0$, we add v' to V^I. The bookkeeping for the spending of budget is done within the INVEST procedure. The allocated budget for $v \in V^E$ is $\theta \cdot c(v)$, where for our competitive analysis, we find that the best competitive guarantee is obtained when $\theta = D$. An important property of the algorithm is that the budget of $v \in V^E$ is only spent on vertices deeper than v, and it invests towards the requests in non-decreasing order of deadline. The algorithm stores the set of vertices which $v \in V^E$ invested in into the variable I_v. Once a given node $v \in V^E$ has spent its budget, it sets $next_v$ as the earliest deadline among the remaining unserved requests beneath v, or to ∞ if there are no such active requests. Therefore, if the next time that v is added during the expansion stage occurs at time t', then if $t' \geq next_v$, the algorithm will also add I_v during the expansion stage.

The following terms and notations will be used throughout the analysis.

▶ **Definition 3** (Expansion vertices, Investment vertices)**.** *Consider the transmission subtree $T_i = \mathcal{T}[V^E \cup V^I]$ transmitted by Algorithm 1 at time t_i. A vertex v is an* expansion vertex at time t_i *if $v \in V^E$, and an* investment vertex at time t_i *if $v \in V^I$. The set of expansion vertices at time t_i will be denoted V_i^E, and the set of investment vertices at time t_i will be denoted V_i^I.*

As the variables I_v and $next_v$ change value within a single timestep, we make the following definition to refer to the values at specific times unambiguously.

▶ **Definition 4** ($I_{v,i}, I'_{v,i}, next_{v,i}, next'_{v,i}$)**.** *If Algorithm 1 transmits a service subtree at time t_i, we define $I_{v,i}$ (respectively, $I'_{v,i}$) to be the value of I_v before (respectively, after) the function call* ONDEADLINE *at t_i. $next_{v,i}$ and $next'_{v,i}$ are defined analogously.*

Each set of expansion vertices V_i^E is partitioned into two disjoint sets.

▶ **Definition 5** (Anticipated vertices, unanticipated vertices)**.** *We call a vertex $v \in V_i^E$* anticipated at time t_i *if $next_{v,i} \leq t_i$. We call a vertex $v \in V_i^E$* unanticipated at time t_i *if it is an expansion vertex at time t_i and $next_{v,i} > t_i$. We denote the set of vertices anticipated at time t_i as A_i, and those which are unanticipated as $\bar{A}_i$, so that $V_i^E = A_i \cup \bar{A}_i$ is a disjoint union.*

Our analysis will be split into two parts. First, we express ALG solely in terms of the nodes that are unanticipated. The following lemma is proved in Section 2.1.

▶ **Lemma 6.**

$$ALG \leq \left(1 + \frac{1}{\theta}\right)^D (1+\theta) \sum_{i=1}^{\ell} c(\bar{A}_i).$$

In Section 2.2, we use a charging argument to show that the costs of unanticipated expansion vertices across time can be charged directly to OPT without double-charging. We prove the following.

▶ **Lemma 7.**

$$\sum_{i=1}^{\ell} c(\bar{A}_i) \leq OPT.$$

Combining Lemmas 6 and 7 implies

$$\text{ALG} \leq \left(1 + \frac{1}{\theta}\right)^{D} (1+\theta)\text{OPT}.$$

The best choice is $\theta = D$ which implies Theorem 1. The proof is provided in the full version.

Before proceeding with the proofs for Lemmas 6 and 7, we highlight some important properties of Algorithm 1, which we will refer to throughout the analysis. We give brief justifications of each after stating them.

▶ **Observation 8.** *For all t_i, the subgraph of $\mathcal{T}$ induced by V_i^E is a subtree rooted at r.*

Observation 8 follows because r is added to V^E during OnDeadline, and whenever we add v to V^E, we add the entire u-v path where $u \in V^E$ already, which maintains connectivity of V^E.

▶ **Observation 9.** *For all t_i and $v \in V(\mathcal{T})$, we have $I_{v,i} \subseteq V(\mathcal{T}_v) \setminus \{v\}$.*

Observation 9 holds at t_1 because in Initialization, I_v is set to $\varnothing$. The variable I_v only changes at times t_i when $v \in V_i^E$. Then, during a call InvestmentStage(V^I, V^E, v) at t_i, we only invest in vertices in $\mathcal{T}_v$. Additionally, v could not invest into itself as $v \in V_i^E$ already, and hence $I_v \subseteq V(\mathcal{T}_v) \setminus \{v\}$ at all moments of the algorithm.

▶ **Observation 10.** *For all t_i, if $v \in V_i^E$ and $v' \in I'_{v,i}$, then all ancestors of v' are in T_i.*

Observation 10 is justified as follows. Consider $v \in V_i^E$ and $v' \in I'_{v,i}$. Then Invest(v, v', V^I) must have been called for $b_v > 0$ at time t_i. Due to line 29, v' was the first vertex on v-v' path not in $V^E \cup V^I$, meaning that all ancestors of v' must be in T_i already.

▶ **Observation 11.** *For all t_i, if $u, v \in V_i^E$ and u is an ancestor of v with $I'_{u,i} \cap \mathcal{T}_v \neq \varnothing$, then $next'_{u,i} \geq next'_{v,i}$.*

Observation 11 is justified as follows. Assume for now that $next'_{v,i} \neq \infty$, so $next'_{v,i} = d_\rho$ where ρ has the earliest deadline among unserved requests in $\mathcal{T}_v \setminus (V^E \cup V^I)$ after the investment stage at v has completed at t_i. Observe that the investment stage for v occurs before the investment stage for u, as u is an ancestor of v. Therefore, when the investment stage for u begins, the earliest deadline among unserved request in $\mathcal{T}_v \setminus (V^E \cup V^I)$ has a deadline at earliest d_ρ. Because the investment stage invests towards requests in non-decreasing order of deadline, it follows that if $I'_{u,i} \cap \mathcal{T}_v \neq \varnothing$, then at some point the request in $\mathcal{T}_u \setminus (V^E \cup V^I)$ with earliest deadline was located at a vertex in $\mathcal{T}_v$. But this implies that at the conclusion of the investment stage for u that all requests in $\mathcal{T}_u \setminus (V^E \cup V^I)$ have deadline at earliest d_ρ. Therefore, $next_u$ would be set no earlier than d_ρ, and therefore $next'_{u,i} \geq d_\rho = next'_{v,i}$. For the other case, if $next'_{v,i} = \infty$, then after the investment stage for v there were no requests in $\mathcal{T}_v \setminus (V^E \cup V^I)$, and so $I'_{u,i} \cap \mathcal{T}_v \neq \varnothing$ is impossible.

2.1 Upper-bounding ALG

In this section, we prove Lemma 6.

Observe that an investment vertex v' at time t_i is only added to V_i^I if a total of $c(v')$ is invested by expansion vertices to reduce the variable $\ell_{v'}$ to zero. As each expansion vertex v can invest no more than $\theta \cdot c(v)$, we obtain

$$\sum_{i=1}^{\ell} c(V_i^I) \leq \sum_{i=1}^{\ell} \theta \cdot c(V_i^E).$$

This yields

$$\text{ALG} = \sum_{i=1}^{\ell} c(T_i) = \sum_{i=1}^{\ell} [c(V_i^E) + c(V_i^I)] \leq \sum_{i=1}^{\ell} [c(V_i^E) + \theta \cdot c(V_i^E)] = (1+\theta) \sum_{i=1}^{\ell} c(V_i^E). \quad (1)$$

Now that we have expressed the cost solely in terms of the expansion vertices, we wish to express the cost solely in unanticipated expansion vertices, as these costs can then be directly associated with OPT in Section 2.2. To do so, we must move the costs of anticipated nodes to those which were unanticipated. We first show the following two lemmas.

▶ **Lemma 12.** *Let $v \in A_i$ be an anticipated vertex at time t_i, then v must have been an expansion vertex at a time $t_j < t_i$, in other words $v \in V_j^E$ for some $j < i$.*

Proof. Suppose $v \in A_i$. By definition this means that $next_{v,i} \leq t_i$, and hence clearly $next_{v,i} \neq \infty$. As the $next_v$ variable is originally assigned to ∞ in the INITIALIZATION of the algorithm, it follows that $next_v$ must have had its value reassigned prior to t_i. As the algorithm's variable $next_v$ is only reassigned at times for which v is an expansion vertex, we therefore know that v must have been an expansion vertex at some time before t_i. ◀

▶ **Lemma 13.** *Let $v \in A_i$ be an anticipated node at time t_i, and let t_j be the latest time before t_i that v was an expansion vertex (see Lemma 12). Let $\theta c(v)$ be the budget assigned to v in the investment stage for v. Then $c(I_{v,i}) \geq \theta c(v)$.*

Proof. Suppose $v \in A_i$. According to Lemma 12, we know that v must have been an expansion vertex at a time earlier than t_i. Let t_j be the most recent time before t_i for which $v \in V_j^E$.

As the variable $next_v$ is not changed between t_j and t_i, we know that $next'_{v,j} = next_{v,i}$, and hence $next'_{v,j} \neq \infty$ also. Therefore, we know that after leaving the while loop on line 27 within the INVESTMENTSTAGE(V^I, V^E, v) call at time t_j, it must be the case that there were still pending requests on vertices in $\mathcal{T}_v - (V^E \cup V^I)$, or else $next'_{v,j}$ would have been set to ∞ in line 31. Thus, it must instead be the case that the condition $b_v > 0$ of the while loop failed, and so the entirety of the budget assigned to v at time t_j was spent. Observe that each call of INVEST(v, v', V^I) will decrease the budget b_v by Δb_v, but will add v' to I_v, where $c(v') \geq \ell_{v'} \geq \Delta b_v$. Thus the amount by which the budget decreases is always at most the amount by which $c(I_v)$ increases. As during the iteration at time t_j, b_v must have decreased by at least $\theta c(v)$, it follows that $c(I_v)$ must have increased by at least $\theta c(v)$ during the investment stage. As I_v is set to $\varnothing$ at the beginning of the investment stage, we know $c(I_v) = 0$ initially, and thus if $c(I_v)$ increases by at least $\theta c(v)$ during the investment phase, we know $c(I'_{v,j}) \geq \theta c(v)$. Observing that $c(I'_{v,j}) = c(I_{v,i})$ finishes the proof. ◀

We can now express the cost of all expansion nodes only in terms of the unanticipated nodes.

▶ **Lemma 14.** *For any t_i, we have*

$$c(V_i^E) = \sum_{v \in V_i^E} c(v) \leq \left(1 + \frac{1}{\theta}\right)^D \sum_{v \in \bar{A}_i} c(v) = \left(1 + \frac{1}{\theta}\right)^D c(\bar{A}_i)$$

Proof. We proceed using the accounting method of amortized analysis. For a fixed t_i, consider placing coins of value $c(v)$ on each anticipated vertex $v \in V_i^E$. Thus the total cost initially on the structure V_i^E equals the left-hand side of the inequality stated by the Lemma. We will redistribute the costs which are on anticipated nodes, to obtain a rearrangement of costs which is upper-bounded by the larger quantity of the lemma.

We start with the root r. We *process* r as follows. If r is an unanticipated node of T_i, we do not move the cost from $c(r)$ – we will only move the costs off of anticipated nodes. Otherwise if r is an anticipated node, then by Lemma 13, $c(r) \leq \frac{1}{\theta}c(I_{r,i}) = \frac{1}{\theta}\sum_{v' \in I_{r,i}} c(v')$. If r is anticipated, then we know that all the nodes in $I_{r,i}$ were added to V_i^E, and so we can reallocate this cost of $c(r)$ from r as an additional cost of at most $\frac{c(v')}{\theta}$ to each $v' \in I_{r,i}$.

We descend the vertices in V_i^E and repeat this process: we call a expansion node v "ready to redistribute" if all of its ancestors have been processed. To process an unanticipated $v \in \bar{A}_i$, we do nothing and the cost remains on v. For any anticipated $v \in A_i$ currently allocated a cost of $C \cdot c(v)$, we process v by reallocating this as an additional cost of $\frac{C}{\theta} \cdot c(v')$ to each node v' in $I_{v,i}$. Recall by Observation 9 that v will only invest into vertices in $\mathcal{T}_v$, and so these costs are strictly pushed to deeper vertices.

▷ Claim 15. When a vertex $v \in V_i^E$ at depth d is ready to redistribute but has not yet been processed, the allocated costs on vertices $v' \in V_i^E \cap \mathcal{T}_v$ is at most $(1 + 1/\theta)^d c(v')$.

Proof. We proceed by induction on d. The base case of $d = 0$ is trivial, as this corresponds to the fact that all vertices in $v \in V_i^E$ have an allocated cost of $c(v)$ before the root r is processed.

Assume inductively that the claim has held true up to a depth d, and a vertex v at depth $d + 1$ is ready to redistribute. This means that when the parent of v, call it u, was ready to redistribute, all nodes $w \in V(\mathcal{T}_u)$ had allocated at most $(1 + 1/\theta)^d c(w)$. If u is unanticipated, then the cost on u is not moved, so none of the vertices in $\mathcal{T}_u$ receive any additional costs. This would imply that the costs on vertices w in $\mathcal{T}_v$ would still be at most $(1 + 1/\theta)^d c(w) \leq (1 + 1/\theta)^{d+1} c(w)$, and so the claim will continue to hold after u is processed. Otherwise, consider what would occur when u was processed if u was anticipated. An additional cost of at most $\frac{1}{\theta}(1 + 1/\theta)^d c(w)$ would be added to all vertices in $w \in \mathcal{T}_u$ (recall that the cost $C \cdot c(u)$ of u is reallocated as an additional cost of $\frac{C}{\theta} \cdot c(w)$ to each node $w \in I_{u,i}$). Therefore immediately following the processing of u, each $w \in V(\mathcal{T}_u)$ would have had a cost of at most $(1 + 1/\theta)^d c(w) + \frac{1}{\theta}(1 + 1/\theta)^d c(w) = (1 + 1/\theta)^{d+1} c(w)$. At this point, v would be ready to redistribute, and as $\mathcal{T}_v$ is a subtree of $\mathcal{T}_u$, the inductive step is finished and we have proven Claim 15. ◁

This redistribution process must terminate eventually, as the tree $\mathcal{T}$ has a bounded depth of D. Observe that this redistribution cost will leave anticipated vertices of V_i^E with no costs, while an unanticipated vertex v in V_i^E at depth d would be left with a cost of $(1 + 1/\theta)^d c(v) \leq (1 + 1/\theta)^D c(v)$. As we have at no point decreased the total cost on the structure V_i^E, the inequality stated by the lemma follows. ◀

Combining inequality (1) and Lemma 14 implies Lemma 6.

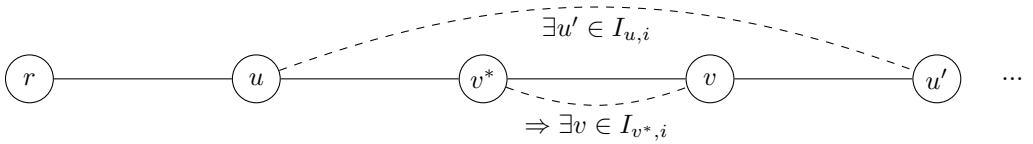
2.2 Charging Scheme against OPT

In this section, we prove Lemma 7.

We accomplish this using a charging argument to map the costs of unanticipated expansion vertices at times t_i to OPT. However, before we can do so, we need to further characterize the unanticipated expansion nodes, which we accomplish in the following lemma.

▶ **Lemma 16.** *Let ρ_i^* denote the pending request which triggered* OnDeadline *at t_i. Then for any $v \in \bar{A}_i$, either v is on the r-$v_{\rho_i^*}$ path, or $v \in I_{v^*,i}$ for some $v^* \in A_i$.*

Proof. Observe that there are only 2 cases in Algorithm 1 for which a vertex v can be added to V_i^E; (1) during OnDeadline, if v is on the path r-$v_{\rho_i^*}$ and (2) during ExpansionStage(V^I, V^E, u), if v is on the path u-u' where $u \in A_i$ and $u' \in I_{u,i}$. Case (1) clearly satisfies the lemma, so we will now assume the second case, that v lies on the path u-u' where $u \in A_i$ and $u' \in I_{u,i}$. As the lemma is obvious for $v = u'$, we continue under the assumption that $v \neq u'$.



By Lemma 12, there must be a time $t_{i^*} < t_i$ such that $u \in V_{i^*}^E$, let t_{i^*} be the most recent such time, and observe that $next'_{u,i^*} = next_{u,i}$ and $I'_{u,i^*} = I_{u,i}$. As $u' \in I_{u,i} = I'_{u,i^*}$, by Observation 10 we know that $v \in T_{i^*}$. We now make an important claim.

▷ Claim 17. For any w on the path u-u' (where $u \in A_i$, $u \in V_{i^*}^E$, and $u' \in I_{u,i} = I'_{u,i^*}$, and t_{i^*} defined as before) if $w \in V_{i^*}^E$ then $w \in A_i$, $next'_{w,i^*} = next_{w,i}$ and $I'_{w,i^*} = I_{w,i}$.

Proof. By Observation 11, we have that $next'_{w,i^*} \leq next'_{u,i^*}$, as w is on the u-u' path. We know that $w \in V_{i^*}^E$, and we also know that $w \in V_i^E$, because w is on the u-u' path where $u, u' \in V_i^E$, and V_i^E forms a subtree due to Observation 8. Further observe that it is impossible for w to be anticipated at any time t with $t_{i^*} < t < t_i$, because this would necessitate that u would be anticipated at time t by Observation 8, which would contradict the fact that t_{i^*} is the latest time before t_i for which u is anticipated. With this observation, it follows that $next'_{w,i^*} = next_{w,i}$. Putting this together, we have that $next_{w,i} = next'_{w,i^*} \leq next'_{u,i^*} = next_{u,i} \leq t_i$, where we used the fact that $u \in A_i$. Hence, $w \in A_i$ by definition, proving the claim. ◁

Earlier we found that $v \in T_{i^*} = V_{i^*}^E \cup V_{i^*}^I$. Due to Claim 17, we now know that $v \notin V_{i^*}^E$ as otherwise $v \in A_i$, which would contradict the fact that v is an unanticipated vertex at time t_i.

Thus $v \in V_{i^*}^I$, meaning there must have been some $v^* \in V_{i^*}^E$ such that Invest(v^*, v, V^I) with $b_{v^*} > 0$ was called at time t_{i^*} and resulted in the addition of v into $V_{i^*}^I$, and so $v \in I'_{v^*,i^*}$. Observe that by the time Invest(u, u', V^I) was called, we would have needed all vertices on the path u-u' already present in $V^E \cup V^I$, and hence v^* must have invested in v before Invest(u, u', V^I) was called. Thus, v^* must be at least as deep as u. v^* must also be an ancestor of v in order for v^* to have invested into v by Observation 9, and hence v^* is on the path u-v, and hence also on the path u-u'. Therefore Claim 17 applies, meaning $v^* \in A_i$. Finally, $v \in I'_{v^*,i^*} = I_{v^*,i}$, finishing the proof. ◀

We are now ready to define the charging scheme.

2.2.1 The Charging Scheme

Let $v \in \bar{A}_i$ be an unanticipated expansion vertex at time t_i. Let ρ_i^* be the request which triggered the creation of a service at time t_i, so $d_{\rho_i^*} = t_i$. Using Lemma 16, we know that v is either on the r-$v_{\rho_i^*}$ path, or $v \in I_{v^*,i}$ for $v^* \in A_i$. We thus split the charging scheme into two cases:

- *Case 1: v is on the path r-$v_{\rho_i^*}$.* Then, OPT must have served ρ_i^* within the time $[a_{\rho_i^*}, d_{\rho_i^*}] = [a_{\rho_i^*}, t_i]$. OPT's service to ρ_i^* must have also included v, and so we charge the cost of $c(v)$ against any of these $c(v)$ costs OPT incurs for including v in its service history within $[a_{\rho_i^*}, t_i]$.
- *Case 2: v is not on the path r-v_{ρ_i}, and $v \in I_{v^*,i}$ for some $v^* \in A_i$.* If there are multiple such v^*, we simply fix one arbitrarily for each unanticipated $v \in \bar{A}_i$. By Lemma 12 we know that v^* must have been an expansion vertex at a time before t_i, so suppose that this occurred most recently at $t_{i^*} < t_i$. Then we would have that $next'_{v^*,i^*} = next_{v^*,i}$. As $v^* \in A_i$, we have $next_{v^*,i} \leq t_i$. Now, as $v \in I_{v^*,i} = I'_{v^*,i^*}$ and ALG processes requests in $\mathcal{T}_{v^*}$ in non-decreasing deadline, it follows that while ALG was investing the budget on v^* at time t_{i^*}, there must have been an unserved request ρ on a vertex v_ρ such that v was on the path v^*-v_ρ. Because $next'_{v^*,i^*}$ is determined as the next pending task in $\mathcal{T}_u$ not yet in $V_i^E \cup V_i^I$, we know that $d_\rho \leq next'_{v^*,i^*}$. Linking this together, we find that $d_\rho \leq next'_{v^*,i^*} = next_{v^*,i} \leq t_i$.
  Thus OPT must have served ρ within the time $[a_\rho, d_\rho]$, which is a subset of $[a_\rho, t_i]$ by the above inequality. Such a service must have included v, and thus we charge $c(v)$ to the associated cost of $c(v)$ incurred by OPT during the timeframe $[a_\rho, t_i]$.

In both Case 1 and Case 2, we find ourselves charging $c(v)$ to a portion of OPT's service history within $[a_\rho, t_i]$ which serves ρ where v_ρ is deeper than v. In order to unify these two cases, we call this particular ρ the request *associated* with unanticipated node v at time t_i.

Using the above charging scheme, the cost of any unanticipated vertex at a time t_i is mapped to a cost in OPT. To establish Lemma 7, we are left with showing that this mapping is one-to-one, i.e., no one portion of OPT is charged to more than once.

▶ **Lemma 18.** *The charging scheme for ALG consists of unique charges against OPT.*

Proof. Seeking a contradiction, suppose instead that two distinct charges for a vertex $v \in \bar{A}_i$ and $v \in \bar{A}_j$ coincide, say at times t_j and t_i where $j < i$. We will do a case analysis based on whether the charge at t_i is Case 1 or Case 2. Suppose that ρ_j and ρ_i are the requests *associated* (as defined above) to v at times t_j and t_i respectively. The charges to OPT's history must have taken place between times $[a_{\rho_j}, t_j]$ and $[a_{\rho_i}, t_i]$. Thus, they could only coincide if $a_{\rho_i} < t_j$ (utilizing our assumption of distinct arrivals and deadlines for distinct services). In words, this means ρ_i was present when v was an unanticipated expansion vertex at time t_j. This leads to the following observation used throughout the proof.

▶ **Observation 19.** *For any $t_k > t_j$, if ρ_i was not served before t_k, we must have $next_{v,k} \leq d_{\rho_i}$.*

We split the remainder of the proof into the two cases depending on the type of charging for v which occurs at time t_i. We will show that both lead to contradiction.

Case 1. In this scenario, v is on the critical path due to the expiry of ρ_i, meaning ρ_i was not served before time t_i, and so $k = i$ satisfies the conditions required for Observation 19. Thus $next_{v,i} \leq d_{\rho_i}$. As $d_{\rho_i} = t_i$, this would imply $next_{v,i} \leq t_i$ which would contradict the fact that $v \in \bar{A}_j$.

Case 2. This means at time t_i, for some ancestor $v^* \in A_i$ of v, that $v \in I_{v^*,i}$. Recall by definition this means that the last time v^* was an expansion node prior to t_i, at time $t_{i^*} < t_i$, ρ_i must have been active where $v_{\rho_i} \in \mathcal{T}_v$. We know that $v^* \in V_i^E$, as $v \in V_i^E$ and V_i^E is a subtree by Observation 8. Therefore it must be true that $t_{i^*} \geq t_j$, as t_{i^*} is defined as the latest time before t_j that v^* was an expansion vertex. More strictly, $t_{i^*} > t_j$ as it is impossible for v^* to invest into v at time t_j because $v \in V_j^E$, i.e it is an expansion vertex at time t_j. Hence we know $t_j < t_{i^*} < t_i$.

We know that ρ_i must not have been served before time t_{i^*}, because it was this service which caused v^* to invest into v at time t_{i^*}, by definition of Case 2 charge. Hence by Observation 19, we know that $next_{v,i^*} \leq d_{\rho'}$ as $t_{i^*} > t_j$. Observe further that v could not have been an expansion vertex at time t_{i^*}, otherwise v^* would not have invested into v at t_{i^*}. Therefore the variable $next_v$ would not be affected at time t_{i^*}, meaning $next_{v,i^*} = next'_{v,i^*}$ and thus $next_{v,i^*} \leq d_{\rho_i}$ still. It is impossible for v to be an expansion vertex within the time interval (t_{i^*}, t_i), because this would imply that v^* would be an expansion vertex within (t_{i^*}, t_i) by Observation 8. Thus it follows that the variable $next_v$ is not changed in the interval (t_{i^*}, t_i), and so $next_{v,i} \leq d_{\rho_i}$.

However, from the discussion in the Case 2 charge definition, we already know that $d_{\rho_i} \leq t_i$, and so $next_{v,i} \leq d_{\rho_i} \leq t_i$. This contradicts the fact that $v \in \bar{A}_i$, thereby finishing the proof. ◀

3 The $(4H + 2)e$-competitive Algorithm

In this section, we prove Theorem 2.

We show that Algorithm 1 can be modified in a way which results in the competitive ratio being bounded above by a linear function of the caterpillar dimension H of $\mathcal{T}$. Throughout this section, we assume that we are given a heavy path decomposition $\mathcal{P}$ for $\mathcal{T}$ with dimension H.

This section builds upon the work from Section 2, and thus we assume familiarity with Algorithm 1 and the notations and definitions introduced previously. We also introduce some new pieces of notation.

▶ **Definition 20** ($P(v)$ and $p(v)$)**.** *Let v be a vertex in $\mathcal{T}$. Let $P(v)$ denote the path which v lies on in $\mathcal{P}$. Let $p(v)$ be the intersection of the path $P(v)$ and the path from r to v.*

The motivation behind Algorithm 2 is that if we wish for the competitive ratio to not be parameterized by the depth of $\mathcal{T}$, we cannot follow the accounting method of Lemma 14, because this explicitly descends the tree node-by-node and invariably leads to a function of D. If we wish to parameterize instead in terms of H, then one would need the accounting method to be performed path-by-path instead.

Algorithm 2 accomplishes this aim by allowing the deepest expansion node at time t_i on each path $P \in \mathcal{P}$, which we will denote $v^{\downarrow}_{P,i}$, to potentially have a much larger budget to spend during the investment stage in comparison to Algorithm 1. Instead of the budget for $v^{\downarrow}_{P,i}$ being $\theta_1 \cdot c(v^{\downarrow}_{P,i})$ for some θ_1 as was the case in Algorithm 1, for these nodes the budget at time t_i becomes $\theta_1 \cdot c(p(v^{\downarrow}_{P,i}))$. Our algorithm will specified in terms of two parameters, θ_1 and θ_2, and we will choose their values to optimize our bound on the competitive ratio during the analysis. Additionally, $v^{\downarrow}_{P,i}$ is made to be the only vertex on $P(v^{\downarrow}_{P,i})$ which can make investments on vertices in $\mathcal{T}_{v^{\downarrow}_{P,i}}$. If $v \in P$ is not the deepest expansion node on $P(v)$, and if the next unserved request is within $\mathcal{T}_{v^{\downarrow}_{P(v),i}}$, then we stop making investments for v, which is done in line 19. The idea is that because $v^{\downarrow}_{P,i}$ has a disproportionally large budget, we can simply use the investments from $v^{\downarrow}_{P,i}$ for v as well.

▶ **Definition 21** ($v^{\downarrow}_{P,i}$, Disproportional vertices, recently disproportional vertices)**.** *We define $v^{\downarrow}_{P,i}$ to be the deepest node in $P \cap V^E_i$, where $P \in \mathcal{P}$. We say a vertex v is* disproportional *at time t_i if $v = v^{\downarrow}_{P(v),i}$. That is, v is the deepest expansion vertex at time t_i in $P(v)$. Let $v \in A_i$, and let $t_j < t_i$ be the latest time before t_i that v was an expansion vertex (which must exist due to Lemma 12). Then we say v was* recently disproportional *at time t_i if v was disproportional at time t_j, i.e $v = v^{\downarrow}_{P(v),j}$.*

Algorithm 2 $(4H+2)e$-competitive algorithm for MLAP-D.

```
1: ———— The following procedures are identical to Algorithm 1 ————
2: procedure INITIALIZATION
3: procedure ONDEADLINE
4: procedure INVEST(v, v', V^I)
5: procedure EXPANSIONSTAGE(V^E, v)
6:
7: procedure MODIFIEDINVESTMENTSTAGE(V^I, V^E, v)  ▷ Modified sections are colored
8:     for child nodes v' of v in V^E do            ▷ Invest for deeper nodes first
9:         V^I ← MODIFIEDINVESTMENTSTAGE(V^I, V^E, v')
10:    I_v ← ∅                                      ▷ Begin investing into new set of nodes
11:    Let v^↓_{P(v)} denote the deepest vertex in V^E ∩ P(v)
12:    if v = v^↓_{P(v)} then ▷ θ1 > 0 is parameter, for best competitive ratio set θ1 = 2H + 1
13:        b_v ← θ1 · c(p(v))
14:    else                   ▷ θ2 > 0 is parameter, for best competitive ratio set θ2 = 2H
15:        b_v ← θ2 · c(v)
16:    while b_v > 0 and there are pending requests on vertices in V(T_v) \ (V^E ∪ V^I) do
17:        Let ρ' be the pending request in V(T_v) \ (V^E ∪ V^I) with the earliest deadline.
18:        Let v' be first vertex on path from v to v_ρ' not in V^E ∪ V^I      ▷ v' ∈ V(T_v)
19:        if v ≠ v^↓_{P(v)} but v' ∈ T_{v^↓_{P(v)}} then
20:            I_v ← I_{v^↓_{P(v)}}                 ▷ Utilize investments from v^↓_{P(v)} instead
21:            Escape while-loop
22:        V^I ← INVEST(v, v', V^I)
23:    next_v ← min({d_ρ : v_ρ ∈ V(T_v) \ (V^E ∪ V^I)} ∪ {∞})
24:    return V^I
```

Similar to the proof of Theorem 1, our analysis is split into two parts. First, we express ALG in terms of the unanticipated nodes. The following lemma is proven in the full version.

▶ **Lemma 22.**

$$ALG \leq \left(1+\frac{1}{\theta_1}\right)^{H+1}\left(1+\frac{1}{\theta_2}\right)^{H}(1+\theta_1+\theta_2)\sum_{i=1}^{\ell} c(\bar{A}_i).$$

Second, we use the same charging argument for Lemma 7 to show that the costs of unanticipated expansion vertices across time can be charged directly to OPT without double-charging. Combining Lemmas 22 and 7 implies

$$\text{ALG} \leq \left(1+\frac{1}{\theta_1}\right)^{H+1}\left(1+\frac{1}{\theta_2}\right)^{H}(1+\theta_1+\theta_2)\text{OPT}.$$

The best choice is $\theta_1 = 2H+1$ and $\theta_2 = 2H$ which implies Theorem 2. The proof is provided in the full version.

Crucially, Algorithm 2 has built upon Algorithm 1 using an approach which allows the key lower bound on OPT provided by Lemma 7 to continue to hold. To see why the analysis of Section 2.2 can be applied identically to Algorithm 2, observe that Observations 8, 9, 10 and 11 are also true for Algorithm 2. Then, because Section 2.2 only utilizes these

observations rather than any features particular to either algorithm, we see that Lemma 7 provides a robust lower bound for OPT. Therefore, Algorithm 1 can be used as a generic framework for MLAP-D algorithms.

4 Conclusion

This paper developed a memory-based framework for online MLAP-D, which allowed for a direct analysis in the case of general trees, and improved on competitiveness results. To show the framework was flexible, we demonstrated that it could be modified to provide competitive ratios linear in caterpillar dimension, a measure which unifies several results for special classes of graph which had been considered previously for MLAP-D. Future research could extend the memory-based framework to the problem of MLAP with delay costs, which would likely tighten the upper bounds of competitive ratios for this problem. The framework could also be modified to work for other special tree classes. The most notable open problem in this area remains the wide asymptotic gap for the competitive ratio, where the best-known lower bound remains constant.

References

1 Alok Aggarwal and James K Park. Improved algorithms for economic lot size problems. *Operations research*, 41(3):549–571, 1993. doi:10.1287/OPRE.41.3.549.

2 Esther Arkin, Dev Joneja, and Robin Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations research letters*, 8(2):61–66, 1989.

3 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 60–71. IEEE, 2019. doi:10.1109/FOCS.2019.00013.

4 Yossi Azar and Noam Touitou. Beyond tree embeddings–a deterministic framework for network design with deadlines or delay. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

5 BR Badrinath. Gathercast: The design and implementation of a programmable aggregation mechanism for the internet. In *Proceedings Ninth International Conference on Computer Communications and Networks (Cat. No. 00EX440)*, pages 206–213. IEEE, 2000.

6 Luca Becchetti, Alberto Marchetti-Spaccamela, Andrea Vitaletti, Peter Korteweg, Martin Skutella, and Leen Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms (TALG)*, 6(1):1–20, 2009. doi:10.1145/1644015.1644028.

7 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiri Sgall, Kim Thang Nguyen, and Pavel Veselỳ. Online algorithms for multi-level aggregation. In *The 24th European Symposium on Algorithms (ESA)*, 2016.

8 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. Online algorithms for multilevel aggregation. *Operations Research*, 68(1):214–232, 2020. doi:10.1287/OPRE.2019.1847.

9 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselỳ. New results on multi-level aggregation. *Theoretical Computer Science*, 861:133–143, 2021. doi:10.1016/J.TCS.2021.02.016.

10 Marcin Bienkowski, Jarosław Byrka, Marek Chrobak, Neil Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Świrszcz, and Neal E Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015. doi:10.1007/S10951-014-0392-Y.

11 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 42–54. SIAM, 2014.

12 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Workshop on Algorithms and Data Structures*, pages 133–145. Springer, 2013.

13 Edward Bortnikov and Reuven Cohen. Schemes for scheduling control messages by hierarchical protocols. *Computer Communications*, 24(7-8):731–743, 2001. doi:10.1016/S0140-3664(00)00276-0.

14 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64:584–605, 2012. doi:10.1007/S00453-011-9567-5.

15 Niv Buchbinder, Moran Feldman, Joseph Naor, and Ohad Talmon. O (depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1235–1244. SIAM, 2017.

16 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *European Symposium on Algorithms*, pages 253–264. Springer, 2007. doi:10.1007/978-3-540-75520-3_24.

17 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Operations Research*, 61(4):1014–1029, 2013. doi:10.1287/OPRE.2013.1188.

18 Wallace B Crowston and Michael H Wagner. Dynamic lot size models for multi-stage assembly systems. *Management Science*, 20(1):14–21, 1973.

19 Daniel R Dooly, Sally A Goldman, and Stephen D Scott. Tcp dynamic acknowledgment delay (extended abstract) theory and practice. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 389–398, 1998.

20 Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgment and other stories about $e/(e-1)$. *Algorithmica*, 36:209–224, 2003.

21 Sanjeev Khanna, Joseph Seffi Naor, and Dan Raz. Control message aggregation in group communication protocols. In *Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*, pages 135–146. Springer, 2002.

22 Alf Kimms. *Multi-level lot sizing and scheduling: methods for capacitated, dynamic, and deterministic models*. Springer Science & Business Media, 2012.

23 Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008. doi:10.1287/MNSC.1070.0781.

24 Retsef Levi, Robin O Roundy, and David B Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–285, 2006. doi:10.1287/MOOR.1050.0178.

25 Jiří Matoušek. On embedding trees into uniformly convex banach spaces. *Israel Journal of Mathematics*, 114(1):221–237, 1999.

26 Jeremy McMahan. A d-competitive algorithm for the multilevel aggregation problem with deadlines, 2021. arXiv:2108.04422.

27 Tim Nonner and Alexander Souza. Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications*, 1(02):153–173, 2009. doi:10.1142/S1793830909000130.

28 L. L. C. Pedrosa. Private communication reported in [7], 2013.

29 Marc P Renault and Adi Rosén. On online algorithms with advice for the k-server problem. *Theory of Computing Systems*, 56(1):3–21, 2015. doi:10.1007/S00224-012-9434-Z.

30 DD SLEATOR and RE TARJAN. A data structure for dynamic trees. *Journal of computer and system sciences (Print)*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

31 Harvey M Wagner and Thomson M Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.

Semirandom Planted Bipartite Subgraphs

Anand Louis
Indian Institute of Science, Bengaluru, India

Kirtan Vora
Indian Institute of Science, Bengaluru, India

Abstract

There have been many recent works studying planted subgraphs problems. The *semirandom planted bipartite subgraph problem* is defined as follows. Starting with a vertex set V, an arbitrary subset $S \subset V$ of size k is chosen, then an arbitrary bipartite graph is added on S. After this between each pair of vertices in $S \times (V \setminus S)$ an edge is added independently with probability p, then an arbitrary graph is added on $V \setminus S$. The analogous semirandom planted clique problem, where S forms a clique, has been studied starting with the work of Feige and Kilian [6]; recent work by [3, 11] gave an algorithm for this problem when $k = \Omega(\sqrt{n \log n})$. We give an algorithm for semirandom planted bipartite subgraph problem when $k = \Omega(\sqrt{n \log n})$ and the two color classes are roughly balanced.

Our algorithms are essentially the same as the elegant greedy algorithm of [3]. We generalize their idea to our setting. Handling the arbitrary nature of the bipartite graph requires some new technical ideas and is our main technical contribution.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Semirandom Models, Spectral Algorithms, Planted Subgraphs, Random Graphs, Approximate Recovery Algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.32

Funding *Anand Louis*: Supported in part by SERB Award CRG/2023/002896 and the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).
Kirtan Vora: Supported by Prime Minister's Research Fellowship, India.

1 Introduction

One of the well known NP-Hard problem in computer science is the problem of finding the maximum independent set (MIS) in a graph. Consider a graph $G = (V, E)$ then unless $P = NP$, [12] showed that for every $\epsilon > 0$ it is hard to approximate MIS within $n^{1-\epsilon}$. Another such problem which is known to be NP-Hard [18] is to find the largest induced bipartite subgraph of the graph $G = (V, E)$.

A series of works on semirandom models have partially explained the tractability of finding the maximum clique problem. An instance of the planted clique problem is sampled as follows: we start with the set of n vertices from which we choose an arbitrary subset S of k vertices and add a clique on them, for any pair of vertices, both of them simultaneously, not in S we add an edge independently with probability p (e.g. $p = 1/2$). [2] designed a spectral algorithm to find the planted clique when $k = \Omega(\sqrt{n})$ but the spectral algorithm is not robust to the significant change in the edges outside of the clique. To study the robustness of clique finding algorithms, Feige and Kilian [6] introduced an insightful and influential semirandom model. In one version of this model, the edges between S and $V \setminus S$ (where V is the set of vertices), also known as cut edges or $cut(S)$, are added independently with probability p, and the edges within $V \setminus S$ can be configured adversarially with access to the random edges between S and $V \setminus S$. In this model S cannot be uniquely recovered since $V \setminus S$ can have $(n - k)/k$ cliques of size k whose cut edges are drawn independently with probability p.

© Anand Louis and Kirtan Vora;
licensed under Creative Commons License CC-BY 4.0

The problem of finding a small list (of size $\frac{n}{k}(1+o(1))$) of cliques of size k in which the planted clique is present has been studied in [6, 5, 16, 4, 3, 11]. [3] gave an algorithm for this problem that works with probability at least $1-1/n$ over the input when $k=\Omega(\sqrt{n}\log^2 n)$. [11] gave an algorithm for this problem that works with probability 0.99 over the input when $k=\Omega(\sqrt{n\log n})$. [17] proved that it is information-theoretically impossible to recover the planted clique when $k=o(\sqrt{n})$

[15] considered the problem of finding an arbitrary planted r-colorable k size subgraph S of G on n vertices such that the edges with both endpoints contained in $V\setminus S$ are sampled independently with probability p, and the edges between S and $V\setminus S$ can be configured adversarially with access to random edges contained in $V\setminus S$. This work designed a polynomial time algorithm to find a subgraph T of G of size at most $(1+\delta)k$ whose intersection with S is at least $(1-\delta)k$ whenever $k=\Omega_{p,\delta}(r\sqrt{n/p})$. Their proof follows by showing that for a novel SDP relaxation of the problem, for which most of the mass is supported on S. Their results also hold in the presence of monotone adversary, i.e., adversary who can arbitrary add edges in $V\setminus S$.

[9] considers the model in which an arbitrary bipartite subgraph on S is planted and the graph on $V\setminus S$ is arbitrary, and then the edges between S and $V\setminus S$ are sampled independently at random with probability p. Note that the model considered by [9] is different from the model proposed by [6] since in this model, adversary doesn't have access to the random edges between S and $V\setminus S$. [9] gave an algorithm for the problem when S is a bipartite graph of size $(1-\epsilon)n$ and recovered a bipartite graph of size $(1-O(\epsilon/p^2))n$ whenever $\epsilon=\Omega(\log n/n)$.

In this paper inspired from [6], we consider the model when the cut edges between the 2-colorable graph and the rest of graph are sampled uniformly and independently at random with probability 1/2. The recent progress on the random planted r-colorable graph problem naturally motivates us to study the Semirandom Planted Bipartite subgraph (SRPB) problem and Semirandom Planted Tripartite subraph (SRPT) problem mentioned below.

▶ **Definition 1** (Semirandom Planted Bipartite Graph, SRPB(n,k,p,t)). *(See figure 1) To an empty graph G with vertex set $V=[n]$:*

1. *Plant a bipartite graph $G(S_1,S_2,E)$: Plant a bipartite graph on a random subset of vertices $S\subseteq V$ with $|S|=k, S=S_1\cup S_2$, where S_1 and S_2 are two independent disjoint components of S in which $1\le |S_2|/|S_1|\le t$.*
2. *Include cut edges at random: Add each edge to $cut(S)$ independently with probability p.*
3. *Choose the rest of the edges adversarially: Adaptively choose any induced graph on $V\setminus S$.*

Our main results are following:

▶ **Theorem 2.** *There exists a polynomial time algorithm such that for every $\epsilon>0$ and $t\in[1,3)$, given an instance of SRPB$(n,k,1/2,t)$, the algorithm outputs a n^4 size list of graphs which contains a subgraph W of G such that $|S\setminus W|=O(\epsilon k)$ and $|W\setminus S|=o(k)$ whenever $k\ge\Omega\left(\frac{1}{\sqrt{\epsilon^{1/2}(3-t)^4}}n^{3/4}\log n\right)$ with probability $1-1/n$ over the instance of SRPB$(n,k,1/2,t)$.*

▶ **Theorem 3.** *There exists a polynomial time algorithm such that for every $\epsilon>0$ and $t\in[1,3)$, given an instance of SRPB$(n,k,1/2,t)$, the algorithm outputs a n^{12} size list of graphs which contains a subgraph W of G such that $|S\setminus W|=O(\epsilon k)$ and $|W\setminus S|=o(k)$ whenever $k\ge\Omega\left(\frac{1}{\epsilon^{1/2}(3-t)^2}\sqrt{n\log n}\right)$ with probability 0.99 over then instance of SRPB$(n,k,1/2,t)$.*

We also study the 3-colorable version of the problem which is added here.

▶ **Definition 4** (Semirandom Planted Tripartite Subgraph, SRPT(n,k,p)). *To an empty graph G with vertex set $V = [n]$:*

1. *Plant a tripartite graph $G(S_1, S_2, S_3, E)$: Plant a tripartite graph on a random subset of vertices $S \subseteq V$ with $|S| = k, S = S_1 \cup S_2 \cup S_3$, where S_1, S_2 and S_3 are three independent disjoint components of S in which $|S_2| = |S_1| = |S_3| = k/3$.*
2. *Include cut edges at random: Add each edge to $cut(S)$ independently with probability p.*
3. *Choose the rest of the edges adversarially: Adaptively choose any induced graph on $V \setminus S$.*

▶ **Theorem 5.** *There exists a polynomial time algorithm such that for every $\epsilon > 0$, given an instance of $SRPT(n, k, 1/2)$, the algorithm outputs a n^{18} size list of graphs that contains a subgraph W of G such that $|S \setminus W| = O(\epsilon k)$ and $|W \setminus S| = o(k)$ whenever $k \geq \Omega\left(\frac{1}{\epsilon^{1/2}}\sqrt{n \log n}\right)$ with probability 0.99 over the instance of $SRPT(n, k, 1/2)$.*

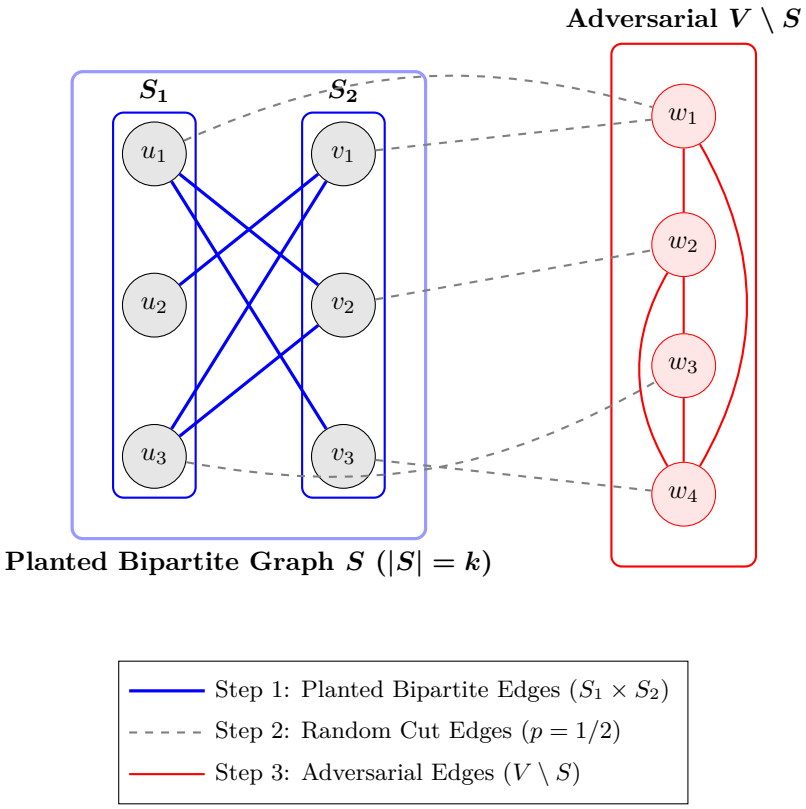


Figure 1 Structure of the Semirandom Planted Bipartite Graph, SRPB($n, k, 1/2, 1$).

1.1 Other Related Work

Random and Semirandom Models. [13] proposed a model in which a d-regular bipartite graph is planted on a set S of size k in G and rest of the edges are added independently with probability p, they designed an algorithm which recovers the planted bipartite subgraph with high probability when $k = \Omega_p(\sqrt{n \log n})$. Their algorithm also works for monotone adversary i.e. their algorithm can also recover the planted bipartite subgraph even if arbitrary edges are added in $G[V \setminus S]$ with addition to the edges that are already randomly generated. They show that the natural SDP formulation of this problem is integral. [14] studied this problem without monotone adversary and gave an simple spectral algorithm to recover when $k = \Omega(\sqrt{n})$.

[9] also study a natural extension of their semirandom model for 2-coloring (defined earlier in this section) to 3-coloring. Assuming that there is an algorithm to color 3-colorable graphs using n^θ colors, they gave an algorithm to compute a set of size $(1-\epsilon f(p,\theta))n$ (for some function f) and an n^θ coloring of it, where $(1-\epsilon)n$ is the size of the planted 3-colorable subgraph.

Vertex Deletion Models. In [1] a polynomial time algorithm was designed for the following problem. Given a graph $G=(V,E)$ where there exists a $(1-\epsilon)|V|$ size subset U of V such that the induced subgraph on U is bipartite, produce a subset U' of V of size at least $(1-O(\epsilon\sqrt{\log n}))|V|$ such that the subgraph induced on U' is bipartite. In [7], for the case where the maximum degree of G is d, they gave an algorithm to produce a subset U' of size at least $(1-O(\sqrt{\epsilon \log d}))|V|$ such that the subgraph induced on U' is bipartite; assuming the Unique Games conjecture, they prove a matching (up to constant factors) hardness. [8] studied the problem when the bipartite graph induced on U is a regular and low threshold-rank graph; they gave an algorithm, running in time exponential in $poly(1/\epsilon)$, to recover a $(1-\epsilon^{1/12})n$ sized subset of $U' \subset V$ such that the graph induced on U' is bipartite. Given a graph $G=(V,E)$ in which there exists a subset U of size $(1-\epsilon)|V|$ such that the graph induced on U is 3-colorable. [9] designed a polynomial time algorithm to produce a subset $S \subset V$ such that $|S| \leq 3\sqrt{\epsilon}|V|$ and color $V \setminus S$ using $\tilde{O}(n^{0.25+O(\epsilon^{0.25})})$. They gave an algorithm to give a more general result.

Edge Deletion Models. In [1], the problem of Min UnCut was studied which is equivalent to the problem of removing a minimum set of edges M from a graph $G=(V,E)$ such that the graph $G'=(V,E\setminus M)$ is bipartite. [1] designed a $O(\sqrt{\log n})$ approximation algorithm based on semi definite programming. [10] gave an algorithm to delete $O(\sqrt{OPT})$ fraction of edges to obtain a bipartite graph where OPT is the optimal fraction of edges in Min UnCut.

2 Proof Overview

We are going to follow the strategy introduced in [3] and will try to extend it to the case of planted bipartite subgraphs.

Let G denote the ± 1 adjacency matrix of the graph $G=(V,E)$ with $V=[n]$, i.e., $G(i,j)=1$ iff $\{i,j\}$ is an edge in G. We also let G_i be the *i-th* row of G, $G_i^{\{j\}} \in \{\pm 1\}^{|S_j|}$ be the projection of G_i to coordinates in the independent components S_1, S_2 of the planted bipartite subgraph S, $G_i^{\text{out}} \in \{\pm 1\}^{n-k}$ be the projection of G_i to the coordinates in $V \setminus S$ and $G_i^S \in \{\pm 1\}^k$ be the projection of G_i to the coordinates in S. N_i denotes the neighbors of vertex i in S. For two sets A and B, $A \Delta B$ denote their symmetric difference. For two vectors $u, v \in \mathbb{R}^m$, $\langle u, v\rangle$ denotes their Euclidean inner product.

Strong correlation between vectors present in same color class of S

We observe that i,j are non-trivially correlated if i,j are both in S_1 (or S_2).

$$\langle G_i, G_j\rangle = \langle G_i^S, G_j^S\rangle + \langle G_i^{\text{out}}, G_j^{\text{out}}\rangle = \langle G_i^{\{1\}}, G_j^{\{1\}}\rangle + \langle G_i^{\{2\}}, G_j^{\{2\}}\rangle + \langle G_i^{\text{out}}, G_j^{\text{out}}\rangle.$$

Since if $i,j \in S_1$, the first term is $|S_1|$ and the second term depends on the number of common neighbors of i and j in S_2 i.e.

$$\left\langle G_i^{\{2\}}, G_j^{\{2\}}\right\rangle = |S_2| - 2|N_i \Delta N_j|.$$

The third term is the sum of $n-k$ $\{\pm 1\}$ terms sampled uniformly and independently at random because every term in $cut(S)$ is chosen independently to be in G. Therefore the third term satisfies $|\langle G_i^{\text{out}}, G_j^{\text{out}}\rangle| \leq O(\sqrt{n\log n})$ with high probability. Thus

$$\langle G_i, G_j\rangle = |S_1| + |S_2| - 2|N_i \Delta N_j| \pm O(\sqrt{n\log n}) = k - 2|N_i\Delta N_j| \pm O(\sqrt{n\log n}).$$

Thus $i,j \in S_1$ will be highly correlated if $k \gg O(\sqrt{n\log n})$ and $|N_i\Delta N_j|$ is smaller than some constant fraction of k. In case of the semirandom planted clique problem [3], we directly get that $\langle G_i, G_j\rangle = k \pm \sqrt{n\log n}$ for $i,j \in S$ because S is a clique. Using the following Lemma 6, which uses the principle of inclusion-exclusion, we will show that many of the pairs $i,j \in S_1$ will be highly correlated. We need to carefully track the arbitrary edges between S_1 and S_2; we do this in Lemma 6 by showing that for three vertices $i,j,w, \in S_1$, all three pairs cannot simultaneously have large symmetric difference of their neighborhood in S.

▶ **Lemma 6.** *For all $i,j,w \in S_1$ (or S_2) if $\langle G_i^S, G_j^S\rangle < \frac{k}{12/(3-t)}$ and $\langle G_i^S, G_w^S\rangle < \frac{k}{12/(3-t)}$, then $\langle G_j^S, G_w^S\rangle \geq \frac{k}{12/(3-t)}$.*

Weak correlation between vectors when $i \in S$ and $j \in V \setminus S$

If $i \in S$ and $j \in V\setminus S$, then $\langle G_i^{\{1\}}, G_j^{\{1\}}\rangle + \langle G_i^{\{2\}}, G_j^{\{2\}}\}\rangle$ will be the sum of k $\{\pm 1\}$ terms sampled uniformly and independently at random whose sum will be $O(\sqrt{k\log k})$ with high probability. Thus i,j will be highly correlated when $\langle G_i^{\text{out}}, G_j^{\text{out}}\rangle$ is sufficiently large. But using Lemma 10 we will show that most of such pairs cannot be highly correlated with high probability; one such easy case to check this is to sample a $\mathcal{G}(n-k,1/2)$ graph on $V\setminus S$ vertices.

In [3] it was shown that for most of the vertices in $S(99\ \%)$ only a small amount $o(k)$ of the vertices of $V\setminus S$ can form a large inner product with a vertex in S, when $k \gg n^{3/4}$. For a vertex $i \in V$, define $V_i = \{j \in V | \langle G_i, G_j\rangle \geq \frac{k}{18/(3-t)}\}$. We show in Lemma 7 using a subtle counting argument that there exist four vertices $i,j,w,l \in S$, using which we recover most of S, i.e. $|(V_i \cup V_j \cup V_w \cup V_l)\Delta S| = O(\epsilon|S|)$.

▶ **Lemma 7.** *For $k = \Omega\left(\sqrt{\frac{24}{\epsilon^{1/2}(3-t)^4}} n^{3/4}\log n\right)$ with probability at least $1-1/n$ over the input of SRPB$(n,k,1/2,t)$ there exist 4 vertices i,j,w,l of S such that $|(V_i \cup V_j \cup V_w \cup V_l)\Delta S| = O(\epsilon k)$.*

To make this work for smaller k, [3] used the Hadamard product of the G_i vectors instead of working with G_i vectors: For $\alpha \subseteq V, |\alpha| = 3$ define G_α as point wise product (Hadamard product) of the vectors G_i, G_j, G_w where $\alpha = \{i,j,w\}$. We note that using the above argument that if $\alpha \subseteq S_1$ and $l \in S$ then

$$\langle G_\alpha, G_l\rangle = k - 2|N_\alpha \Delta N_l| \pm O(\sqrt{n\log n})$$

where N_α corresponds to a neighborhood of a natural imaginary vertex formed by α in S_1 which will be explained later. Similarly if $l \in V\setminus S$ then

$$\langle G_\alpha^{S_1}, G_l^{S_1}\rangle + \langle G_\alpha^{S_2}, G_l^{S_2}\rangle = O(\sqrt{k\log k}).$$

The analogue of the previous two equations for the semirandom planted clique problem was shown in [3]. In [3, 11] it was also shown that for most of the 3 size subset of $S(99\ \%)$, at most $o(k)$ vertices of $V\setminus S$ satisfies $\langle G_\alpha^{\text{out}}, G_l^{\text{out}}\rangle \geq k/3$ they showed this using restricted isometric property (RIP) of random matrices. To recover the planted bipartite subgraph when $k \gg \sqrt{n\log n}$, we use this strategy to prove results analogous to Lemma 6 and Lemma 7 (see Lemma 12).

3 Algorithm and Analysis

3.1 Semirandom Planted Bipartite Subgraph for size $k = \Omega(n^{3/4})$

In this section we prove Theorem 2 using Algorithm 1.

▶ **Theorem 2.** *There exists a polynomial time algorithm such that for every $\epsilon > 0$ and $t \in [1,3)$, given an instance of SRPB$(n,k,1/2,t)$, the algorithm outputs a n^4 size list of graphs which contains a subgraph W of G such that $|S \setminus W| = O(\epsilon k)$ and $|W \setminus S| = o(k)$ whenever $k \geq \Omega\left(\frac{1}{\sqrt{\epsilon^{1/2}(3-t)^4}} n^{3/4} \log n\right)$ with probability $1-1/n$ over the instance of SRPB$(n,k,1/2,t)$.*

Algorithm 1 Greedy algorithm for a n^4 sized list containing the planted bipartite graph.

1: **for** $i = 1$ to n **do**
2: Collect all the vertices j such that $\langle G_i, G_j \rangle \geq \frac{k}{18/(3-t)}$, call this set V_i.
3: **end for**
4: **for** $i,j,w,l = 1$ to n **do**
5: Output the subgraph induced by the vertex set $V_i \cup V_j \cup V_w \cup V_l$.
6: **end for**

First we show that no three vertices $i,j,w \in S_1$ (or S_2) can be simultaneously uncorrelated with each other.

▶ **Lemma 6.** *For all $i,j,w \in S_1$ (or S_2) if $\langle G_i^S, G_j^S \rangle < \frac{k}{12/(3-t)}$ and $\langle G_i^S, G_w^S \rangle < \frac{k}{12/(3-t)}$, then $\langle G_j^S, G_w^S \rangle \geq \frac{k}{12/(3-t)}$.*

Proof. Assume $i,j,w \in S_1$ and $\langle G_i^S, G_j^S \rangle, \langle G_j^S, G_w^S \rangle, \langle G_i^S, G_w^S \rangle < \frac{k}{12/(3-t)}$. Then

$$\langle G_i^S, G_j^S \rangle = k - 2|N_i \Delta N_j| < k/\frac{12}{3-t} \implies |N_i \Delta N_j| > \frac{k}{2}\left(1 - \frac{3-t}{12}\right).$$

Similarly the same will be also true for $|N_i \Delta N_w|$ and $|N_j \Delta N_w|$. By the formula of inclusion-exclusion, we have

$$\begin{aligned} 2|N_i \cup N_j \cup N_w| &= 2|N_i| + 2|N_j| + 2|N_w| - 2|N_i \cap N_j| - 2|N_i \cap N_w| - 2|N_w \cap N_j| \\ &\quad + 2|N_i \cap N_j \cap N_w| \\ &= |N_i \Delta N_j| + |N_i \Delta N_w| + |N_j \Delta N_w| + 2|N_i \cap N_j \cap N_w| \end{aligned}$$

Since $N_i \cup N_j \cup N_w \subseteq S_2$ and $|S_2| \leq tk/(1+t)$, we have

$$\begin{aligned} 2tk/(1+t) \geq 2|N_i \cup N_j \cup N_w| &> \frac{3k}{2}\left(1 - \frac{3-t}{12}\right) + 2|N_i \cap N_j \cap N_w| \\ \implies |N_i \cap N_j \cap N_w| &< -\frac{(3-t)^2}{16(1+t)} < 0 \text{ if } t < 3, \end{aligned}$$

which gives us a contradiction. Similarly if $i,j,k \in S_2$ then $N_i \cup N_j \cup N_w \subset S_1$ and $|S_1| \leq k/2$. Therefore

$$2|N_i \cap N_j \cap N_w| < k - \frac{3k}{2}\left(1 - \frac{3-t}{12}\right) = \frac{-k}{8}(1+t) < 0. \qquad \blacktriangleleft$$

▶ Remark 8. Note that we can prove a much tighter bound if $i,j,w \in S_2$, since $|S_2| \geq |S_1|$. By following the same argument as above we get that for all $i,j,w \in S_2$ if $\langle G_i^S, G_j^S \rangle < \frac{k}{3}$ and $\langle G_i^S, G_w^S \rangle < \frac{k}{3}$, then $\langle G_j^S, G_w^S \rangle \geq \frac{k}{3}$.

Call a pair of vertices $i, j \in S$ good if $\langle G_i^S, G_j^S \rangle \geq \frac{k}{12/(3-t)}$, otherwise call them bad. Lemma 9 shows that all but one vertex of S_1 forms a good pair with some other vertex of S_1.

▶ **Lemma 9.** *At least $|S_1| - 1$ vertices of S_1 forms a good pair with some other vertex in S_1; the same is true for the vertices of S_2.*

Proof. To see this define

$$T = \{i \in S_1 \mid i \text{ forms a good pair with some other vertex in } S_1\}$$

if $S_1 \setminus T$ is nonempty then choose a vertex i from it. Since $i \notin T$ it means i forms bad pairs with all other vertices in S_1. Then by Lemma 6 we can conclude that all the vertices in $S_1 \setminus \{i\}$ forms good pairs in each other, so $|T| \geq |S_1| - 1$. Similar argument can be applied to S_2. ◀

The following lemma is taken directly from [3] which will imply that a vertex from S cannot be highly correlated with many vertices of $V \setminus S$. We reproduce their proof here to make dependence on t explicit since this case doesn't arise in their setting.

▶ **Lemma 10** ([3])**.** *Let $v_1, v_2, v_3, ..., v_k \in \{\pm 1\}^n$ be vectors sampled uniformly and independently at random for $k = \Omega(\sqrt{n \log n})$ Then with probability at least $1 - 1/n$ over the draw of vectors v_i, for every $u \in \{\pm 1\}^n$ there are at most $O\left(\frac{24^2}{(3-t)^2} n^2/k^2\right)$ vectors v_i such that $\langle u, v_i \rangle \geq \frac{k}{24/(3-t)}$.*

Proof. Consider the $n \times k$ matrix H with columns $v_1, v_2, ..., v_k$. Then, by standard results in random matrix theory, $||H||_2 \leq O(\sqrt{n} + \sqrt{k})$ with probability $1 - 1/n$. Let $\mathcal{B} \subset [k]$ be such that, for every $i \in \mathcal{B}, \langle u, v_i \rangle \geq \frac{k}{24/(3-t)}$. Then, by the Cauchy-Schwartz inequality, we have

$$\langle u, H1_{\mathcal{B}} \rangle \leq ||u||_2 ||H1_{\mathcal{B}}||_2 \leq \sqrt{n}\sqrt{|\mathcal{B}|}||H||_2 \leq O(n\sqrt{|\mathcal{B}|}).$$

On the other hand, by the choice of set $\mathcal{B}$, we have

$$\langle u, H1_{\mathcal{B}} \rangle = \sum_{i \in \mathcal{B}} \langle u, v_i \rangle \geq |\mathcal{B}| \frac{k}{24/(3-t)}.$$

Combining those two inequalities and rearranging, we get $|\mathcal{B}| \leq O\left(\frac{24^2}{(3-t)^2} n^2/k^2\right)$. ◀

Greedy Procedure. For $i \in V$, define $V_i = \{j \in V | \langle G_i, G_j \rangle \geq \frac{k}{18/(3-t)}\}$. We show that we can reconstruct a bipartite graph which is close to the planted bipartite subgraph using $i, j, w, l \in S$.

We apply the Lemma 10 with G_i^{out} for each $i \in S$ as a uniform random element of $\{\pm 1\}^{n-k}$ and by taking $u = G_j^{\text{out}}$ for any $j \in V \setminus S$. By Lemma 10, for each $j \in V \setminus S$, at most $O\left(\frac{24^2}{(3-t)^2} n^2/k^2\right)$ elements of i of S will satisfy $\langle G_i^{\text{out}}, G_j^{\text{out}} \rangle \geq \frac{k}{24/(3-t)}$. Using this and the fact that for all $i \in S$ and $j \in V \setminus S$, $\langle G_i^{\text{S}}, G_j^{\text{S}} \rangle = O(\sqrt{k \log k})$ with probability $1 - 1/n$, we get that there are in total $O\left(\frac{24^2}{(3-t)^2} n^3/k^2\right)$ pairs (i, j) such that $i \in S, j \notin S$ satisfying

$$\langle G_i, G_j \rangle \geq \frac{k}{18/(3-t)} \geq \frac{k}{24/(3-t)} + O(\sqrt{k \log k}).$$

By Markov's inequality, a uniformly random $i \in S$ satisfies $\langle G_i, G_j \rangle \geq \frac{k}{18/(3-t)}$ for at most $O\left(\frac{24^2}{\epsilon(3-t)^2} n^3/k^3\right)$ vertices $j \in V \setminus S$ with probability $1 - \epsilon$. If $k \gg O\left(\frac{24^2}{\epsilon(3-t)^2} n^3/k^3\right)$ e.g.

$k = \Omega\left(\sqrt{\frac{24}{\epsilon^{1/2}(3-t)}} n^{3/4}\gamma(n)\right)$ such that $\lim_{n\to\infty}\gamma(n) = \infty$, then at least $k(1-\epsilon)$ vertices i of S will satisfy $|V_i \setminus S| = O\left(\frac{k}{\gamma^4(n)}\right) = o(k)$, which means V_i will contain at most $o(k)$ vertices not from S, call such a set of vertices i of S as S^*. Therefore

$$|S \setminus S^*| \leq \epsilon k. \tag{1}$$

Note that to make the contribution from random terms small we need $\frac{k}{18/3-t} - \frac{k}{24/3-t} > \Omega(\sqrt{k\log k})$. Therefore we also need $k = \Omega(\log k/(3-t)^2)$.

In [3] because S was a clique, the above argument implies that for a vertex $i \in S^*$, V_i contains the whole of S and $o(k)$ vertices from $V \setminus S$. To see this, observe that each vertex $j \in S$ will satisfy $\langle G_i, G_j\rangle = k \pm O(\sqrt{n\log n}) \geq k/6$ and all but $o(k)$ vertices of $V \setminus S$ will not be in V_i because $i \in S^*$. Our goal is to extend this idea to recover bipartite subgraphs. We show that in SRPB$(n, k, 1/2, t)$ we can recover $1 - O(\epsilon)$ fraction of vertices of S using at most 4 vertices of S.

▶ **Lemma 7.** *For $k = \Omega\left(\sqrt{\frac{24}{\epsilon^{1/2}(3-t)^4}} n^{3/4}\log n\right)$ with probability at least $1-1/n$ over the input of SRPB$(n, k, 1/2, t)$ there exist 4 vertices i, j, w, l of S such that $|(V_i \cup V_j \cup V_w \cup V_l)\Delta S| = O(\epsilon k)$.*

Proof. Let $i \in S_1 \cap S^*$. Intuitively Lemma 6 gives us that if V_i doesn't contain all vertices of S_1 i.e. if $S_1 \setminus V_i \neq \phi$, then any two vertices $j, w \in S_1 \setminus V_i$ will satisfy $\langle G_i^S, G_w^S\rangle \geq k/\frac{12}{3-t}$ so $V_j \supset S_1 \setminus V_i$ i.e. $V_i \cup V_j \supset S_1$. But we also need j to be in S^* for $|(V_i \cup V_j) \setminus S| = o(k)$ which brings us to two cases.

Case 1. Assume that $(S_1 \setminus V_i) \setminus (S \setminus S^*) \neq \phi$. Then by Lemma 6, any vertex $j \in (S_1 \setminus V_i) \setminus (S \setminus S^*)$ (i.e. $j \notin V_i$ and $j \in S_1 \cap S^*$) will satisfy $V_j \supset S_1 \setminus V_i$ because for any other vertex $w \in S_1 \setminus V_i$, it must be that $\langle G_i, G_w\rangle = \langle G_i^S, G_w^S\rangle + \langle G_i^{\text{out}}, G_w^{\text{out}}\rangle < \frac{k}{18/(3-t)}$, and since $\langle G_i^{\text{out}}, G_w^{\text{out}}\rangle = O(\sqrt{n\log n}) \ll \frac{k}{36/(3-t)}$ with very high probability, it implies that $\langle G_i^S, G_w^S\rangle < \frac{k}{12/(3-t)}$. This implies that by Lemma 6 $\langle G_j^S, G_w^S\rangle \geq \frac{k}{12/(3-t)}$ which means $\langle G_j, G_w\rangle \geq \frac{k}{18/(3-t)}$ for any $w \in S_1 \setminus V_i$, so $V_i \cup V_j \supset S_1$. Also since $j \in S_1 \cap S^*$ we have $|(V_1 \cup V_j) \setminus S| \leq |V_i \setminus S| + |V_j \setminus S| = o(k)$.

Case 2. $(S_1 \setminus V_i) \setminus (S \setminus S^*) = \phi$, then V_i has almost all the vertices of S_1 because $S_1 \setminus V_i \subset S \setminus S^*$, so $|S_1 \setminus V_i| \leq |S \setminus S^*| = O(\epsilon k)$ and since $i \in S_1 \cap S^*$ we have $|V_1 \setminus S| = o(k)$.

Therefore either by case 1 or case 2 there exists either one or two vertices of S_1 such that $|S_1 \setminus (V_i \cup V_j)| = O(\epsilon k)$ and $|(V_i \cup V_j) \setminus S| = o(k)$. Similarly we can find two vertices $w, l \in S_2$ which satisfies $|S_2 \setminus (V_w \cup V_l)| = O(\epsilon k)$ and $(|V_w \cup V_l) \setminus S| = o(k)$, which shows that there exists at most four vertices such that $|(V_i \cup V_j \cup V_w \cup V_l)\Delta S| = O(\epsilon k)$. ◀

From the above four lemmas we can finish the proof of Theorem 2.

Proof of Theorem 2. Algorithm 1 goes through all the four set of vertices and take their union. Then by Lemma 7 one of them will be close to the planted bipartite subgraph. Since Algorithm 1 enumerates over all choice of i, the size of list is at most n^4. ◀

3.2 Semirandom Planted Bipartite Subgraph for size $k = \Omega(\sqrt{n\log n})$

In this subsection we prove Theorem 3.

▶ **Theorem 3.** *There exists a polynomial time algorithm such that for every $\epsilon > 0$ and $t \in [1,3)$, given an instance of SRPB$(n,k,1/2,t)$, the algorithm outputs a n^{12} size list of graphs which contains a subgraph W of G such that $|S \setminus W| = O(\epsilon k)$ and $|W \setminus S| = o(k)$ whenever $k \geq \Omega\left(\frac{1}{\epsilon^{1/2}(3-t)^2}\sqrt{n \log n}\right)$ with probability 0.99 over then instance of SRPB$(n,k,1/2,t)$.*

Similar to [3, 11], we will show that we can recover S using a constant no. of vertices in S. For $\alpha \subset V$, define $G_\alpha \in \{\pm 1\}^n$ as $G_\alpha(i) = \Pi_{j\in\alpha} G_j(i)$ i.e. G_α is a Hadamard product of all the vectors G_i such that $i \in \alpha$. For $\alpha = \{i,j,w\}$, we will interchange between the notation G_α and G_{ijw}. For $\alpha \subset V$ define $V_\alpha = \left\{ i \in V \mid \langle G_\alpha, G_i \rangle \geq \frac{k}{18/(3-t)} \right\}$.

Algorithm 2 Greedy algorithm for a n^{12} sized list containing the planted bipartite graph.

for $i,j,w = 1$ to n **do**
  Collect all the vertices t such that $\langle G_{ijw}, G_t \rangle \geq \frac{k}{18/(3-t)}$, call that set V_{ijw}
end for
for $\alpha_i, \alpha_j, \alpha_w, \alpha_l = 1$ to $\binom{n}{3}$ **do** ▷ Go through all 3-size subsets of V
  Output the subgraph induced by the vertex set $V_{\alpha_i} \cup V_{\alpha_j} \cup V_{\alpha_w} \cup V_{\alpha_l}$.
end for

A small observation tells us that if $\alpha_1 = \{i,j,l\}$ and $\alpha_2 = \{l,w,t\}$ then

$$\langle G_{\alpha_1}, G_{\alpha_2} \rangle = \langle G_{ijw}, G_t \rangle. \tag{2}$$

The next lemma uses the 1-norm analogue of RIP of matrices. For reference see proposition 2 and section 4 of [11].

▶ **Lemma 11** ([11]). *For every $i \in V \setminus S$, at most $O\left(\frac{24^2}{(3-t)^2} n^2/k^2\right)$ subsets $\alpha \subset S$ of size 3 satisfy $\langle G_\alpha^{\text{out}}, G_i^{\text{out}} \rangle \geq \frac{k}{24/(3-t)}$ with probability 0.99 whenever $k \geq \Omega(\frac{1}{3-t}\sqrt{n \log n})$.*

Thus in total $O\left(\frac{24^2}{(3-t)^2} n^3/k^2\right)$ pairs (α, j) such that $\alpha \in \binom{S}{3}, j \notin S$ satisfying $\langle G_\alpha, G_j \rangle \geq \frac{k}{18/(3-t)} \geq \frac{k}{24/(3-t)} + O(\sqrt{k \log k})$, by Markov's inequality, a uniformly random $\alpha \in \binom{S}{3}$ satisfies $\langle G_\alpha, G_j \rangle \geq \frac{k}{18/(3-t)}$ for at most $O\left(\frac{24^2}{\epsilon^3(3-t)^2} n^3/k^5\right)$ with probability $1-\epsilon^3$ and if $k \gg O\left(\frac{24^2}{\epsilon^3(3-t)^2} n^3/k^5\right)$ and $k = \Omega\left(\frac{1}{3-t}\sqrt{n \log n}\right)$ e.g. $\left(k = \Omega\left(\frac{24^{1/3}}{\epsilon^{1/2}(3-t)}\sqrt{n \log n}\right)\right.$ then for a uniformly chosen $\alpha \in \binom{S}{3}$ will satisfy $|V_\alpha \setminus S| = o(k)$, i.e. $(1-\epsilon^3)$ fractions elements of $\binom{S}{3}$ will form the vertex set V_α which contains at most $o(k)$ vertices not from S, call such a set of $(1-\epsilon^3)\binom{k}{3}$ elements of $\binom{S}{3}$ as T^*. Therefore

$$|T/T^*| \leq O(\epsilon^3 k^3). \tag{3}$$

In [3, 11] because S was a clique, the above argument implies that for a triplet $\alpha \in T^*$, V_α contains whole of S and $o(k)$ vertices from $V \setminus S$. To see this, observe that each vertex $j \in S$ will satisfy $\langle G_\alpha, G_j \rangle = k \pm O(\sqrt{n \log n}) \geq k/6$ and because $\alpha \in T^*$, all but $o(k)$ vertices of $V \setminus S$ will not be in V_α. Our goal is to extend this idea to recover bipartite subgraphs. We show that in SRPB$(n,k,t,1/2)$ we can recover $1 - O(\epsilon)$ fraction of vertices of S using at most 12 vertices of S.

▶ **Lemma 12.** *Whenever $k = \Omega\left(\frac{24^{1/3}}{\epsilon^{1/2}(3-t)^2}\sqrt{n \log n}\right)$, with probability 0.99 there exists $\alpha_1, \alpha_2 \in \binom{S_1}{3} \cap T^*$ such that $|S_1 \setminus (V_{\alpha_1} \cup V_{\alpha_2})| \leq O(\epsilon k)$ and $|(V_{\alpha_1} \cup V_{\alpha_2}) \setminus S| \leq o(k)$, $\alpha_3, \alpha_4 \in \binom{S_2}{3} \cap T^*$ such that $|S_2 \setminus (V_{\alpha_3} \cup V_{\alpha_4})| \leq O(\epsilon k)$ and $|(V_{\alpha_3} \cup V_{\alpha_4}) \setminus S| \leq o(k)$, which implies $|S \Delta (V_{\alpha_1} \cup V_{\alpha_2} \cup V_{\alpha_3} \cup V_{\alpha_4})| \leq O(\epsilon k)$.*

Proof.

Case 1. There exists $\alpha_1, \alpha_2 \in T^* \cap \binom{S_1}{3}$ such that $S_1 \subset V_{\alpha_1} \cup V_{\alpha_2}$, which means that $S_1 \setminus (V_{\alpha_1} \cup V_{\alpha_2}) = \phi$. Since $V_{\alpha_1}, V_{\alpha_2} \in T^* \cap \binom{S_1}{3}$, it implies that $|V_{\alpha_1} \setminus S|, |V_{\alpha_2} \setminus S| = o(k)$. Therefore $|(V_{\alpha_1} \cup V_{\alpha_2}) \setminus S| \leq o(k)$.

Case 2. For all pairs of $\alpha_1, \alpha_2 \in T^* \cap \binom{S_1}{3}$, there exists a vertex $i_{\alpha_1,\alpha_2} \in S_1$ such that $\langle G_{\alpha_1}, G_{i_{\alpha_1,\alpha_2}} \rangle < \frac{k}{18/(3-t)}$ and $\langle G_{\alpha_2}, G_{i_{\alpha_1,\alpha_2}} \rangle < \frac{k}{18/(3-t)}$. Note that this implies that for α_1, $\langle G^S_{\alpha_1}, G^S_{i_{\alpha_1,\alpha_2}} \rangle < \frac{k}{12/(3-t)}$ because $\langle G^{\text{out}}_{\alpha_1}, G^{\text{out}}_{i_{\alpha_1,\alpha_2}} \rangle = O(\sqrt{n \log n}) \ll \frac{k}{18/(3-t)}$, and the same is true for α_2 .

We will now show that $\langle G^S_{\alpha_1}, G^S_{\alpha_2} \rangle \geq \frac{k}{12/(3-t)}$. Observe that $G_{\alpha_1}(i) = -1$ whenever $i \in S_1$, and the same is true for α_2. Consider a new instance of bipartite graph T consisting of color classes W_1, W_2 of size $|S_1|, |S_2|$ respectively such that W_1 consists of three vertices a, b, c whose neighbors in W_2 are determined by the vectors $G^{S_2}_{\alpha_1}, G^{S_2}_{\alpha_2}, G^{S_2}_{i_{\alpha_1,\alpha_2}}$ respectively, i.e. a is connected to a vertex $p \in W_2$ if and only if $G_{\alpha_1}(p) = 1$ and rest of the vertices of W_1 have no neighbors in W_2. Recall that in our notation T_a is the a^{th} row of the signed adjacency matrix of T(similarly for b and c). By the construction of T we get that

$$\langle T_a, T_c \rangle = \langle G^S_{\alpha_1}, G^S_{i_{\alpha_1,\alpha_2}} \rangle < \frac{k}{12/(3-t)}, \ \langle T_b, T_c \rangle = \langle G^S_{\alpha_2}, G^S_{i_{\alpha_1,\alpha_2}} \rangle < \frac{k}{12/(3-t)}, \text{ and}$$
$$\langle T_a, T_b \rangle = \langle G^S_{\alpha_1}, G^S_{\alpha_2} \rangle$$

Then by Lemma 6 $\langle T_a, T_b \rangle \geq \frac{k}{12/(3-t)}$, therefore for all $\alpha_1, \alpha_2 \in T^*$

$$\langle G^S_{\alpha_1}, G^S_{\alpha_2} \rangle \geq \frac{k}{12/(3-t)}. \tag{4}$$

For $\beta \in \binom{S_1}{2}$, let b_β be the number of vertices $i \in S_1$ such that $\beta \cup \{i\} \in \binom{S_1}{3} \setminus T^*$. Then

$$\frac{1}{3} \sum_{\beta \in \binom{S_1}{2}} b_\beta \leq |\binom{S_1}{3} \setminus T^*| \leq O(\epsilon^3 k^3).$$

Then by Markov's inequality, $1 - \epsilon$ fraction elements β of $\binom{S_1}{2}$ will have

$$b_\beta \leq O(\epsilon^2 k). \tag{5}$$

For $i \in S_1$, let b_i be the number of elements of $\binom{S_1}{2}$ such that $\beta \cup i \in \binom{S_1}{3} \setminus T^*$, then by similar argument as above, $1 - \epsilon$ fraction of b_i will satisfy

$$b_i \leq O(\epsilon^2 k^2). \tag{6}$$

Choose an $m \in S_1$ for which $b_m \leq O(\epsilon^2 k^2)$, now for $t \in S_1$ call c_t to be the number of elements t' such that $\{m, t, t'\} \in \binom{S_1}{3} \setminus T^*$. Next we observe that $b_m = \frac{1}{2} \sum_{t \in S_1} c_t$, therefore $\sum_{t \in S_1} c_t \leq O(\epsilon^2 k^2)$. This implies that $1 - \epsilon$ fraction elements of S_1 satisfy

$$c_t \leq O(\epsilon k). \tag{7}$$

Now choose $\beta = \{i, j\} \subset S_1$ such that $\{m\} \cup \beta \in T^*$ and $b_\beta \leq O(\epsilon^2 k)$ (such a β exists because number of pairs $\beta \in \binom{S_1}{2}$ that satisfies $\{m\} \cup \beta \in T^*$ is $\binom{|S_1|-1}{2} - b_m \approx \frac{|S_1|^2}{2}(1 - O(\epsilon^2))$ (Equation 6), and most 2 size subsets β of S_1, i.e. $\binom{|S_1|}{2}(1 - \epsilon)$ elements of S_1 satisfy

$b_\beta \leq O(\epsilon^2 k)$ (Equation 5). Therefore the number of sets $\beta \in \binom{S_1}{2}$ that satisfies $\beta \cup \{m\} \in T^*$ are $\approx \frac{|S_1|^2}{2}(1-\epsilon-O(\epsilon^2)))$. Now, choose a $l \in S_1$ for which $c_l \leq O(\epsilon k)$ and $\{l\} \cup \beta \in T^*$ (such a l exist because at least $(1-\epsilon)|S_1|$ elements $l \in S_1$ satisfy $c_l \leq O(\epsilon k)$ (Equation 7), and $(1-O(\epsilon^2))|S_1|$ elements $l \in S_1$ satisfy $\beta \cup \{l\} \in T^*$ (Equation 5), so number of elements $l \in S_1$ satisfying both of the conditions simultaneously are $(1-\epsilon-O(\epsilon^2))|S_1|)$.

Since $|c_l| \leq O(\epsilon k)$, for at least $(1-O(\epsilon))|S_1|$ elements $x \in S_1$ we have $\{m,l,x\} \in T^* \cap \binom{S_1}{3}$. Moreover $\beta \cup \{l\} = \{i,j,l\} \in T^* \cap \binom{S_1}{3}$. Therefore by Equation 2 and Equation 4

$$\langle G^S_{i,j,m}, G^S_x \rangle = \langle G^S_{i,j,l}, G^S_{m,l,x} \rangle \geq \frac{k}{12/(3-t)}.$$

This implies that for at least $(1-O(\epsilon))|S_1|$ vertices x of S_1,

$$\langle G_{i,j,m}, G_x \rangle = \langle G^S_{i,j,m}, G^S_x \rangle + \langle G^{\text{out}}_{i,j,m}, G^{\text{out}}_x \rangle \geq \frac{k}{12/(3-t)} - O(\sqrt{n \log n}) \geq \frac{k}{18/(3-t)}$$

and therefore $x \in V_{ijm}$. This shows that $|S_1 \setminus V_{ijm}| \leq O(\epsilon k)$, and since $\{i,j,m\} \in T^*$ we have $V_{ijm} \setminus S = o(k)$. Therefore choosing $\alpha_1 = \{i,j,m\}$ we get the desired result.

Similarly there exist $\alpha_3, \alpha_4 \in \binom{S_2}{3} \cap T^*$ satisfying the desired property. Therefore there exist $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in T^*$ such that $|S \Delta (V_{\alpha_1} \cup V_{\alpha_2} \cup V_{\alpha_3} \cup V_{\alpha_4})| \leq O(\epsilon k)$. ◀

From the above two lemmas we can finish proof of Theorem 3 observation the Algorithm 2 gives a n^{12} size list of almost bipartite graphs which will surely contain a graph close to the planted bipartite subgraph on S.

Proof of Theorem 3. Using Algorithm 2 and Lemma 12 there exists $\alpha_i, \alpha_j, \alpha_w, \alpha_l \in \binom{S}{3}$ such that the subgraph induced on the set $V_{\alpha_i} \cup V_{\alpha_j} \cup V_{\alpha_w} \cup V_{\alpha_l}$ satisfies the conditions of Theorem 3. Since Algorithm 1 enumerates over all choice of α_i, the size of list is at most n^{12}. ◀

▶ Remark 13. For any $t \geq 1$, in particular for $t \geq 3$, we can still recover most of S_2 as follows. Using the arguments of Lemma 12, we can show that there exist $\alpha, \beta \in \binom{S_2}{3}$ such that $(S_2 \setminus (V_\alpha \cup V_\beta) \leq O(\epsilon k)$ where $V_\alpha = \left\{ i \in V \mid \langle G_\alpha, G_i \rangle \geq \frac{k}{6} \right\}$. Therefore Algorithm 2 can be modified to output a list of subgraphs from which at least one subgraph W will satisfy $|W \cap S| \geq (1-O(\epsilon))tk/(1+t)$ and $|W \setminus S| \leq o(k)$.

3.3 Semirandom Planted Tripartite Subgraph

Most of the work required to prove Theorem 5 is already done in the previous subsections. We use Algorithm 3 to prove Theorem 5.

Algorithm 3 Greedy algorithm for a n^{18} sized list containing the planted tripartite graph.

for $i,j,w = 1$ to n **do**
    Collect all the vertices t such that $\langle G_{ijw}, G_t \rangle \geq \frac{k}{18/(3-t)}$, call that set V_{ijw}
end for
for $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6 = 1$ to $\binom{n}{3}$ **do**
    Output the subgraph induced by the vertex set $V_{\alpha_1} \cup V_{\alpha_2} \cup V_{\alpha_3} \cup V_{\alpha_4} \cup V_{\alpha_5} \cup V_{\alpha_6}$.
end for

Proof of Theorem 5. Observe that if $i, j \in S_1$ then

$$\langle G_i^S, G_j^S \rangle = k - 2|N_i \Delta N_j|$$

where N_i denotes the set of neighbors of i in S. $\langle G_i^S, G_j^S \rangle$ doesn't depend on the edges between S_2 and S_3, this case is similar to the case of SRPB$(n, k, 1/2, 2)$. Therefore using Lemma 12, there exists six subsets of S $\alpha_1, \alpha_2 \in \binom{S_1}{3}, \alpha_3, \alpha_4 \in \binom{S_2}{3}, \alpha_5, \alpha_6 \in \binom{S_3}{3}$ which satisfies $|S_1 \setminus (V_{\alpha_1} \cup V_{\alpha_2})| = O(\epsilon k)$ and $|(V_{\alpha_1} \cup V_{\alpha_2}) \setminus S_1| = o(k)$, and similarly for $V_{\alpha_3} \cup V_{\alpha_4}$ with S_2 and $V_{\alpha_5} \cup V_{\alpha_6}$ with S_3. So the Algorithm 3 gives the desired result. ◀

4 Conclusion

In this work, we gave algorithms for recovering planted SRPB. In Theorem 2 we output a list of n^4 subgraphs in which one subgraph is guaranteed to have to have small symmetric difference with S, whereas in Theorem 3 the list size is n^{12}. A natural open question is to obtain list size of $(1 + o(1))\binom{\lfloor n/k \rfloor}{2}$. [3, 11] obtain an optimal list size of $(1 + o(1))n/k$ for the semirandom planted clique problem by using some combinatorial methods to reduce the size of their list obtained from their greedy procedure. Such techniques don't seem to directly extend to SRPB.

Our algorithms also require the planted bipartite subgraph to be close to being balanced, i.e., $t \in [1, 3)$. Proving an analogue of Theorem 2 and Theorem 3 is also an open question whenever $t \geq 3$.

References

1. Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. O($\sqrt{\log n}$) approximation algorithms for min uncut, min 2cnf deletion, and directed cut problems. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 573–581, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1060590.1060675.
2. Noga Alon, Michael Krivelevich, and Benny Sudakov. Finding a large hidden clique in a random graph. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 594–598, USA, 1998. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=314613.315014.
3. Jaroslaw Blasiok, Rares-Darius Buhai, Pravesh K. Kothari, and David Steurer. Semirandom Planted Clique and the Restricted Isometry Property . In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 959–969, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00064.
4. Rares-Darius Buhai, Pravesh K. Kothari, and David Steurer. Algorithms approaching the threshold for semi-random planted clique. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1918–1926, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585184.
5. Moses Charikar, Jacob Steinhardt, and Gregory Valiant. Learning from untrusted data. In *Proceedings of the 49th annual ACM SIGACT symposium on theory of computing*, pages 47–60, 2017. doi:10.1145/3055399.3055491.
6. Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001. doi:10.1006/jcss.2001.1773.
7. Suprovat Ghoshal and Anand Louis. Approximation algorithms and hardness for strong unique games. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '21, pages 414–433, USA, 2021. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611976465.26.

8 Suprovat Ghoshal and Anand Louis. Approximating csps with outliers. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference), September 19-21, 2022*, volume 245 of *LIPIcs*, pages 43:1–43:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.43.

9 Suprovat Ghoshal, Anand Louis, and Rahul Raychaudhury. Approximation Algorithms for Partially Colorable Graphs. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.28.

10 Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. doi:10.1145/227683.227684.

11 Venkatesan Guruswami and Hsin-Po Wang. Semirandom planted clique via 1 norm isometry property. *International Conference on Integer Programming and Combinatorial Optimization*, pages 270–282, 2025.

12 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999. doi:10.1007/BF02392825.

13 Akash Kumar, Anand Louis, and Rameesh Paul. Exact Recovery Algorithm for Planted Bipartite Graph in Semi-Random Graphs. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84:1–84:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.84.

14 Anand Louis and Rameesh Paul. No sdp needed for efficiently recovering planted regular bipartite graphs. In *2026 SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 538–550. SIAM, 2026.

15 Anand Louis, Rameesh Paul, and Prasad Raghavendra. Robust algorithms for recovering planted r-colorable graphs. In Nika Haghtalab and Ankur Moitra, editors, *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pages 3766–3794. PMLR, 30 June–04 July 2025. URL: https://proceedings.mlr.press/v291/louis25a.html.

16 Theo McKenzie, Hermish Mehta, and Luca Trevisan. A new algorithm for the robust semi-random independent set problem. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 738–746, USA, 2020. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975994.45.

17 Jacob Steinhardt. Does robustness imply tractability? A lower bound for planted clique in the semi-random model. *Electron. Colloquium Comput. Complex.*, TR17-069, 2017. URL: https://eccc.weizmann.ac.il/report/2017/069.

18 Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 253–264, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804355.

Polychromatic 2-Colorings with Bounded Discrepancy for Triangulations

Alma Arevalo Loyola
School of Computer Science, Carleton University, Ottawa, Canada

Ahmad Biniaz
School of Computer Science, University of Windsor, Canada

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada

Thomas Shermer
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

A polychromatic 2-coloring of a triangulation is a 2-coloring of the vertices such that no face is monochromatic. The discrepancy of a coloring is the maximum difference between the sizes of the color classes. Asayama and Matsumoto (Graphs and Combinatorics, 2022) proved that every triangulation admits a polychromatic 2-coloring with discrepancy at most $\frac{5n-16}{9}$, and that there exists a class of triangulations for which every polychromatic 2-coloring has discrepancy at least $\frac{n}{3}-2$, where n is the number of vertices. We improve the upper bound, showing that every triangulation admits a polychromatic 2-coloring with discrepancy at most $\frac{3n-16}{7}$ and such a 2-coloring can be computed in quadratic time. We also show a discrepancy of at most $n-\frac{4M}{3}$ for triangulations with a matching of size M. This implies, for example, that Delaunay triangulations admit a discrepancy of at most $\frac{n}{3}$. We provide a linear-time algorithm to compute a 2-coloring whose discrepancy is at most $\frac{5n-24}{7}$.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Computational geometry

Keywords and phrases polychromatic coloring, triangulation, balanced coloring, matching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.33

Funding *Ahmad Biniaz*: Supported in part by NSERC.
Prosenjit Bose: Supported in part by NSERC.
Thomas Shermer: Supported in part by NSERC.

Acknowledgements Part of this work was done at the 12th Annual Workshop on Geometry and Graphs, held at Bellairs Research Institute in Barbados in February 2025. We thank the organizers and the participants.

1 Introduction

Colorings of planar graphs play a central role in both combinatorial and algorithmic graph theory. The study of colorings that ensure coverage or visibility conditions has received growing attention in recent years, particularly in contexts where each region of a planar subdivision must contain representatives from multiple categories or sensors. For instance, such colorings arise naturally in network coverage problems, geometric guarding, and frequency assignment, where it is desirable that every bounded region has access to all available resources. This motivated the study of *polychromatic colorings*, colorings in which every face of a planar embedding contains at least one vertex of each color. For a survey on polychromatic colorings see [8].

© Alma Arevalo Loyola, Ahmad Biniaz, Prosenjit Bose, and Thomas Shermer;
licensed under Creative Commons License CC-BY 4.0

For the case of two colors, a polychromatic 2-coloring of a triangulation assigns one of two colors to each vertex such that no face is monochromatic. Unlike proper colorings, which aim to separate adjacent vertices, polychromatic colorings of planar graphs emphasize inclusion: every face must *see* both colors. Although such colorings always exist, the balance between the sizes of the two color classes is not guaranteed. The *discrepancy* of a specific polychromatic 2-coloring of G is defined as the absolute difference between the sizes of its two color classes. It measures how unevenly the two colors are distributed. For a graph G, we define $\mathrm{disc}(G)$ as the minimum discrepancy over all possible polychromatic 2-colorings of G. Thus, $\mathrm{disc}(G)$ captures the best achievable balance between the two colors while maintaining the polychromatic property. A polychromatic coloring of G is said to be *balanced* when its discrepancy is at most one.

Asayama and Matsumoto [1] initiated the systematic study of balanced polychromatic 2-colorings in planar triangulations. Among other results, they proved that every triangulation on n vertices admits a polychromatic 2-coloring with discrepancy at most $\frac{5n-16}{9}$, and they constructed infinite families of triangulations where every such coloring has discrepancy at least $\frac{n}{3} - 2$. They conjectured that $\frac{n}{3}$ is the best possible asymptotic upper bound.

In this work we make progress toward that conjecture. We show that every triangulation on n vertices admits a polychromatic 2-coloring satisfying $\mathrm{disc}(G) \leq \frac{3n-16}{7}$, which improves the previous bound of $\frac{5n-16}{9}$. Our proof refines the discharging arguments used in earlier analyses and integrates structural constraints derived from the Four Color Theorem. Moreover, the proof is constructive and leads to an $O(n^2)$ algorithm to compute such a coloring.

We further establish that if a triangulation G admits a matching of size M, then G has a polychromatic 2-coloring with discrepancy at most $\frac{n-4M}{3}$. This bound is tight for certain triangulations and immediately implies that Delaunay triangulations satisfy $\mathrm{disc}(G) \leq n/3$, since they always satisfy having a perfect matching. In addition, we provide a linear-time recoloring procedure that guarantees a polychromatic 2-coloring with discrepancy at most $\frac{5n-24}{7}$. This yields the first efficient algorithm that achieves a nontrivial upper bound on discrepancy in planar triangulations.

Our results strengthen the connection between structural graph properties and color balance in planar settings, advancing the quantitative understanding of how evenly a triangulation can be 2-colored while ensuring that every face remains non-monochromatic.

2 Preliminaries

Most of the basic notions about graph coloring can be found in [11] and [17]. For general concepts of graph theory, see [4] and [9]. We consider graphs that are finite and simple; that is, they have no loops or multiple edges. We denote a graph by $G = (V, E)$, where V is the set of vertices, and E is the set of edges. For planar graphs, we denote by F the set of faces. When needed, we denote by $V(G)$, $E(G)$ and $F(G)$ the sets of vertices, edges and faces to specify the graph to which we refer. We use n, m, and l to denote the number of vertices, edges and faces, respectively. The *degree* of a vertex $v \in V$ is the number of vertices adjacent to v, and it is denoted by $d(v)$. We denote by $\Delta(G)$ and $\delta(G)$ the maximum and minimum degree of the vertices of G, respectively. The degree of a face $f \in F$, denoted by $d(f)$, is the number of edges (or vertices) on its boundary. A *vertex coloring* χ of G is a function $\chi : V \to \{c_1, c_2, ..., c_k\}$. We call it a *$k$-coloring* to emphasize the number of colors.

▶ **Definition 1.** *Let G be a graph with n vertices, let χ be a vertex coloring of G, and let $\{V_1, V_2, \ldots, V_k\}$ be the partition induced by χ. We define the* discrepancy *of χ, denoted as* $\mathrm{disc}(\chi)$*, to be the difference between the sizes of the largest and the smallest color class.*

$$\mathrm{disc}(\chi) = \max\{|V_i| - |V_j| : i, j \in \{1, 2, \ldots, k\}\}$$

We say that a coloring χ is *balanced* when $\text{disc}(\chi) \leq 1$. Balanced proper colorings are also called *equitable colorings* [13].

The following are some known results that we use in the following sections.

▶ **Proposition 2** (Euler's Formula). *Let G be a connected planar graph with n vertices, m edges, and l faces. Then $n - m + l = 2$.*

▶ **Proposition 3** (Diestel [9]). *Every planar graph with $n \geq 3$ vertices has at most $3n - 6$ edges and at most $2n - 4$ faces. Every planar triangulation with $n \geq 3$ vertices has exactly $3n - 6$ edges and $2n - 4$ faces. Every planar quadrangulation with $n \geq 3$ vertices has exactly $2n - 4$ edges and $n - 2$ faces.*

An *independent set* I of G is a subset of vertices such that no two elements of I are adjacent. Observe that in a proper coloring, every color class is an independent set. The *independence number* of G, denoted by $\alpha(G)$, is the size of the maximum independent set.

▶ **Proposition 4** (Caro and Roditty [7]). *Let G be a planar triangulation with $n \geq 4$ vertices and minimum degree $\delta(G)$. Then $\alpha(G) \leq \frac{2n-4}{\delta(G)}$.*

Given a planar graph G and its planar embedding, the *dual graph* of G, denoted by G^* is such that the set of vertices of G^* corresponds to the set of faces of G, and two vertices are adjacent if and only if the corresponding two faces share an edge of their boundary. The following claim is implied by Tait's reformulation of the Four Color Theorem [16].

▶ **Proposition 5.** *Let G be a planar triangulation. The dual graph G^* has a proper 3-edge-coloring.*

A *matching* M of G is a subset of edges such that no two elements of M share a common vertex. A *maximum matching* is a matching with maximum cardinality. A *perfect matching* is a matching that includes all vertices of G, and a *near-perfect matching* includes all of them, but one.

▶ **Proposition 6** (Nishizeki and Baybars [14]). *Every planar triangulation G with n vertices has a matching of size at least $\frac{n}{3}$.*

3 Upper Bounds

In this section, we prove that every planar triangulation has a polychromatic 2-coloring with discrepancy at most $\frac{3n-16}{7}$. We first show some bounds related to other parameters of G.

In the first subsection, given a planar triangulation G with a maximum matching M, we show how to find a polychromatic 2-coloring whose discrepancy is at most $n - \frac{4|M|}{3}$. In particular, the discrepancy is at most $\frac{n}{3}$ when G has a perfect matching.

In the second subsection, given a proper 4-coloring of the triangulation G we show how to partition the four color classes into two sets so that the resulting coloring is polychromatic and its discrepancy is bounded depending on the size of the largest color class in the 4-coloring. In particular, we guarantee the existence of a polychromatic 2-coloring with discrepancy at most $\frac{n}{3}$ for graphs with independent number at most $\frac{n}{2}$. In the last subsection, we extend this idea to improve the general upper bound.

3.1 Discrepancy and Matchings

Polychromatic 2-colorings of triangulations are closely related to spanning quadrangulations and matchings as stated in [1], [2], and [5]. It is known that every triangulation has a spanning quadrangulation, where a *quadrangulation* is a plane graph such that each of its faces is a quadrilateral. It is also known that every quadrangulation is bipartite, and hence 2-colorable.

Let G be a planar triangulation, and let Q be a spanning quadrangulation of G. Every face of Q is the union of two adjacent faces of G, with their shared edge removed. Moreover, Q contains exactly two boundary edges of each face of G. Observe that the unique proper 2-coloring of $V(Q)$ corresponds to a polychromatic 2-coloring of $V(G)$. We now show how the size of the discrepancy relates to the size of a matching of Q in the following lemma.

▶ **Lemma 7.** *Let ρ be the unique proper 2-coloring of Q and let M_Q be a matching in Q. Then, ρ is a polychromatic 2-coloring in G with* $\text{disc}(\rho) \leq n - 2|M_Q|$.

Proof. Let R and B be the color classes in ρ. Every edge in the matching M_Q has one endpoint in each color class. Let $U_R \subseteq R$ and $U_B \subseteq B$ be the unmatched vertices. Then $|R| = |M_Q| + |U_R|$ and $|B| = |M_Q| + |U_B|$. Without loss of generality, assume that $|R| \geq |B|$. Then, $\text{disc}(\chi) = |R| - |B| = |U_R| - |U_B| \leq |U_R| + |U_B| = n - 2|M_Q|$. ◀

Now, to define an appropriate spanning quadrangulation with the appropriate matching, the strategy is to start with a matching M in the triangulation G, and then show that there exists a spanning quadrangulation containing a large fraction of edges from M.

There is a well-known equivalence between the 4-coloring of the vertices of a triangulation G and the 3-coloring of the edges of its dual G^*. Such edge-coloring induces a partition of the edges of G^* into 3 perfect matchings of G^*. Let E_1, E_2 and E_3 be the sets of edges of G corresponding to each of the 3 perfect matchings in G^*. Let $Q_i = G \setminus \{E_i\}$ for $i \in \{1, 2, 3\}$. We note that each Q_i is a spanning quadrangulation of G. (see e.g. [1], [5], [6], and [12]).

▶ **Lemma 8.** *Let M be a matching in G. There exists a spanning quadrangulation of G that has a matching of size at least $\frac{2}{3}|M|$.*

Proof. Every edge of M appears by construction in exactly 2 of the Q_i's, which implies that one of the Q_i's contains at least $\frac{2}{3}|M|$ edges of M. ◀

The following is the main result of this subsection.

▶ **Theorem 9.** *Every triangulation G with n vertices has a polychromatic 2-coloring with discrepancy at most $n - \frac{4}{3}|M|$, where M is a matching in G.*

Proof. By Lemma 8, G has a a spanning quadrangulation Q that has a matching M_Q of size at least $\frac{2}{3}|M|$. By Lemma 7 the proper 2-coloring of Q is a polychromatic 2-coloring of G with discrepancy at most $n - 2|M_Q| \leq n - \frac{4}{3}|M|$. ◀

▶ **Corollary 10.** *Every planar triangulation G has a polychromatic 2-coloring whose discrepancy is at most $\frac{5n-16}{9}$.*

Proof. Follows from Theorem 9 with the matching that is guaranteed by Proposition 6. ◀

A relevant application of Theorem 9 is to obtain a better upper bound for the discrepancy of polychromatic 2-colorings in Delaunay triangulations, stated in the following Corollary.

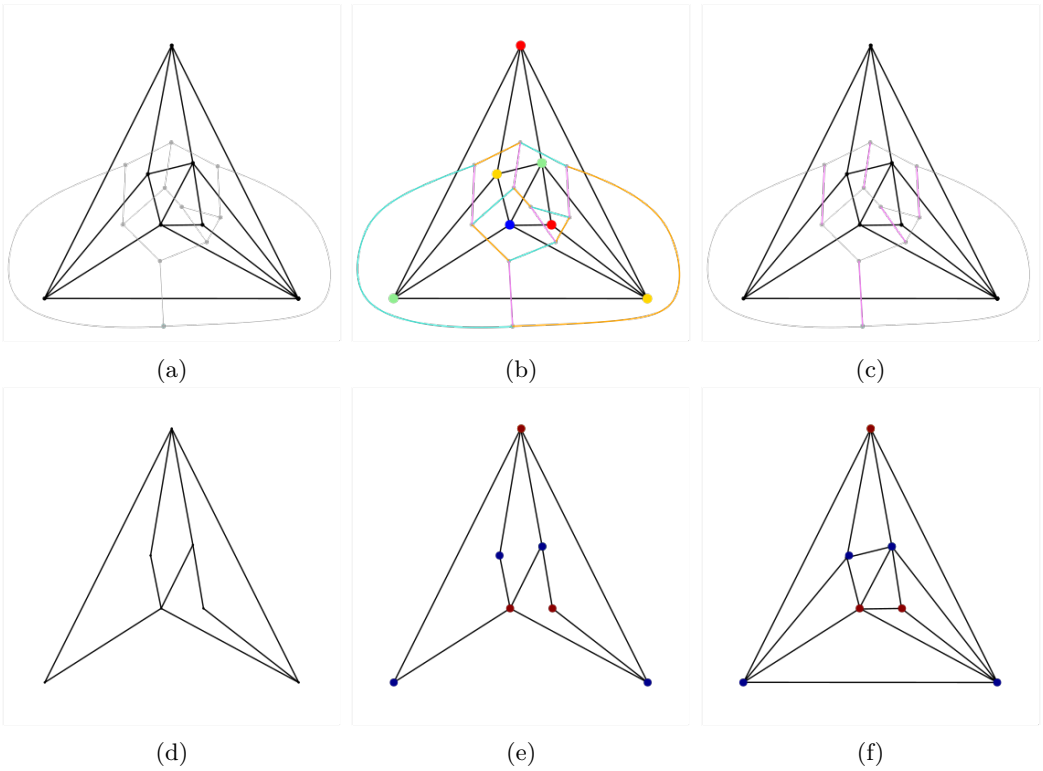


Figure 1 (a) A triangulation G and its dual graph G^*. (b) Equivalence between a 4-coloring on $V(G)$ and a 3-coloring on $E(G)$. (c) Perfect matching on G^*, corresponding to one color class in the 3-edge-coloring. (d) Spanning quadrangulation of G induced by deleting the dual edges of the perfect matching on G^*. (e) Proper 2-coloring on $V(Q)$. (f) Polychromatic 2-coloring on $V(G)$.

▶ **Corollary 11.** *Let D a graph on n vertices corresponding to the Delaunay triangulation of a set of n points in the plane. D has a polychromatic 2-coloring whose discrepancy is at most $\frac{n}{3}$.*

Proof. It is known that Delaunay triangulations have perfect matchings [3, 10]. Then by Theorem 9, D has a polychromatic 2-coloring whose discrepancy is at most $n - \frac{4}{3}\left(\frac{n}{2}\right)$. ◀

Observe that the bound $\frac{n}{3}$ is true for any triangulation with a perfect matching.

3.2 Discrepancy and Independent Sets

In this section, the strategy is to start with a proper 4-coloring of the triangulation, and define a 2-coloring that is polychromatic whose discrepancy is bounded according to the sizes of the color classes in the 4-coloring. This idea was used in [1] to prove the general bound. We extend the idea to relate the discrepancy to the size of the maximum independent set, which generalizes two of their main results and Theorem 9.

Let G be a planar triangulation with n vertices. Let $\chi : V(G) \to \{1, 2, 3, 4\}$ be a 4-proper coloring of $V(G)$. Let V_1, V_2, V_3, V_4 be the respective color classes and let n_1, n_2, n_3, n_4 be their respective sizes. We note that $n_1 + n_2 + n_3 + n_4 = n$. Without loss of generality, assume $n_1 \geq n_2 \geq n_3 \geq n_4$. Since $n_2 + n_3 + n_4 = n - n_1$ we get the following inequality:

▶ **Observation 12.** $n_4 \leq \frac{n-n_1}{3}$.

We describe a 2-coloring ρ of $V(G)$, whose color classes are R, B (sometimes referred to as the colors red and blue), and r, b their respective sizes. The discrepancy of ρ is defined as $\text{disc}(\rho) = |r - b|$ with $r + b = n$. Therefore, $\text{disc}(\rho) = |2r - n|$. We consider two cases to eliminate the absolute value.

▶ **Observation 13.** *If $r \geq \frac{n}{2}$, then $\text{disc}(\rho) = 2r - n$. If $r < \frac{n}{2}$, then $\text{disc}(\rho) = n - 2r$.*

▶ **Lemma 14.** *If $n_1 + n_4 \leq \frac{2n}{3}$ or $n_1 \leq \frac{n}{2}$, then G has a polychromatic 2-coloring whose discrepancy is at most $\frac{n}{3}$.*

Proof. Let ρ be the 2-coloring that assigns $R = V_1 \cup V_4$ and $B = V_2 \cup V_3$. Then $r = n_1 + n_4$ and $b = n_2 + n_3$. We claim that ρ is a polychromatic 2-coloring. Since every face of G is triangular, it contains vertices from 3 different color classes of the 4-coloring χ. Therefore, in ρ every face contains at least one red and one blue vertex.

Since $n_1 + n_4 \geq n_2 \geq n_3$, we have that $r = n_1 + n_4 \geq \frac{n}{3}$. Thus, if $n_1 + n_4 \leq \frac{2n}{3}$ then $\text{disc}(\rho) = |2r - n| \leq \frac{n}{3}$.

If $n_1 \leq \frac{n}{2}$ then by applying Observation 12 we get $n_2 + n_3 = n - n_1 - n_4 \geq \frac{2n-2n_1}{3} \geq \frac{n}{3}$. Once again we get then $\text{disc}(\rho) = |2r - n| \leq \frac{n}{3}$. ◀

▶ **Lemma 15.** *If $n_1 + n_4 \geq \frac{n}{2}$, then G has a polychromatic 2-coloring whose discrepancy is at most $\frac{4n_1-n}{3}$.*

Proof. Let ρ be the polychromatic 2-coloring defined in the proof of Lemma 14. By observation 12, we know that $r = n_1 + n_4 \leq \frac{n+2n_1}{3}$. Since $r = n_1 + n_4 \geq \frac{n}{2}$, we have $\text{disc}(\rho) = 2r - n$. We conclude that

$$\text{disc}(\rho) \leq 2\left(\frac{n + 2n_1}{3}\right) - n \leq \frac{4n_1 - n}{3}.$$ ◀

▶ **Theorem 16.** *Let G be a planar triangulation with n vertices. If G has a proper 4-coloring χ whose largest color class has size at most s, then G has a polychromatic 2-coloring ρ such that $\text{disc}(\rho) \leq \max\left(\frac{n}{3}, \frac{4s-n}{3}\right)$.*

Proof. The sizes of the color classes in χ meet the conditions of Lemma 14 or 15. ◀

▶ **Observation 17.** *By the Four Color Theorem, every triangulation has a proper 4-coloring. Since the color classes are independent sets, Theorem 16 can always be applied with $s = \alpha(G)$, the independence number of G.*

▶ **Corollary 18.** *Let G be a planar triangulation with $n \geq 4$ vertices. Then G has a polychromatic 2-coloring whose discrepancy is at most $\frac{5n-16}{9}$. Moreover, if $\delta(G) \geq 4$ then G has a polychromatic 2-coloring whose discrepancy is at most $\frac{n}{3}$.*

Proof. By the Four Color Theorem, G has a proper 4-coloring χ. Let n_1 be the size of the largest color class of χ. By Observation 17 and Proposition 4, we have $n_1 \leq \alpha(G) \leq \frac{2n-4}{\delta(G)}$ and we can apply Theorem 16 with $s = \frac{2n-4}{\delta(G)}$. Since every planar triangulation with $n \geq 4$ vertices has minimum degree at least 3, it follows that:

$$\text{disc}(\rho) \leq \max\left(\frac{n}{3}, \frac{4\left(\frac{2n-4}{3}\right) - n}{3}\right) = \frac{5n - 16}{9}.$$

If $\delta(G) \geq 4$ then

$$\text{disc}(\rho) \leq \max\left(\frac{n}{3}, \frac{4\left(\frac{2n-4}{\delta(G)}\right) - n}{3}\right) = \frac{n}{3}.$$ ◀

There is a relation between the size of a matching and the size of the maximum independent set in G.

▶ **Observation 19.** *Given a matching M of G, any subset of more than $n - |M|$ vertices includes an edge of M and then it is not independent. Therefore, $\alpha(G) \leq n - |M|$ for any matching M. Based on this fact and Observation 17, we can apply Theorem 16 with $s = n - |M|$ and get Theorem 9.*

We have provided an alternative proof for two theorems from [1]. Moreover, we have generalized these results by establishing an upper bound on the discrepancy of a polychromatic 2-coloring in terms of an upper bound on the size of the largest color class in a proper 4-coloring.

3.3 Improving the General Discrepancy Bounds

In this section we prove the following theorem:

▶ **Theorem 20.** *Let G be a planar triangulation with $n \geq 6$ vertices. Then G has a polychromatic 2-coloring whose discrepancy is at most $\frac{3n-16}{7}$.*

Recall the two color classes from the previous section (that are obtained as a partition of the colors of a proper 4-coloring). In view of Lemma 14 we may assume that $n_1+n_4 \geq n_2+n_3$ as otherwise the discrepancy is at most $n/3$. The main proof idea is to start by coloring all vertices in V_1 red, all vertices in $V_2 \cup V_3$ blue, and possibly splitting the vertices in V_4 between red and blue to get a smaller discrepancy.

Let $F_1 \subset F(G)$ be the set of faces that contain a vertex from V_1 and let $F_{234} = F(G) \setminus F_1$ be the set of faces that do not. If $V_1 \subseteq R$ and $V_2 \cup V_3 \subseteq B$ then all faces in F_1 are guaranteed to be polychromatic, regardless of the color of the vertices in V_4. In addition, every face $f \in F_{234}$ contains a vertex from V_4 that we can assign to R to make ρ polychromatic. Let $S_4 \subseteq V_4$ be the set of vertices in V_4 belonging to at least one face in F_{234}. Let $s_4 = |S_4|$. We call the vertices in S_4 *essential*, since they are required to be in the color class R to make the coloring polychromatic. The rest of the vertices in V_4 are called *non-essential*, and we denote the set as $\bar{S}_4 = V_4 \setminus S_4$. Let ρ be the 2-coloring that assigns $R = V_1 \cup S_4$ and let $B = V_2 \cup V_3 \cup \bar{S}_4$. The sizes of the color classes of ρ are $r = n_1 + s_4$ and $b = n_2 + n_3 + n_4 - s_4$. Observe that ρ is a polychromatic 2-coloring and $disc(\rho) = 2r - n$, by Observation 13 and Lemma 14. As a warm-up, in the following proposition we bound the discrepancy of ρ by analyzing the properties of S_4; this result is independent of the rest of the section.

▶ **Proposition 21.** $disc(\rho) \leq \frac{n-4}{2}$.

Proof. We give an upper bound for $r = n_1 + s_4$ based on the following observations:

1. Every vertex in V_1 is in at least 3 faces from F_1 because $\delta(G) \geq 3$. Every face in F_1 has exactly one vertex from V_1. Then $3n_1 \leq |F_1|$.
2. Every vertex in S_4 is in at least one face from F_{234} by definition. Every face in F_{234} has exactly one vertex in S_4. Then $s_4 \leq |F_{234}|$.

By Proposition 3, the total number of faces of G is $|F_1| + |F_{234}| = 2n - 4$, then we conclude that:

$$3n_1 + s_4 \leq |F_1| + |F_{234}| = 2n - 4 \tag{1}$$

We know that $S_4 \subseteq V_4$, then $s_4 \leq n_4$. By Observation 12, we have $n_4 \leq \frac{n-n_1}{3}$, which implies that:

$$n_1 + 3s_4 \leq n_1 + 3\left(\frac{n-n_1}{3}\right) = n \tag{2}$$

By adding up the equations 1 and 2, we get $4n_1 + 4s_4 \leq 3n - 4$. Then $r = n_1 + s_4 \leq \frac{3n-4}{4}$ and $\text{disc}(\rho) = 2r - n \leq \frac{n-4}{2}$. ◀

Some extra observations help to improve this bound. Let G be a planar triangulation with n vertices. Let χ_{min} be a proper 4-coloring of $V(G)$ such that the difference between the sizes of the largest and the smallest color class, i.e. $n_1 - n_4$, is minimum over all proper 4-colorings of $V(G)$. From the results in Subsection 3.2 (Lemma 14), we know that if $n_1 + n_4 \leq \frac{2n}{3}$ or $n_1 \leq \frac{n}{2}$, then the coloring that assigns $R = V_1 \cup V_4$ and $B = V_2 \cup V_3$ is polychromatic and has a discrepancy at most $\frac{n}{3}$. Thus, we may assume that $n_1 > \frac{n}{2}$ and $n_1 + n_4 > \frac{2n}{3}$ in the following. Then, Observation 12 implies that $n_4 \leq \frac{n}{6}$. Therefore, we get

▶ **Observation 22.** $n_1 - n_4 \geq \frac{n}{3}$.

Let $V_1^{d=3}$ be the set of vertices in V_1 whose degree equals 3 and let $V_1^{d\geq 4} = V_1 \setminus V_1^{d=3}$ be those with degree at least 4.

We refer to the faces in F_{234} as *good faces*. We classify every essential vertex as *type I* if it is in exactly one good face and *type II* if it is in at least two good faces. For any vertex v in G, let $N_i(v)$ be the set of neighbors of v in V_i for all $i \in \{1, 2, 3, 4\}$.

▶ **Lemma 23.** *Every essential vertex x of type I has an odd degree $d(x) \geq 3$ and $|N_1(x)| = \frac{d(x)-1}{2}$.*

Proof. Let $x \in S_4$ be an essential vertex of type I with degree $d(x)$. Since x is in exactly one good face, there is exactly one vertex in $N_2(x)$ and one vertex in $N_3(x)$ that are adjacent to each other. The other $d(x) - 2$ vertices are alternating between elements in $N_1(x)$ and elements in $N_2(x) \cup N_3(x)$ starting and ending with color 1 adjacent to the good face, otherwise we contradict that vertex x is of type I. This implies that $\frac{d(x)-1}{2}$ of the neighbors have color 1. ◀

We say that a connected subgraph of G is *i-j-alternating* if it is properly 2-colored with colors i and j.

▶ **Lemma 24.** *Let $n > 4$. If $x \in S_4$ is an essential vertex and every vertex in $N_1(x)$ has degree 3, then the vertices in $N_2(x) \cup N_3(x)$ form a 2-3-alternating separating cycle C in G. Moreover, if x is type I, then x has odd degree $d(x) \geq 7$.*

Proof. Let $y \in N_1(x)$. Since $d(y) = 3$, y is inside a separating triangle formed by x and the endpoints of a 2-3-edge. Then there is a 2-3-edge for each vertex in $N_1(x)$ and a 2-3 edge for each good face incident to x, all those edges together form a 2-3-alternating separating cycle C whose inside vertices are $x \in V_4$ and all the vertices in $N_1(x)$. Let $\ell(C)$ be the length of C. Observe that $\ell(C)$ is even and at least 4, so in the case where x is of type I, exactly one of the edges in C is part of a good face containing x and $\ell(C) - 1$ edges are part of a separating triangle that contains a vertex of color 1. Then $|N_1(x)| = \ell(C) - 1$. Note that $d(x) = |N_1(x)| + \ell(C) = 2\ell(C) - 1 \geq 2(4) - 1 = 7$. ◀

▶ **Observation 25.** *For every vertex x satisfying Lemma 24, $|N_1(x)| < |N_2(x) \cup N_3(x)|$ as shown in the proof of the lemma. Since we assume that $n_2 + n_3 < \frac{n}{3}$, we have that $|N_1(x)| < \frac{n}{3}$.*

▶ **Observation 26.** *Let C be a 2-3-alternating separating cycle. If we swap the colors 1 and 4 of the vertices inside C, we obtain another valid 4-coloring of $V(G)$.*

▶ **Lemma 27.** *Every essential vertex x of type I is adjacent to at least one vertex in $V_1^{d\geq 4}$.*

Proof. Let $x \in S_4$ be an essential vertex of type I. By Lemma 23, since $d(x) \geq 3$, $|N_1(x)| \geq 1$. Assume for a contradiction that every vertex in $N_1(x)$ has degree 3. By Lemmas 23 and 24, x has odd degree $d(x) \geq 7$, $|N_1(x)| = \frac{d(x)-1}{2} \geq 3$ and there is a 2-3-separating alternating

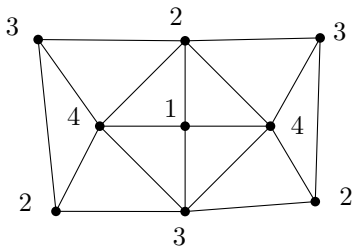


Figure 2 Illustration for Observation 28.

cycle C containing exactly one vertex of color 4, namely x, and $|N_1(x)|$ vertices of color 1 in its interior. By swapping colors 1 and 4 in C (Observation 26) we obtain a valid proper 4-coloring χ' whose color classes that has $n_1 - |N_1(x)| + 1$ vertices of color 1 and $n_4 - 1 + |N_1(x)|$ vertices of color 4. This would decrease the difference between n_1 and n_4 and hence contradicts our choice of χ_{min}.

We have to be careful here as color 4 may now be the largest color class in χ'. Hence the difference would be

$$(n_4 - 1 + |N_1(x)|) - (n_1 - |N_1(x)| + 1) = (n_4 - n_1) + 2|N_1(x)| - 2 < \frac{n}{3},$$

where the inequality is valid by Observation 22 and Observation 26. This still contradicts our choice of χ_{min} in view of Observation 26. ◀

Similar to the partition of V_4 in essential and non-essential vertices, we partition V_1 according to the faces they belong to. Let $S_1 \subseteq V_1$ be the set of vertices in V_1 belonging to at least one face that does not contain a vertex from S_4. Let $s_1 = |S_1|$. Let $\bar{S}_1 = V_1 \setminus S_1$ and call these vertices *non-essential* since they can be colored red or blue without violating the polychromatic 2-coloring property.

▶ **Observation 28.** *If a vertex $y \in V_1$ has degree 4 and two neighbors in S_4, then the four faces that contain y contain a vertex in S_4 and thus y is non-essential.*

In the following Lemma we state an upper bound for $s_1 + s_4$, which we later use to bound the discrepancy of a polychromatic 2-coloring.

▶ **Lemma 29.** $s_1 + s_4 \leq \frac{5n-8}{7}$.

Proof. We use a discharging argument. We assign 1 unit of charge to every face in F_{234}. For every vertex $y \in V_1$, we assign a charge equal to $d(y)$. Thus, the total initial charge is $|F_{234}| + |F_1| = 2n - 4$. Apply the transfer of charge as follows: Each face in F_{234} transfers its charge to its unique vertex in V_4. For every vertex $y \in V_1$, if $d(y) \geq 4$, then transfer 1 unit of charge to each of its neighbors that is in S_4. The total final charge is equal to the total initial charge. We analyze the final charge in the vertices of S_1 and S_4.

▷ Claim 1. Every vertex in S_1 has final charge at least 3.

Let $y \in S_1$. We know $d(y) \geq 3$. If $d(y) = 3$, then y has final charge equal to 3. If $d(y) = 4$, then y has at most one neighbour in S_4, otherwise it contradicts Observation 28. Therefore, the final charge is at least 3. If $d(y) \geq 5$, then y has at most $\lfloor \frac{d(y)}{2} \rfloor$ neighbors in S_4, since no two color 4 vertices are adjacent. Thus, the final charge of y is at least $d(y) - \lfloor \frac{d(y)}{2} \rfloor = \lceil \frac{d(y)}{2} \rceil \geq \lceil \frac{5}{2} \rceil = 3$.

▷ Claim 2. Every vertex in S_4 has final charge at least 2.

Let $x \in S_4$ be an essential vertex. If x is of type II, then it gets at least 2 units of charge from the two faces in F_{234}. Assume x is type I, then it gets 1 unit of charge from its incident F_{234} face. By Lemma 27, x is adjacent to a vertex $y \in V_1$ such that $d(y) \geq 4$, from which it gets an additional unit of charge. Thus, every vertex in S_4 has final charge at least 2.

Since the total final charge is equal to the total initial charge, the final charge in $S_1 \cup S_4$ is at most the total initial charge.

$$3s_1 + 2s_4 \leq 2n - 4 \tag{3}$$

We know that $S_1 \subseteq V_1$ and $S_4 \subseteq V_4$, then $s_1 \leq n_1$ and $s_4 \leq n_4$. By Observation 12, we have $n_4 \leq \frac{n-n_1}{3}$, which implies that:

$$s_1 + 3s_4 \leq n_1 + 3\left(\frac{n-n_1}{3}\right) = n \tag{4}$$

Multiplying by 2 both sides of (3) and adding it to (4), we get $7s_1 + 7s_4 \leq 5n - 8$. Then $s_1 + s_4 \leq \frac{5n-8}{7}$. ◀

We know that $S_1 \cup S_4 \subseteq V_1 \cup V_4$, and $|V_1 \cup V_4| = n_1 + n_4 > \frac{n}{2}$. If $s_1 + s_4 \geq \frac{n}{2}$, then let ρ be the 2-coloring that assigns $R = S_1 \cup S_4$. Otherwise, add some vertices from $\bar{S}_1 \cup \bar{S}_4$ to R in order to guarantee $r = |R| \geq \frac{n}{2}$. The sizes of the color classes of ρ are $r = \max(s_1 + s_4, \frac{n}{2})$ and $b = n - r$. Observe that ρ is a polychromatic 2-coloring and $disc(\rho) = 2r - n$. By Lemma 29, $\text{disc}(\rho) = 2r - n \leq 2\left(\frac{5n-8}{7}\right) - n = \frac{3n-16}{7}$. This finishes the proof of Theorem 20.

▶ Remark. The only place that we use the minimality of our 4-coloring is the contradictory argument in the proof of Lemma 27 where a color-4 vertex x and its color-1 neighbors are inside a 2-3-separating cycle formed by its color-2 and color-3 neighbors. Any such cycle is identified by the neighbors of a color-4 vertex. To get an algorithm, we could start from an arbitrary 4-coloring (that can be computed in $O(n^2)$ time [15]) and then for each such color-4 vertex x we swap the color of x with its color-1 neighhbors. This reduces $n_1 - n_4$. This extra step takes a total of $O(n)$ time for all such x's.

4 Linear-time Algorithm to Compute a Polychromatic 2-Coloring with Bounded Discrepancy for Planar Triangulations

In this section, we show how to compute, for a given planar triangulation, a polychromatic 2-coloring whose discrepancy is at most $\frac{5n-24}{7}$, in linear time. We use the linear-time algorithm by Bose et al. [5] to compute an initial polychromatic 2-coloring. Their algorithm uses a linear time algorithm to compute a maximum matching in the dual graph of a triangulation by Biedl et al. [2]. Thus, for any given planar graph, a polychromatic 2-coloring can be computed in linear time. However, this polychromatic 2-coloring may not have a guarantee on the discrepancy. We show how to extend this algorithm to guarantee a discrepancy of at most $\frac{5n-24}{7}$, without increasing the time complexity. The main approach is to make local modifications. We start by showing that the discrepancy of any polychromatic 2-coloring on the vertices of a triangulation G has an upper bound in terms of the maximum degree of G.

▶ **Theorem 30.** *Let G be a planar triangulation with $n \geq 3$ vertices and maximum degree $\Delta(G)$. Every polychromatic 2-coloring of G has discrepancy at most $\frac{(\Delta(G)-4)n+8}{\Delta(G)}$.*

Proof. Let G be a planar triangulation with n vertices, and $\chi : V(G) \to \{c_1, c_2\}$ a polychromatic 2-coloring of G. By Proposition 3 the number of faces of G is exactly $2n-4$. Since χ is a polychromatic coloring, every face contains at least one vertex of each color class. The number of faces that contain a vertex v is the degree of the vertex, $d(v)$. Then, any vertex belongs to at most $\Delta(G)$ faces. Therefore, every color class of χ has size at least $\frac{2n-4}{\Delta(G)}$. This implies that $\text{disc}(\chi) \leq n - 2\left(\frac{2n-4}{\Delta(G)}\right) = \frac{(\Delta(G)-4)n+8}{\Delta(G)}$. ◀

To bound the discrepancy after a recoloring process, we define a class of colorings in which there is no single vertex to recolor that decreases the discrepancy and preserves polychromaticity. A polychromatic 2-coloring ρ of the vertices of a triangulation G is *locally minimal* if there is no vertex in $V(G)$ that can be recolored to reduce the discrepancy of ρ without producing a monochromatic face. This naturally raises the question: How large can the discrepancy of a locally minimal coloring be?

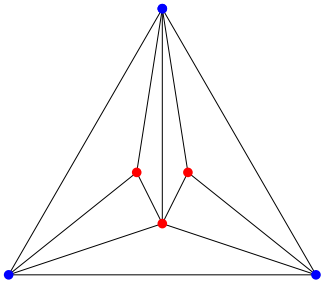


Figure 3 Construction that can be repeated in every face of a triangulation to get a locally minimal coloring. Blue points represent vertices in B and red points represent vertices in R.

▶ **Theorem 31.** *Let G be a planar triangulation with $n \geq 6$ vertices. Every polychromatic 2-coloring of G that is locally minimal has discrepancy at most $\frac{5n-24}{7}$.*

Proof. We start with the upper bound on the discrepancy. Consider an arbitrary planar triangulation G and a locally minimal polychromatic 2-coloring $\hat{\rho}$ as described before. Let R, B be the respective color classes, and let r, b be their respective sizes. Without loss of generality, assume $r \geq b$. If $r \leq b+1$ then the discrepancy is at most 1. Assume that $r \geq b+2$. Due to minimality of $\hat{\rho}$, every vertex $v \in R$ is incident to a face with an edge $e = (x, y)$ such that both x and y are in R, because otherwise we could recolor v to reduce discrepancy. Each such edge e is in exactly two faces of G, and hence count for at most two vertices in R. That means that the number of monochromatic edges of color B is at least $\frac{r}{2}$. Let G_2 be the subgraph of G induced by B. G_2 is a planar graph with b vertices, then by Proposition 3 it has at most $3b-6$ edges. Therefore, there are no more than $3b-6$ monochromatic edges of color B. It follows that $\frac{r}{2} \leq 3b-6$. Since $r+b=n$, we have that $7b-12 \geq n$, or $b \geq \frac{n+12}{7}$. Thus $r = n-b \leq n - \frac{n+12}{7} = \frac{6n-12}{7}$. The discrepancy $r-b$ is at most $\frac{6n-12}{7} - \frac{n+12}{7} = \frac{5n-24}{7}$.

We show that this bound is tight by constructing an infinite family of triangulations with a locally minimal polychromatic 2-coloring whose discrepancy is $\frac{5n-24}{7}$. We start with G_2 equal to a planar triangulation on b vertices of color B. Into each triangular face of G_2, we place three vertices of color R, connected as shown (for one face) in Figure 3. Note that no R-vertex can be changed to a B-vertex in any instance of this construction, as each is adjacent to an edge of G_2; thus the construction is minimal. With b vertices, G_2 has $2b-4$ triangular faces, by Proposition 3. This means that $r = 3(2b-4) = 6b-12$, and so the

construction has $r > b$ for each $b \geq 3$. Since $r + b = n$, we have $7b - 12 = n$, or $b = \frac{n+12}{7}$. Thus, $r = \frac{6n-12}{7}$, as in the upper bound. The discrepancy $r - b$ is then $\frac{5n-24}{7}$. So we meet the upper bound by starting with any triangulation of any size. ◀

▶ **Theorem 32.** *Let G be a planar triangulation with $n \geq 6$ vertices. A polychromatic 2-coloring on the vertices of G whose discrepancy is at most $\frac{5n-24}{7}$ can be computed in $O(n)$ time complexity.*

Proof. The proof follows from the following algorithm.

Algorithm 1 Algorithm to compute a minimal polychromatic 2-coloring.

Require: A planar triangulation G, with n vertices.
Ensure: A locally minimal polychromatic 2-coloring on the vertices of G.
Compute a polychromatic 2-coloring. Let R and B be the two color classes. Without loss of generality assume $|R| \geq |B|$.
while there is a vertex $v \in R$, such that v is not in a face with two vertices from B **do**
Recolor v from red to blue and update the color classes to $R \backslash \{v\}$ and $B \cup \{v\}$.
end while
Output the two color classes.

Correctness. The first step is to run the algorithm provided in [5], to compute a coloring with no monochromatic faces. The algorithm always maintains a valid polychromatic 2-coloring, since it only recolors a vertex when it does not produce monochromatic faces. We only recolor vertices from red to blue and every vertex is recolored at most once. Then the algorithm terminates, and the discrepancy decreases in every recoloring step until the coloring is locally minimal.

Running Time. Each iteration of the **while** loop examines a vertex and possibly changes its color. We say a vertex $v \in V$ is *recolorable* if $v \in R$, and v is not in a face with two vertices from B. We can process the vertices in any given order, since a vertex that is not recolorable never becomes recolorable. For every vertex v, we can verify if it is recolorable in $O(d(v))$ because we evaluate all the faces that contain v. Since every face is triangular, during the whole process we check every face three times, and since G is planar, the number of faces is $3n - 6$. Therefore, the total run of the algorithm takes $O(n)$ assuming constant-time adjacency and face incidence queries. ◀

5 Conclusions

We studied the discrepancy of polychromatic 2-colorings for planar triangulations and presented improved bounds and algorithms. We conclude with a few open questions for further investigation. The main conjecture in this area remains open: does every triangulation on n vertices admit a polychromatic 2-coloring with discrepancy at most $\frac{n}{3}$? From an algorithmic perspective, it remains open whether a polychromatic 2-coloring with discrepancy less than $\frac{5n-24}{7}$ can be computed in linear time.

References

1 Yoshihiro Asayama and Naoki Matsumoto. Balanced polychromatic 2-coloring of triangulations. *Graphs and Combinatorics*, 38(1):1–12, 2022.

2 Therese C. Biedl, Prosenjit Bose, Erik D. Demaine, and Anna Lubiw. Efficient algorithms for Petersen's matching theorem. *J. Algorithms*, 38(1):110–134, 2001. doi:10.1006/JAGM.2000.1132.

3 Ahmad Biniaz. A short proof of the toughness of delaunay triangulations. *J. Comput. Geom.*, 12(1):35–39, 2021. Also in SOSA'20. doi:10.20382/JOCG.V12I1A2.

4 J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008.

5 Prosenjit Bose, David Kirkpatrick, and Zaiqing Li. Worst-case-optimal algorithms for guarding planar graphs and polyhedral surfaces. *Computational Geometry*, 26(3):209–219, 2003. doi:10.1016/S0925-7721(03)00027-0.

6 Prosenjit Bose, Thomas Shermer, Godfried Toussaint, and Binhai Zhu. Guarding polyhedral terrains. *Computational Geometry*, 7(3):173–185, 1997. doi:10.1016/0925-7721(95)00034-8.

7 Yair Caro and Yehuda Roditty. On the vertex-independence number and star decomposition of graphs. *Ars Combin*, 20:167–180, 1985.

8 Július Czap and Stanislav Jendrol. Facially-constrained colorings of plane graphs: A survey. *Discrete Mathematics*, 340(11):2691–2703, 2017. doi:10.1016/J.DISC.2016.07.026.

9 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 6 edition, 2025.

10 M.B. Dillencourt. Toughness and delaunay triangulations. *Discrete & computational geometry*, 5(6):575–602, 1990. doi:10.1007/BF02187810.

11 Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 2011.

12 André Kündgen and Carsten Thomassen. Spanning quadrangulations of triangulated surfaces. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 87, pages 357–368. Springer, 2017.

13 Walter Meyer. Equitable coloring. *The American mathematical monthly*, 80(8):920–922, 1973.

14 Takao Nishizeki and Ilker Baybars. Lower bounds on the cardinality of the maximum matchings of planar graphs. *Discrete Mathematics*, 28(3):255–267, 1979. doi:10.1016/0012-365X(79)90133-X.

15 Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997. doi:10.1006/JCTB.1997.1750.

16 Tait. 4. on the colouring of maps. *Proceedings of the Royal Society of Edinburgh*, 10:501–503, 1880. doi:10.1017/S0370164600044229.

17 Zsolt Tuza. Graph coloring. In *Handbook of Graph Theory*, chapter 5, pages 408–438. CRC Press, 2nd edition, November 2013.

Linear-Time Exact Computation of Influence Spread on Bounded-Pathwidth Graphs

Kengo Nakamura
Communication Science Laboratories, NTT, Inc., Kyoto, Japan

Masaaki Nishino
Communication Science Laboratories, NTT, Inc., Kyoto, Japan

Abstract

Given a network and a set of vertices called seeds to initially inject information, influence spread is the expected number of vertices that eventually receive the information under a certain stochastic model of information propagation. Under the commonly used independent cascade model, influence spread is equivalent to the expected number of vertices reachable from the seeds on a directed uncertain graph, and the exact evaluation of influence spread offers many applications, e.g., influence maximization. Although its evaluation is a #P-hard task, there is an algorithm that can precisely compute the influence spread in $O(mn\omega_p^2 \cdot 2^{\omega_p^2})$ time, where ω_p is the pathwidth of the graph. We improve this by developing an algorithm that computes the influence spread in $O((m+n)\omega_p^2 \cdot 2^{\omega_p^2})$ time. This is achieved by identifying the similarities in the repetitive computations in the existing algorithm and sharing them to reduce computation. Although similar refinements have been considered for the probability computation on undirected uncertain graphs, a greater number of similarities must be leveraged for directed graphs to achieve linear time complexity.

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

Keywords and phrases Influence spread, bounded pathwidth, network reliability, linear time algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.34

Related Version *Full Version*: `https://arxiv.org/abs/2604.13526`

Funding This work was supported by JSPS KAKENHI Grant Number JP26K02906.

1 Introduction

Given a network and *seed set* S, which is a set of vertices that are initially injected information, *influence spread* [5] $\sigma(S)$ is the expected number of vertices that eventually receives the information under a stochastic model of information propagation. Influence spread is implemented in viral marketing applications with the aim of evaluating the influence of a particular node within a social network. Currently, it is also a keystone for other network applications such as network monitoring [11], rumor control [23], target advertisement [13], and social recommendation [24]. Among several information diffusion models, the most basic and commonly used one is the *independent cascade (IC) model* [6]. In this model, a network is modeled as a directed graph $G = (V, E)$, where each edge $e \in E$ is associated with a probability p_e. When a vertex v receives information, it stochastically propagates the information along every outgoing edge e independently with a given probability p_e.

The influence spread $\sigma(S)$ under the IC model is equivalent to the expected number of vertices reachable from at least one vertex in S on a directed *uncertain graph*. Given directed graph $G = (V, E)$ and probability p_e for every edge $e \in E$, we consider an uncertainty in which each edge $e \in E$ is present with probability p_e and absent with probability $1 - p_e$. We assume that each edge's presence or absence is stochastically independent of the other edges.

© Kengo Nakamura and Masaaki Nishino;
licensed under Creative Commons License CC-BY 4.0

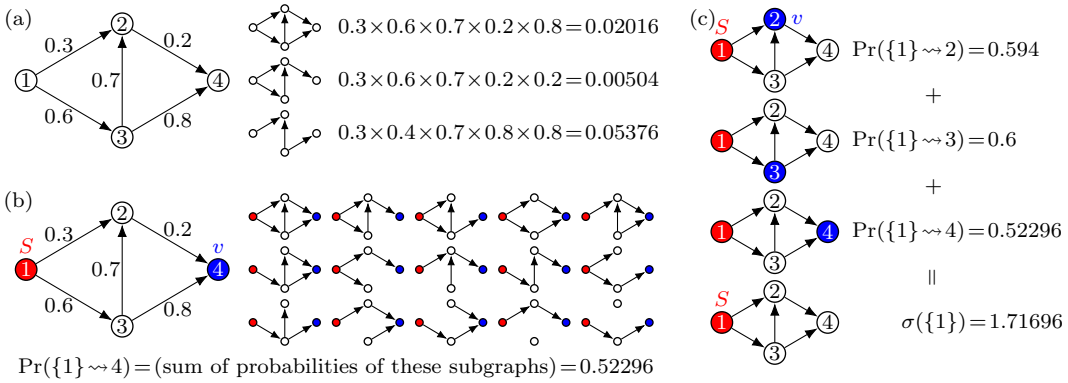


Figure 1 (a) Example of directed uncertain graph and probabilities of subgraphs. (b) All 15 subgraphs in which $v = 4$ is reachable from $\{S\} = 1$. $\Pr(\{1\} \rightsquigarrow 4)$ is the sum of probabilities of these subgraphs. (c) $\sigma(\{1\})$ is computed as the sum of $\Pr(\{1\} \rightsquigarrow v)$s for $v = 2, 3, 4$.

Accordingly, the probability that a subgraph (V, E') $(E' \subseteq E)$ appears is

$$\Pr(E') \coloneqq \prod_{e \in E'} p_e \cdot \prod_{e \in E \setminus E'} (1 - p_e). \tag{1}$$

Here, a directed path from some $s \in S$ to vertex v on the subgraph (V, E') corresponds to the route of information propagation from s to v. Therefore, given $S \subseteq V$ and $v \in V$, the probability that v receives information is equal to the probability $\Pr(S \rightsquigarrow v)$ that v can be reached from some vertices in S under uncertainty. Here, $\Pr(S \rightsquigarrow v) = \sum_{E' \in \mathcal{E}_{S \rightsquigarrow v}} \Pr(E')$, where $\mathcal{E}_{S \rightsquigarrow v}$ is the family of subsets E' of edges such that v can be reached from some vertices in S on (G, E'). Finally, the influence spread can be represented as $\sigma(S) = \sum_{v \in V \setminus S} \Pr(S \rightsquigarrow v)$.

▶ **Example 1.1.** Figure 1a depicts examples of the probabilities $\Pr(E')$ of subgraphs (V, E') $(E' \subseteq E)$. For this uncertain graph, there are 15 subgraphs in which $v = 4$ is reachable from $\{S\} = 1$ (Figure 1b); thus, $\Pr(\{1\} \rightsquigarrow 4)$ is the sum of $\Pr(E')$ over all these subgraphs. By evaluating $\Pr(\{1\} \rightsquigarrow 2)$, $\Pr(\{1\} \rightsquigarrow 3)$, and $\Pr(\{1\} \rightsquigarrow 4)$ as in Figure 1c, $\sigma(\{1\})$ can be computed as their sum.

Since evaluating the influence spread $\sigma(S)$ is a #P-hard task [3], most studies have relied on Monte Carlo simulation for evaluating $\sigma(S)$ [7, 10, 11, 18]. However, exact computation of $\sigma(S)$ is important for the following reasons. First, this allows us to more accurately evaluate the influence spread in larger networks, since real networks often consist of small communities; the exact computation of influence spread in each small community will improve overall evaluation quality. Second, this allows us to rank the influential vertices in descending order of influence spread. For this purpose, Monte Carlo simulations are not sufficient because $\Omega(1/\varepsilon^2)$ samples are needed to obtain $(1 \pm \varepsilon)$-approximation with high probability [18]; to distinguish every vertex's influence spread, high accuracy such as $\varepsilon \sim 10^{-5}$ is sometimes needed, which is costly for Monte Carlo methods.

Maehara et al. [15] proposed the only non-trivial algorithm to exactly compute $\sigma(S)$ under the IC model. Given seed set S and v, it efficiently computes $\Pr(S \rightsquigarrow v)$, i.e., the probability that v receives information. Given the path decomposition of G whose width is ω_p, this probability for a single v can be computed in $O(m\omega_p^2 \cdot 2^{\omega_p^2})$ time, where m is the number of edges. Hereafter, we mean the pathwidth of a directed graph G in the sense of the pathwidth of the underlying undirected graph of G as used in the literature [15, 20]. The influence spread $\sigma(S) = \sum_{v \in V \setminus S} \Pr(S \rightsquigarrow v)$ can be computed in $O(mn\omega_p^2 \cdot 2^{\omega_p^2})$ time, where n is the number of vertices.

Here, even for bounded-pathwidth graphs, obtaining the $\sigma(S)$ value requires super-linear ($O(mn)$) time. Since there are similarities in the repetitive computation for different vertices v using this algorithm, we investigated the use of these similarities to refine their algorithm and thus improve the running time for computing $\sigma(S)$.

1.1 Our Contribution

In this paper, we propose an algorithm for computing $\sigma(S)$ for given S under the IC model. Specifically, the proposed algorithm simultaneously computes $\Pr(S \rightsquigarrow v)$ in an uncertain directed graph for all vertices v. Our main result can be expressed as follows.

▶ **Theorem 1.2.** *Given directed graph G, seed set S, each edge's probability p_e of presence, and the path decomposition of G whose width is ω_p, the probability $\Pr(S \rightsquigarrow v)$ for every vertex v can be computed in $O((m+n)\omega_p^2 \cdot 2^{\omega_p^2})$ time in total.*

Since even inputting the entire graph requires linear time, this algorithm is asymptotically optimal for graphs with a bounded pathwidth when the constant factor is ignored. Moreover, the exponential factor of the pathwidth, hidden in the constant, remains the same as in the algorithm of Maehara et al. [15]: the dependence on the graph sizes is improved from $O(mn)$ to $O(m+n)$ while the dependence on the pathwidth is kept the same.

Our algorithm shares certain ideas with the existing algorithm for simultaneously computing the probabilities of connection in undirected graphs [16] in that both use similarities in the computation of the previous algorithms [8, 15] for different vertices. However, there is a non-trivial gap in extending that algorithm [16] to directed graphs due to the difference between connectivity in undirected graphs and reachability in directed graphs. We fill this gap by finding additional similarities in computing $\sigma(S)$ with the previous algorithm [15]; details are discussed in Section 5.

1.2 Related Work

Influence maximization. The most well-studied problem related to influence spread is influence maximization [10], that is, the problem of choosing a seed set S to maximize the influence spread. This has attracted much attention over the past two decades [12, 14, 19]. Although the influence maximization problem under most diffusion models including the IC model is NP-hard [10], a simple greedy algorithm [10] can achieve a $(1-1/e)$-approximation provided that every $\sigma(S)$ is computed exactly. Various methods, e.g., the sketch-based methods described in a survey [12], have improved this greedy algorithm. However, if we resorted to using Monte Carlo simulation to evaluate $\sigma(S)$, we could not obtain a deterministic approximation bound for influence maximization, even with greedy or more sophisticated algorithms. Moreover, although many studies have used an alternative proxy for $\sigma(S)$ to speed up the computation (e.g., proxy-based methods [12] and learning-based methods [14]), they also lack approximation bounds. This situation accentuates the theoretical importance of an exact evaluation of influence spread.

Network reliability. Computing the probability that some vertices are connected in an uncertain graph has been traditionally studied as *network reliability* evaluation in the network community. Most studies on network reliability evaluation have modeled the network as an *undirected* graph and, given a vertex set K and the probability p_e of presence for every edge, they computed the probability that the vertices in K are connected. Even for undirected graphs, computing this probability is #P-complete [22]. Nevertheless, Hardy et al. [8] developed an algorithm that could compute this probability

in linear time for graphs with bounded pathwidth. Another line of research has attempted to compute the expected number of vertices connected to vertex u [17], which is an analogue of the influence spread under the IC model for undirected graphs. Nakamura et al. [16] proposed an algorithm that could compute the probability that a vertex u is connected to v for *every* $v \in V$ in linear time for an undirected graph with bounded pathwidth. By summing all the probabilities for every v, it is also possible to compute the expected number of vertices connected to v in linear time. This algorithm uses the similarities in the computation of the probability that u and v are connected with Hardy's algorithm [8] for different v; therefore, it shares similar ideas with our work. However, we cannot straightforwardly extend this algorithm to handle directed graphs, as described in Section 1; detailed discussions are given in Section 5. Note that these algorithms [8, 16] also run in polynomial time for graphs with bounded pathwidth, indicating the importance of algorithms dealing with bounded-pathwidth graphs.

2 Preliminaries

For sets U, W of vertices, $U \rightsquigarrow_G W$ means that u can reach w, i.e., w can be reached from u, for some $u \in U$ and $w \in W$ in directed graph G. If U (or W) consists of only a single vertex u (w), we simply write, e.g., $u \rightsquigarrow_G W$ and $u \rightsquigarrow_G w$ instead of $\{u\} \rightsquigarrow_G W$ and $\{u\} \rightsquigarrow_G \{w\}$. When it is clear from the context, we omit the subscript G.

A directed graph is called *strongly connected* if $u \rightsquigarrow v$ for every ordered pair (u, v) of vertices. We say vertex subset C is a *strongly connected component* (SCC) of G if the vertex-induced subgraph of G induced by C is strongly connected but that induced by $V \cup \{v\}$ is not strongly connected for any $v \in V \setminus C$. It is known that, given directed graph G, all SCCs in G can be computed in linear time in the size of G [21]. Throughout this paper, n and m denote the number of vertices and edges, respectively, in the input graph G.

We formally define the problems to solve as follows.

▶ **Problem 2.1.** Given a directed graph G, probabilities $\{p_e\}_{e \in E}$, and a seed set $S \subseteq V$, compute the probability $\Pr(S \rightsquigarrow v)$ for every $v \in V \setminus S$.

For convenience, we assume that G contains no self-loops and the underlying undirected graph of G is connected. Note that if G contains self-loops, we can safely remove it, since it would never affect reachability. If G is disconnected, we can solve the problem individually for each connected component.

3 Review of Previous Algorithm

3.1 Decomposition of Probability

In the following, we review Maehara's algorithm [15] for exactly computing $\Pr(S \rightsquigarrow v)$ for a given S, v. Although the literature [15] only showed the algorithm for the case $|S| = 1$, here we show the general S case because it can be easily extended.

The basic idea in evaluating $\Pr(S \rightsquigarrow v)$ is to decompose $\Pr(S \rightsquigarrow v)$ into terms that can be easily computed. For $E', E'' \subseteq E$ such that $E' \cap E'' = \emptyset$, let $\Pr(\cdot \mid E', \overline{E''})$ be the conditional probability given that the edges in E' are present and those in E'' are absent. From the case analysis of whether $e \in E \setminus (E' \cup E'')$ is present or absent,

$$\Pr(S \rightsquigarrow v \mid E', \overline{E''}) = p_e \cdot \Pr(S \rightsquigarrow v \mid E' \cup \{e\}, \overline{E''}) + (1 - p_e) \cdot \Pr(S \rightsquigarrow v \mid E', \overline{E'' \cup \{e\}}). \quad (2)$$

By ordering edges as $e_1, \dots, e_m$, $\Pr(S \rightsquigarrow v)$ can be decomposed by recursively applying (2):

$$\begin{aligned}\Pr(S \rightsquigarrow v) &= p_{e_1} \cdot \Pr(S \rightsquigarrow v \mid \{e_1\}, \overline{\emptyset}) + (1-p_{e_1}) \cdot \Pr(S \rightsquigarrow v \mid \emptyset, \overline{\{e_1\}}) \\ &= p_{e_1}p_{e_2} \cdot \Pr(S \rightsquigarrow v \mid \{e_1, e_2\}, \overline{\emptyset}) + p_{e_1}(1-p_{e_2}) \cdot \Pr(S \rightsquigarrow v \mid \{e_1\}, \overline{\{e_2\}}) + \\ &\quad (1-p_{e_1})p_{e_2} \cdot \Pr(S \rightsquigarrow v \mid \{e_2\}, \overline{\{e_1\}}) + (1-p_{e_1})(1-p_{e_2}) \cdot \Pr(S \rightsquigarrow v \mid \emptyset, \overline{\{e_1, e_2\}}) \\ &= \cdots = \sum_{E' \subseteq E_{<i}} \Pr_{<i}(E') \cdot \Pr(S \rightsquigarrow v \mid E', \overline{E_{<i} \setminus E'}) = \cdots, \end{aligned} \tag{3}$$

where $E_{<i} \coloneqq \{e_1, \dots, e_{i-1}\}$, $\Pr_{<i}(E') \coloneqq \prod_{e \in E'} p_e \cdot \prod_{e \in E_{<i} \setminus E'} (1 - p_e)$.

This expansion eventually reaches the definition $\Pr(S \rightsquigarrow v) \coloneqq \sum_{E' \in \mathcal{E}_{S \rightsquigarrow v}} \Pr(E')$. Therefore, by using this decomposition naively, we have $O(2^m)$ terms, which incurs exponential complexity of the graph size in evaluating it.

To prevent this, we attempt to detect the equality among probabilities, i.e., to find subsets $E', F' \subseteq E_{<i}$ $(E' \neq F')$ such that $\Pr(S \rightsquigarrow v \mid E', \overline{E_{<i} \setminus E'}) = \Pr(S \rightsquigarrow v \mid F', \overline{E_{<i} \setminus F'})$. If such equality can be detected, we simply need to further decompose only one among the equal probabilities, which reduces the number of terms. If the following condition (#) holds, we can confirm $\Pr(S \rightsquigarrow v \mid E', \overline{E_{<i} \setminus E'}) = \Pr(S \rightsquigarrow v \mid F', \overline{E_{<i} \setminus F'})$:

(#) For any $H \subseteq E_{\geq i} \coloneqq E \setminus E_{<i}$, $S \rightsquigarrow_{(V, E' \cup H)} v$ if and only if $S \rightsquigarrow_{(V, F' \cup H)} v$.

To check condition (#), we focus on the reachability relation on the subgraph (V, E'). If $u \rightsquigarrow_{(V,E')} w$ and $w \rightsquigarrow_{(V,F')} v$ are equivalent for any $u, w \in V$, we can confirm that (#) holds. In brief, this is because, given a path from $s \in S$ to v on $(V, E' \cup H)$, we can construct a path from s to v on $(V, F' \cup H)$ by using the above equivalence, and vice versa.

Maehara et al. [15] indicated that only a limited subset of vertices called *frontier vertices* is important for checking (#).

▶ **Definition 3.1.** *Given edge ordering $e_1, \dots, e_m$, (i-th)* frontier vertices W_i *are those appearing in both $E_{<i}$ and $E_{\geq i}$.*

We consider the reachability relations among $W_i \cup S \cup \{v\}$ on (V, E') as follows.

▶ **Definition 3.2.** *Let $W_i^{S \rightsquigarrow}(E') \coloneqq \{w \in W_i \mid S \rightsquigarrow_{(V,E')} w\}$ (frontier vertices reachable from S) and $W_i^{\rightsquigarrow v}(E') \coloneqq \{w \in W_i \mid w \rightsquigarrow_{(V,E')} v\}$ (those that can reach v). A binary matrix $\Phi_i^v(E')$ is defined as follows: (I) It is indexed by $(W_i \setminus W_i^{S \rightsquigarrow}(E') \cup \{S\}) \times (W_i \setminus W_i^{\rightsquigarrow v}(E') \cup \{v\})$. (II) $\Phi_i^v(E')_{uw} = 1$ if and only if $u \rightsquigarrow_{(V,E')} w$. Here, S is treated as a special (single) row index, and v is treated as a column index. We call $\Phi_i^v(E')$ a* transversal configuration *(TC).*

We can then prove the following.

▶ **Lemma 3.3.** *$\Phi_i^v(E') = \Phi_i^v(F')$ is a sufficient condition for (#), meaning that $\Pr(S \rightsquigarrow v \mid E', \overline{E_{<i} \setminus E'}) = \Pr(S \rightsquigarrow v \mid F', \overline{E_{<i} \setminus F'})$. Note that $\Phi_i^v(E') = \Phi_i^v(F')$ means that the indices of matrices as well as their entries are equal.*

This can be shown by transforming a u-v directed path on $(V, E' \cup H)$ for some $u \in S$ to a u'-v directed path on $(V, F' \cup H)$ for some $u' \in S$. Now we can rewrite $\Pr(S \rightsquigarrow v \mid E', \overline{E_{<i} \setminus E'})$ as $\Pr(S \rightsquigarrow v \mid \Phi_i^v(E'))$.

▶ **Example 3.4.** On the graph in Figure 2a with $v = 5$, we examine $\{e_1, e_4\}, \{e_1, e_2, e_3, e_4\}, \{e_1, e_2, e_4, e_6\}, \{e_2, e_5, e_6\} \subseteq E_{<8}$ corresponding to the four edge subgraphs of Figure 2b. These subsets have an identical TC Φ_8^5 because $W_8 = \{4, 5\}$. Thus, the conditional probabilities of $S \rightsquigarrow 5$ are all equal for these subsets.

Algorithm 1 Previous algorithm [15].

```
1  N'_1 ← {Φ^v_1(∅)}, N'_i ← {} (i ≥ 2)
2  for i ← 1 to m do                          // Diagram construction
3  |  foreach Φ ∈ N'_i do
4  |  |  foreach f ∈ {lo, hi} do
5  |  |  |  if f^v(Φ) = ⊥ then Φ.f ← ⊥
6  |  |  |  else if f^v(Φ) = ⊤ then Φ.f ← ⊤
7  |  |  |  else N'_{i+1} ← N'_{i+1} ∪ {f^v(Φ)},
   |  |  |    Φ.f ← f^v(Φ) ∈ N'_{i+1}
8  p[Φ] ← 1 for Φ ∈ N'_1, p[Φ] = 0 for Φ ∈ N'_i (i ≥ 2)
9  for i ← 1 to m do                          // Top-down DP
10 |  foreach Φ ∈ N'_i do                    // p[Φ] stores Pr(Φ)
11 |  |  p[Φ.lo] += (1 − p_{e_i}) · p[Φ]
12 |  |  p[Φ.hi] += p_{e_i} · p[Φ]
13 return p[⊤]                               // Pr(S ⇝ v) = Pr(⊤)
```

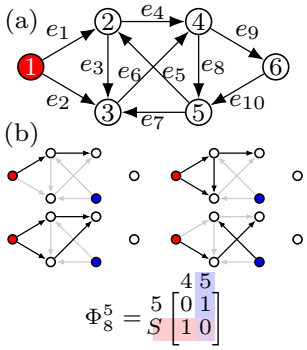


Figure 2 (a) Example of input graph G. Red vertex is included in S. (b) Example of different edge subsets of $E_{<8}$ leading to the same TC Φ^5_8.

Moreover, $\Phi^v_i(\cdot)$ has the following property: if $\Phi^v_i(E') = \Phi^v_i(F')$ for $E', F' \subseteq E_{<i}$, both $\Phi^v_{i+1}(E') = \Phi^v_{i+1}(F')$ and $\Phi^v_{i+1}(E' \cup \{e_i\}) = \Phi^v_{i+1}(F' \cup \{e_i\})$ hold. Thus, there exist transition functions $\mathsf{lo}^v, \mathsf{hi}^v$ from Φ^v_i to Φ^v_{i+1} satisfying $\mathsf{lo}^v(\Phi^v_i(E')) = \Phi^v_{i+1}(E')$ and $\mathsf{hi}^v(\Phi^v_i(E')) = \Phi^v_{i+1}(E' \cup \{e_i\})$ for any $E' \subseteq E_{<i}$. By combining this fact and (2), we have, for any $\Phi = \Phi^v_i(E')$ $(E' \subseteq E_{<i})$,

$$\Pr(S \rightsquigarrow v \mid \Phi) = p_e \cdot \Pr(S \rightsquigarrow v \mid \mathsf{hi}^v(\Phi)) + (1 - p_e) \cdot \Pr(S \rightsquigarrow v \mid \mathsf{lo}^v(\Phi)). \tag{4}$$

Using this decomposition recursively, $\Pr(S \rightsquigarrow v)$ can be decomposed as follows:

$$\begin{aligned} \Pr(S \rightsquigarrow v) &= \Pr(S \rightsquigarrow v \mid \Phi^v_1(\emptyset)) \\ &= p_{e_1} \cdot \Pr(S \rightsquigarrow v \mid \mathsf{hi}^v(\Phi^v_1(\emptyset))) + (1 - p_{e_1}) \cdot \Pr(S \rightsquigarrow v \mid \mathsf{lo}^v(\Phi^v_1(\emptyset))) \\ &= \cdots = \sum_{\Phi \in \mathcal{N}_i} \Pr(\Phi) \cdot \Pr(S \rightsquigarrow v \mid \Phi) = \cdots, \end{aligned} \tag{5}$$

$$\text{where} \quad \Pr(\Phi) \coloneqq \sum_{E' \subseteq E_{<i} : \Phi^v_i(E') = \Phi} \Pr_{<i}(E'), \quad \mathcal{N}_i \coloneqq \{\Phi^v_i(E') \mid E' \subseteq E_{<i}\}. \tag{6}$$

In other words, $\Pr(\Phi)$ is the probability that TC Φ appears and $\mathcal{N}_i$ is the set of all TCs of the subsets of $E_{<i}$. We later show that $|\mathcal{N}_i|$ is bounded by a constant if the pathwidth is bounded in Lemma 4.15. This is the main reason to achieve $O(m)$ time complexity in computing $\Pr(S \rightsquigarrow v)$.

For some TCs Φ, we can confirm that $S \rightsquigarrow v$ must hold or must not hold, i.e., $\Pr(S \rightsquigarrow v \mid \Phi) = 1$ or 0. For such TCs, we do not need to further decompose $\Pr(S \rightsquigarrow v \mid \Phi)$ with (4). We define such base cases with the following $\top$-pruning and $\bot$-pruning as follows. Let i_u be the largest index i such that one of the endpoints of e_i is u and let $i_S \coloneqq \max_{u \in S} i_u$.

- If $\mathsf{hi}^v(\Phi)_{Sv} = 1$, determining the presence of e_i confirms $S \rightsquigarrow v$, i.e., $\Pr(S \rightsquigarrow v \mid \mathsf{hi}^v(\Phi)) = 1$. In such a case, we let $\mathsf{hi}^v(\Phi) = \top$, a special TC meaning that $S \rightsquigarrow v$ must hold ($\top$-pruning).
- Suppose that $i \geq i_S$ and $\mathsf{lo}^v(\Phi)_{Su} = 0$ for any u, or that $i \geq i_v$ and $\mathsf{lo}^v(\Phi)_{uv} = 0$ for any u. In these cases, no further vertices can be reached from S or reach v by determining the absence of e_i, i.e., $\Pr(S \rightsquigarrow v \mid \mathsf{lo}^v(\Phi)) = 0$. Thus, we let $\mathsf{lo}^v(\Phi) = \bot$, which is a special TC meaning that $S \rightsquigarrow v$ must not hold ($\bot$-pruning)[1]. We also let $\mathsf{hi}^v(\Phi) = \bot$ if $\top$-pruning does not occur and $\mathsf{hi}^v(\Phi)$ satisfies the same condition as above.

[1] Although the original version [15] uses different pruning rules, they require a pre-computation of transitive closures of $(V, E_{\geq i})$, which takes cubic time. We use an alternative pruning rule that can be checked in linear time.

Eventually, all the TCs are transformed into either $\top$ or $\bot$, and (5) reaches $\Pr(S \rightsquigarrow v) = \Pr(\top) \cdot 1 + \Pr(\bot) \cdot 0 = \Pr(\top)$. Thus, computing $\Pr(\top)$ is sufficient for evaluating $\Pr(S \rightsquigarrow v)$.

3.2 Procedure and Complexity

Now we describe the procedure to compute $\Pr(\top)$. We again use the transition of TCs. By considering the pruning described above, we redefine the set of TCs as $\mathcal{N}'_1 \coloneqq \{\Phi_1^v(\emptyset)\}$ and $\mathcal{N}'_i \coloneqq \left(\bigcup_{\Phi \in \mathcal{N}'_{i-1}} \{\mathsf{hi}^v(\Phi), \mathsf{lo}^v(\Phi)\}\right) \setminus \{\top, \bot\}$, i.e., we remove the pruned TCs from $\mathcal{N}_i$. Then, from the definition of $\Pr(\Phi)$ (Eq. (6)), we have

$$\Pr(\Phi) = p_i \cdot \sum_{\Phi' \in \mathcal{N}'_{i-1} : \mathsf{hi}(\Phi') = \Phi} \Pr(\Phi') + (1 - p_i) \cdot \sum_{\Phi' \in \mathcal{N}'_{i-1} : \mathsf{lo}(\Phi') = \Phi} \Pr(\Phi'). \tag{7}$$

To compute $\Pr(\top)$, we can use a similar formula:

$$\Pr(\top) = p_i \cdot \sum_{\Phi' \in \mathcal{N}'_1 \cup \cdots \cup \mathcal{N}'_m : \mathsf{hi}(\Phi') = \top} \Pr(\Phi') + (1 - p_i) \cdot \sum_{\Phi' \in \mathcal{N}'_1 \cup \cdots \cup \mathcal{N}'_m : \mathsf{lo}(\Phi') = \top} \Pr(\Phi'). \tag{8}$$

Starting from $\mathcal{N}'_1 = \{\Phi_1^v(\emptyset)\}$, we can generate $\mathcal{N}'_{i+1}$ from $\mathcal{N}'_i$ recursively using transition functions $\mathsf{hi}^v, \mathsf{lo}^v$. By recording the transition, we can compute every $\Pr(\Phi)$ and $\Pr(\top)$ with (7) and (8).

Algorithm 1 formally presents the procedures. Lines 1–7 recursively generate the successive TCs using transition functions $\mathsf{lo}^v, \mathsf{hi}^v$ while executing $\bot, \top$-pruning. This process can be seen as constructing a diagram of TCs as in Figure 3. Every Φ except for $\bot, \top$ has two outgoing edges heading to $\mathsf{lo}^v(\Phi)$ and $\mathsf{hi}^v(\Phi)$. In Algorithm 1, each Φ has two entries $\Phi.\mathsf{lo}$ and $\Phi.\mathsf{hi}$ that store the pointer to $\mathsf{lo}^v(\Phi)$ and $\mathsf{hi}^v(\Phi)$. Here, $\mathcal{N}'_i$ stores the TCs of the subsets of $E_{<i}$ as a hash map; pruned TCs are not stored. Along this diagram, we can compute every $\Pr(\Phi)$ with simple dynamic programming (DP) in lines 8–13; this simulates the formulas (7) and (8). The array $\mathsf{p}[\Phi]$ for $\Phi \in \mathcal{N}'_i$ is prepared to store $\Pr(\Phi)$. Note that even if an identical TC Φ appears in $\mathcal{N}'_i$ and $\mathcal{N}'_j$ $(i \neq j)$, we distinguish $\Phi \in \mathcal{N}'_i$ and $\Phi \in \mathcal{N}'_j$ in computing $\mathsf{p}[\cdot]$. After computing all $\mathsf{p}[\Phi]$, $\Pr(S \rightsquigarrow v) = \Pr(\top)$ is stored in $\mathsf{p}[\top]$.

Next, we analyze the complexity of this algorithm. For each TC Φ, computing $\mathsf{lo}^v(\Phi)$ and $\mathsf{hi}^v(\Phi)$ takes $O((|W_i| + |W_{i+1}|)^2)$ time; details are in Lemma 4.14 of Section 4.4. If every $\mathcal{N}'_i$ is implemented as a hash map, storing or searching a TC also takes $O(|W_i|^2)$ time. The number of patterns on Φ_i^v can be bounded by $O(2^{|W_i|^2})$; since this was not rigorously proved [15], we prove it in Lemma 4.15 of Section 4.4. The overall complexity can be bounded by $O(\sum_i (|W_i| + |W_{i+1}|)^2 \cdot 2^{|W_i|^2}) = O(m\omega^2 \cdot 2^{\omega^2})$, where $\omega = \max_i |W_i|$.

In fact, ω is related to the pathwidth of a graph. Given a path decomposition of G with width ω_p, we can generate an edge ordering with $\omega \leq \omega_p$ in linear time [9]. Since we can construct a path decomposition in linear time for a graph with bounded pathwidth [2], we have a linear time algorithm for computing $\Pr(S \rightsquigarrow v)$ on a graph with bounded pathwidth. However, computing $\sigma(S)$ requires $O(mn\omega^2 \cdot 2^{\omega^2})$ time by computing $\Pr(S \rightsquigarrow v)$ for $v \in V \setminus S$.

4 Linear-Time Influence Spread Computation

We now describe the proposed algorithm for Problem 2.1. First, we explain the high-level concept of the proposed algorithm. In computing $\sigma(S)$ with the algorithm described in Section 3, different diagrams of TCs are generated for different $v \in V \setminus S$ in computing $\Pr(S \rightsquigarrow v)$. To argue the similarities among these diagrams, we divide each of them into three parts: *upper levels*, *lower levels*, and *middle levels*. More specifically, let $A_i \subseteq V$ be the vertices

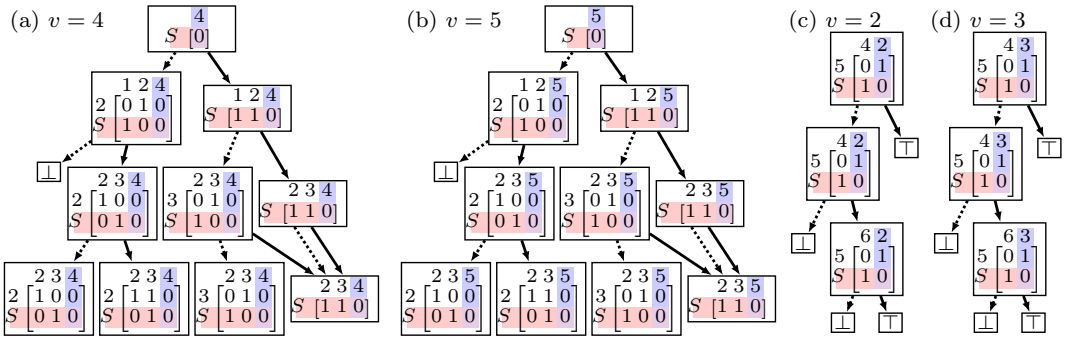


Figure 3 (a)(b) Upper four levels of diagrams constructed from graph in Figure 2a with $v = 4$ and $v = 5$. (c)(d) Parts of lower levels of diagrams constructed from graph in Figure 2a with $v = 2$ and $v = 3$; (c) depicts part below $\Phi_8^2(\{e_2, e_5, e_6\})$ and (d) depicts part below $\Phi_8^3(\{e_1, e_4, e_7\})$. Solid arcs indicate connection from Φ to $\Phi.\mathsf{hi}$, while dashed arcs indicate connection from Φ to $\Phi.\mathsf{lo}$.

that only appear in $E_{<i}$ and do not appear in $E_{\geq i}$, and $B_i \subseteq V$ be those that only appear in $E_{\geq i}$ and do not appear in $E_{<i}$. A_i, B_i, and W_i (Definition 3.1) constitute a partition of V. Moreover, since $E_{<i} \subset E_{<i+1}$ and $E_{\geq i} \supset E_{\geq i+1}$ for $i = 1, \ldots, m$, $V = B_1 \supseteq \cdots \supseteq B_{m+1} = \emptyset$ and $\emptyset = A_1 \subseteq \cdots \subseteq A_{m+1} = V$. Thus, by fixing v, the diagram of TCs $\Phi_i^v(\cdot)$ made from the reachability to v can be divided into three parts according to i: levels i s.t. $v \in B_i$ (upper levels), those s.t. $v \in W_i$ (middle levels), and those s.t. $v \in A_i$ (lower levels).

We first find that upper levels of these diagrams exhibit an identical structure, i.e., diagrams of Φ_i^v and Φ_i^w exhibit an identical structure as long as $v, w \in B_i$ (Observation 4.2); we extract the essential information to build this identical diagram as a *shared transversal configuration* (STC) $\Psi = \Psi_i(E')$. We can also confirm that middle levels of these diagrams were also governed by the STCs (Lemma 4.8). Thus, by considering STCs $\Psi_i(\cdot)$, we can have a similar formula to (5) using the STCs when $v \in B_i \cup W_i$:

$$\Pr(S \rightsquigarrow v) = \sum_{\Psi \in \mathcal{M}_i} \Pr(\Psi) \cdot \Pr(S \rightsquigarrow v \mid \Psi), \tag{9}$$

$$\text{where} \quad \Pr(\Psi) \coloneqq \sum_{E' \subseteq E_{<i} : \Psi_i(E') = \Psi} \Pr_{<i}(E'), \quad \mathcal{M}_i \coloneqq \{\Psi_i(E') \mid E' \subseteq E_{<i}\}.$$

For every v, there exists i such that $v \in B_i \cup W_i$. Thus, for every v, if $\Pr(\Psi)$ and $\Pr(S \rightsquigarrow v \mid \Psi)$ are efficiently computed for all $\Psi = \Psi_i(\cdot)$ with the above i, we can compute $\Pr(S \rightsquigarrow v)$ with (9). Here, since $\Pr(\Psi)$ has a similar recursion as $\Pr(\Phi)$ (Lemma 4.9), all $\Pr(\Psi)$s can be computed in a manner similar to that used in Section 3 by building a diagram of STCs from $i = 1$ to $m + 1$. Since $|\mathcal{M}_i|$ is constant for a graph with bounded pathwidth (Lemma 4.16), it can be performed in $O(m)$ time.

Indeed, when $v \in W_i$ (middle levels), $\Pr(S \rightsquigarrow v \mid \Psi)$ for $\Psi = \Psi_i(\cdot)$ can also be computed efficiently. To compute this, we use the lower levels of the diagram of TCs $\Psi_i^v(\cdot)$. Although naively constructing the diagrams of TCs Φ for every v costs $O(mn)$ time, we find that the lower levels of these diagrams have many identical substructures (Lemma 4.5). By using this, we can build all possible substructures of diagrams that appear in the lower level of these diagrams in $O(m)$ time. In addition, we again use middle levels' equivalence (Lemma 4.8) to compute $\Pr(S \rightsquigarrow v \mid \Psi)$ on middle levels in $O(m)$ time. Here we use the recursion on $\Pr(S \rightsquigarrow v \mid \Psi)$ (Lemmas 4.11 and 4.12).

Eventually, for every v and i where $v \in W_i$, we obtain $\Pr(\Psi)$ and $\Pr(S \rightsquigarrow v \mid \Psi)$ for $\Psi = \Psi_i(\cdot)$. Thus, by choosing i such that $v \in W_i$, we can compute $\Pr(S \rightsquigarrow v)$ with (9). Since (9) contains $O(|\mathcal{M}_i|)$ terms, which is constant for bounded-pathwidth graphs, the

computation of $\Pr(S \rightsquigarrow v)$ for every v costs $O(n)$ time in total. The overall time complexity is bounded by $O(m+n)$ for a graph with bounded pathwidth. Note that, for every v, there exists i such that $v \in W_i$ if v has degree more than 1. For vertices with degree 1, such i might not exist, but we can easily compute $\Pr(S \rightsquigarrow v)$ for such vertices, as described later.

In the following, we first identify the similarities in the previous algorithm [15] for different v in Section 4.1. Then, we derive DP formulas for computing $\Pr(S \rightsquigarrow v)$ in Section 4.2. Finally, we explain the main procedures and the complexity in Sections 4.3 and 4.4.

4.1 Equivalence of Diagrams for Different Vertices

We formally state the equivalence of different diagrams of TCs for upper levels ($v \in B_i$), lower levels ($v \in A_i$), and middle levels ($v \in W_i$).

Upper level equivalence. First, we consider the case $v, w \in B_i$. No edges in $E_{<i}$ are incident to v, meaning that no vertex in W_i can reach v on $(V, E_{<i})$. Furthermore, no vertex in S can reach v on $(V, E_{<i})$. Therefore, for any $E' \subseteq E_{<i}$, $\Phi_i^v(E')_{uv} = 0$ for any u. This means we need to consider the reachability relations among only $W_i \cup \{S\}$, not $W_i \cup \{S, v\}$.

▶ **Definition 4.1.** *Let $\Psi_i(E')$ be a binary matrix indexed by $(W_i \setminus W_i^{S\rightsquigarrow}(E') \cup \{S\}) \times W_i$, where $\Psi_i(E')_{ux} = 1$ if and only if $u \rightsquigarrow_{(V,E')} x$. Here, S is treated as a special (single) row index. We call $\Psi_i(\cdot)$ a* shared transversal configuration *(STC).*

Since $v \in B_i$, we can recover $\Phi_i^v(E')$ from $\Psi_i(E')$ by $\Phi_i^v(E')_{ux} = \Psi_i(E')_{ux}$ for any u as well as $x \neq v$ and $\Phi_i^v(E')_{uv} = 0$ for any u. In other words, the diagram structure is completely determined by the transition of $\Psi_i(\cdot)$ instead of $\Phi_i^v(\cdot)$ when $v \in B_i$. The above argument also holds for the other $w \in B_i$. Therefore, the following observation holds.

▶ **Observation 4.2.** *Consider the diagrams generated using Algorithm 1 with different v, w. If $v, w \in B_i$, the diagrams above any TCs $\Phi_i^v(E')$ and $\Phi_i^w(E')$ are identical. Moreover, we can instead use STC $\Psi_i(\cdot)$ in building the diagram.*

▶ **Example 4.3.** Figures 3a and 3b show the upper four levels of the diagram generated using Algorithm 1 with the graph of Figures 2a and $v = 4$ or $v = 5$. Since $4, 5 \in B_4$, the first four levels are identical, as depicted in these figures.

Lower level equivalence. Next, we consider the case $v, w \in A_i$. Recall that $\Phi_i^v(\cdot)$ captures the reachability among $W_i \cup \{S, v\}$, and $\Phi_i^w(\cdot)$ captures those among $W_i \cup \{S, w\}$. Since $v, w \in A_i$, all edges incident to v and w are in $E_{<i}$, meaning that all incident edges are determined when the edges in $E_{<i}$ are determined. Thus, intuitively, if Φ_i^v and Φ_i^w are "identical," i.e., the reachability relations are identical by substituting v with w, $S \rightsquigarrow v$ and $S \rightsquigarrow w$ can be regarded as equivalent events. We formally define such an equivalence.

▶ **Definition 4.4.** *For $E', F' \subseteq E_{<i}$, consider two TCs $\Phi_i^v(E')$ and $\Phi_i^w(F')$, where $v, w \in A_i$. We say these TCs are* identical *if $\Phi_i^v(E')_{ux} = \Phi_i^w(F')_{ux}$ for any u and $x \neq v, w$ and $\Phi_i^v(E')_{uv} = \Phi_i^w(F')_{uw}$ for any u. This means that $\Phi_i^v(E')$ and $\Phi_i^w(F')$ are identical matrices, except for the column indices being different.*

▶ **Lemma 4.5.** *Suppose $\Phi_i^v(E')$ and $\Phi_i^w(F')$ are identical for $v, w \in A_i$ and $E', F' \subseteq E_{<i}$. Then, for any $H \subseteq E_{\geq i}$, $S \rightsquigarrow_{(V,E' \cup H)} v$ if and only if $S \rightsquigarrow_{(V,F' \cup H)} w$.*

Proof. Fix $H \subseteq E_{\geq i}$ and suppose that $S \rightsquigarrow_{(V,E' \cup H)} v$. There is then a directed walk P from some $s \in S$ to v in $(V, E' \cup H)$. The walk P can be divided into small walks $P = P_0 P_1 P_2 \cdots P_{2k-1} P_{2k}$ satisfying the conditions that P_{2j} consists of only the edges in $E' \subseteq E_{<i}$ and that P_{2j+1} consists of only the edges in $H \subseteq E_{\geq i}$. Let p_j be the contact vertex of P_j and P_{j+1}. Note that P_0 may be empty when $s \in W_i \cup B_i$; in such a case, we set $p_0 = s$. Since p_j $(j \geq 1)$ is incident to both edges in $E' \subseteq E_{<i}$ and those in $H \subseteq E_{\geq i}$, $p_j \in W_i$. If P_0 is nonempty, p_0 is also in W_i. We can also assume that $p_1, p_2, \ldots, p_{2k-2} \notin W_i^{S \rightsquigarrow}(E')$; otherwise, we can take a directed walk $P^\star$ from some $s \in S$ to p_j within (V, E') and consider a walk $P^\star P_{j+1} \cdots P_{2k}$ instead. With a similar argument, we can assume that $p_1, p_2, \ldots, p_{2k-2} \notin W_i^{\rightsquigarrow v}(E')$. Now, we constitute a directed walk P' from some $s' \in S$ to w in $(V, F' \cup H)$. First, when P_0 is not empty, $\Phi_i^v(E')_{Sp_0} = 1$. Since $\Phi_i^v(E')$ and $\Phi_i^w(F')$ are identical, $\Phi_i^w(F')_{Sp_0} = 1$; thus, there is a directed walk P'_0 from some $s' \in S$ to p_0 in (V, F'). This also holds when P_0 is empty; in such a case, $s' = s = p_0$ and P'_0 is also empty. Next, for $1 \leq j \leq k-1$, P_{2j}'s existence and identity indicate $\Phi_i^v(E')_{p_{2j-1}p_{2j}} = 1 = \Phi_i^w(F')_{p_{2j-1}p_{2j}}$, meaning that there is a directed walk P'_{2j} from p_{2j-1} to p_{2j} in (V, F'). Finally, P_{2k}'s existence and identity indicate $\Phi_i^v(E')_{p_{2k-1}v} = 1 = \Phi_i^w(F')_{p_{2k-1}w}$, meaning that there is a directed walk P'_{2k} from p_{2k-1} to w in (V, F'). Now $P' = P'_0 P_1 P'_2 \cdots P_{2k-1} P'_{2k}$ is a directed walk from $s' \in S$ to w in $(V, F' \cup H)$, indicating $S \rightsquigarrow_{(V,F' \cup H)} w$.

In the same manner, we can show the reverse direction. Thus, the lemma holds. ◀

This indicates that $\Pr(S \rightsquigarrow v \mid \Phi_i^v) = \Pr(S \rightsquigarrow w \mid \Phi_i^w)$ when Φ_i^v and Φ_i^w are identical.

▶ **Observation 4.6.** *Consider the diagrams generated using Algorithm 1 with different v, w. If $v, w \in A_i$ and identical TCs Φ_i^v, Φ_i^w appear in each diagram, the diagram structures below Φ_i^v and below Φ_i^w are identical.*

▶ **Example 4.7.** Figures 3c and 3d show parts of the last four levels of the diagrams below $\Phi_8^2(\{e_2, e_5, e_6\})$ $(v = 2)$ and below $\Phi_8^3(\{e_1, e_4, e_7\})$ $(v = 3)$. Since $2, 3 \in A_8$ and these TCs are identical, these diagram structures are also identical.

Middle level equivalence. Finally, we examine the case $v, w \in W_i$. In this case, the TC $\Phi_i^v(\cdot)$ captures the reachability among $W_i \cup \{S, v\} = W_i \cup \{S\}$. In addition, the STC $\Psi_i(\cdot)$ also captures the reachability among $W_i \cup \{S\}$. Thus, we can recover $\Phi_i^v(E')$ from $\Psi_i(E')$ by removing the columns corresponding to $W_i^{\rightsquigarrow v}(E')$. In other words, if $\Psi_i(E') = \Psi_i(F')$, then $E', F' \subseteq E_{<i}$ satisfy condition (#).

For $\Psi = \Psi_i(E')$, let $W_i^{S \rightsquigarrow}(\Psi)$ be the set of vertices in W_i that can be reached from S under Ψ, which is determined by the missing row indices of Ψ. Consider a graph $G_i(\Psi) \coloneqq (W_i \setminus W_i^{S \rightsquigarrow}(\Psi), E_i(\Psi))$, where $E_i(\Psi) \coloneqq \{(u, x) \mid u, x \in W_i \setminus W_i^{S \rightsquigarrow}(\Psi), \Psi_{ux} = 1\}$. In other words, when $\Psi = \Psi_i(E')$, $G_i(\Psi)$ is the subgraph of the transitive closure of (V, E') induced by $W_i \setminus W_i^{S \rightsquigarrow}(\Psi)$. Then, we can prove the following.

▶ **Lemma 4.8.** *Suppose $\Psi_i(E') = \Psi_i(F') =: \Psi$ for $E', F' \subseteq E_{<i}$ and $v, w \in W_i \setminus W_i^{S \rightsquigarrow}(E')$ are in the same SCC of $G_i(\Psi)$. Then, for any $H \subseteq E_{\geq i}$, $S \rightsquigarrow_{(V,E' \cup H)} v$ if and only if $S \rightsquigarrow_{(V,F' \cup H)} w$.*

Proof. Fix $H \subseteq E_{\geq i}$. We can prove the equivalence of $S \rightsquigarrow_{(V,E' \cup H)} v$ and $S \rightsquigarrow_{(V,F' \cup H)} v$ from $\Psi_i(E') = \Psi_i(F')$ with almost the same argument as that in the proof of Lemma 4.5. Since v and w are in the same SCC of $G_i(\Psi)$, $v \rightsquigarrow_{(V,F')} w$ and $w \rightsquigarrow_{(V,F')} v$. Consequently, $S \rightsquigarrow_{(V,F' \cup H)} v$ indicates $S \rightsquigarrow_{(V,F' \cup H)} w$ and vice versa. This concludes $S \rightsquigarrow_{(V,E' \cup H)} v \iff S \rightsquigarrow_{(V,F' \cup H)} w$. ◀

4.2 New Decomposition and DP Formulas

Decomposition formula. Using the equivalences described above, we derive a formula for $\Pr(S \rightsquigarrow v)$ using the STCs. Since we confirmed that $\Phi_i^v(E')$ can be recovered from $\Psi_i(E')$ when $v \in A_i \cup W_i$, we can derive the decomposition formula (9). If $v \in W_i$, it can be further transformed. Based on Lemma 4.8, we can rewrite $\Pr(S \rightsquigarrow v \mid \Psi)$ as $\Pr(S \rightsquigarrow C_\Psi(v) \mid \Psi)$, where $C_\Psi(v)$ is the SCC of $G_i(\Psi)$ containing v when $v \notin W_i^{S\rightsquigarrow}(\Psi)$. If $v \in W_i^{S\rightsquigarrow}(\Psi)$, we immediately have $\Pr(S \rightsquigarrow v \mid \Psi) = 1$. Now (9) can be further transformed into

$$\Pr(S \rightsquigarrow v) = \sum_{\Psi \in \mathcal{M}_i} \begin{cases} \Pr(\Psi) & (v \in W_i^{S\rightsquigarrow}(\Psi)) \\ \Pr(\Psi) \cdot \Pr(S \rightsquigarrow C_\Psi(v) \mid \Psi) & (v \in W_i \setminus W_i^{S\rightsquigarrow}(\Psi)) \end{cases}. \tag{10}$$

If we can obtain $\Pr(\Psi)$ and $\Pr(S \rightsquigarrow C \mid \Psi)$ for every STC Ψ and every SCC C of $G_i(\Psi)$, we can compute every $\Pr(S \rightsquigarrow v)$ by choosing i such that $v \in W_i$ and then use (10). Note that if all vertices have degree not less than 2, there must exist index i such that $v \in W_i$, since G contains no self-loops. The $\Pr(S \rightsquigarrow v)$ value for degree 1 vertices can be easily obtained; details are given in Section 4.3.

DP formulas. Since TCs can be recovered from STCs and admit transition functions $\mathsf{hi}^v, \mathsf{lo}^v$, STCs also admit transition functions lo and hi satisfying that, for any $E' \subseteq E_{<i}$, $\mathsf{lo}(\Psi_i(E')) = \Psi_{i+1}(E')$ and $\mathsf{hi}(\Psi_i(E')) = \Psi_{i+1}(E' \cup \{e_i\})$. Here, we set $\mathsf{f}(\Psi) = \bot$ for $\mathsf{f} \in \{\mathsf{hi}, \mathsf{lo}\}$ when $i \geq i_S$ and $\mathsf{f}(\Psi)_{Su} = 0$ for any u, which confirms $\Pr(S \rightsquigarrow w \mid \mathsf{f}(\Psi)) = 0$ for any further vertices $w \in W_{i+1} \cup B_{i+1}$ ($\bot$-pruning). Now $\Pr(\Psi)$ can be computed in the same manner as described in Section 3: From the definition of $\Pr(\Psi)$ (Eq. (9)), we have the following.

▶ **Lemma 4.9.** *For $\Psi = \Psi_i(E')$ for some $E' \subseteq E_{<i}$, we have*

$$\Pr(\Psi) = p_i \cdot \sum_{\Psi' \in \mathcal{M}'_{i-1} : \mathsf{hi}(\Psi') = \Psi} \Pr(\Psi') + (1 - p_i) \cdot \sum_{\Psi' \in \mathcal{M}'_{i-1} : \mathsf{lo}(\Psi') = \Psi} \Pr(\Psi'), \tag{11}$$

where $\mathcal{M}'_1 := \{\Psi_1(\emptyset)\}$ and $\mathcal{M}'_i := \left(\bigcup_{\Psi \in \mathcal{M}'_{i-1}} \{\mathsf{hi}(\Psi), \mathsf{lo}(\Psi)\}\right) \setminus \{\bot\}$, i.e., we remove pruned STCs from $\mathcal{M}_i$.

It remains to be explained how to compute $\Pr(S \rightsquigarrow C \mid \Psi)$ for an SCC C of $G_i(\Psi)$. If there exists $u \in C$ such that $u \in W_{i+1}$, we can decompose it using the case analysis of e_i: If e_i is present, $\Pr(S \rightsquigarrow C \mid \Psi)$ equals $\Pr(S \rightsquigarrow C' \mid \mathsf{hi}(\Psi))$, where C' is the SCC of $G_{i+1}(\mathsf{hi}(\Psi))$ containing u. If e_i is absent, $\Pr(S \rightsquigarrow C \mid \Psi)$ equals $\Pr(S \rightsquigarrow C'' \mid \mathsf{lo}(\Psi))$, where C'' is the SCC of $G_{i+1}(\mathsf{lo}(\Psi))$ containing u. Here, the relation between C and C', C'' can be seen as the transition from C to successive SCCs C', C''. By considering the pruned cases, we formally define the successive SCCs as follows.

▶ **Definition 4.10.** *For SCC C of $G_i(\Psi)$, we define* successive SCC $\mathsf{f}(C)$ *for $\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}$ as follows. (i) If $\mathsf{f} = \mathsf{hi}$, and $e_i = (u, v)$ with $u \in W_i^{S\rightsquigarrow}(\Psi) \cup S$ and $v \in C$, $\mathsf{hi}(C) := \top$, meaning that C can be reached from S by determining the presence of e_i. (ii) If (i) does not hold and $\mathsf{f}(\Psi) = \bot$, $\mathsf{f}(C) := \bot$, meaning that C cannot be reached from S. (iii) If neither (i) nor (ii) holds and there exists $u \in C$ such that $u \in W_{i+1}$, $\mathsf{f}(C) := C_{\mathsf{f}(\Psi)}(u)$. (iv) Otherwise, $\mathsf{f}(C) := \emptyset$, meaning that there is no successive SCC.*

▶ **Lemma 4.11.** *If there exists $u \in C$ such that $u \in W_{i+1}$, i.e., successive SCCs exist, we can decompose $\Pr(S \rightsquigarrow C \mid \Psi)$ for $\Psi = \Psi_i(E')$ and SCC C of $G_i(\Psi)$ as follows:*

$$\Pr(S \rightsquigarrow C \mid \Psi) = p_{e_i} \cdot \Pr(S \rightsquigarrow \mathsf{hi}(C) \mid \mathsf{hi}(\Psi)) + (1 - p_{e_i}) \cdot \Pr(S \rightsquigarrow \mathsf{lo}(C) \mid \mathsf{lo}(\Psi)). \tag{12}$$

Note that if $u \in W_{i+1}^{S\rightsquigarrow}(\mathsf{hi}(\Psi))$, since $\Pr(S \rightsquigarrow u \mid \mathsf{hi}(\Psi)) = 1$, we replace $\Pr(S \rightsquigarrow C_{\mathsf{hi}(\Psi)}(u) \mid \mathsf{hi}(\Psi))$ in the above formula with 1. This replacement is reflected in case (i) of Definition 4.10.

What can be done when such a vertex u does not exist? In this case, we attempt to use the original TC. We observe in Section 4.1 that if $u \in W_i$, we can recover $\Phi = \Phi_i^u(E')$ from $\Psi_i(E')$ with an appropriate transformation. Thus, by fixing some $u \in C$ and letting Φ^u be a TC recovered from STC Ψ, we have the following case analysis: If e_i is present, $\Pr(S \rightsquigarrow C \mid \Psi) = \Pr(S \rightsquigarrow u \mid \Phi^u)$ equals $\Pr(S \rightsquigarrow u \mid \mathsf{hi}^u(\Phi^u))$. If e_i is absent, $\Pr(S \rightsquigarrow C \mid \Psi)$ equals $\Pr(S \rightsquigarrow u \mid \mathsf{lo}^u(\Phi^u))$. This leads to the following decomposition.

▶ **Lemma 4.12.** *For $\Psi = \Psi_i(E')$ and SCC C of $G_i(\Psi)$, suppose that successive SCCs do not exist. Let $u \in C$ and Φ^u be a TC recovered from STC Ψ. Then,*

$$\Pr(S \rightsquigarrow C \mid \Psi) = p_{e_i} \cdot \Pr(S \rightsquigarrow u \mid \mathsf{hi}^u(\Phi^u)) + (1 - p_{e_i}) \cdot \Pr(S \rightsquigarrow u \mid \mathsf{lo}^u(\Phi^u)). \tag{13}$$

To compute the terms in the right-hand-side of (13), we can use the previous algorithm in Section 3. However, naively computing these terms for every $u \in V$ requires $O(mn\omega_p^2 \cdot 2^{\omega_p^2})$ time, since this amounts to constructing a diagram of the previous algorithm for every u.

To overcome this, we use Lemma 4.5. Since we only need to compute $\Pr(S \rightsquigarrow v \mid \Phi^v)$ for the case $v \in A_i$, we can share identical TCs, as in Observation 4.6. Accordingly, we do not need to construct different diagrams for every v to compute $\Pr(S \rightsquigarrow v \mid \Phi^v)$, that is, if identical TCs Φ^v and Φ^w from different v, w appear, we share them. This enables us to compute all $\Pr(S \rightsquigarrow v \mid \Phi^v)$s in linear time.

4.3 Procedure

We formally describe the proposed method in Algorithm 2. Lines 1–6 construct the diagram of STCs Ψ. $\Psi.\mathsf{f}$ stores a pointer to $\mathsf{f}(\Psi)$. We consider the successive SCCs in lines 7–11. If a successive SCC of C exists, $C.\mathsf{f}$ stores the pointer to it. Otherwise, following the approach of Section 4.2, we recover the original TC Φ^u, where $u \in C$, and then carry out transition lo^u or hi^u on it. $\Phi^u(\Psi)$ recovers the original TC by removing every column x satisfying $\Psi_{xu} = 1$ from Ψ, that is, we remove columns whose indices can reach u. If a pruning occurs by considering $\Phi = \mathsf{f}^u(\Phi^u(\Psi))$, $C.\mathsf{f}$ stores $\bot$ or $\top$. Otherwise, $C.\mathsf{f}$ stores a pointer to $\mathsf{change}^t(\Phi)$, where $\mathsf{change}^t(\cdot)$ changes the column index u to t. Here, t is an imaginary vertex that is assumed to be contained within A_i. Since Observation 4.6 confirms that the transitions of identical TCs are identical, we standardize the column index u by changing it to t. We build a diagram of $\Phi_i^t(\cdot)$ in Lines 12–15.

Lines 17–19 compute $\mathsf{p}[\Psi] = \Pr(\Psi)$ for every $\Psi \in \mathcal{M}_i'$ in the same manner as Algorithm 1, and lines 20–22 compute $\mathsf{q}[\Phi] = \Pr(S \rightsquigarrow t \mid \Phi)$ for every $\Phi \in \mathcal{N}_i'$ in a bottom-up manner. Lines 23–26 compute $\mathsf{r}[\Psi, C] = \Pr(S \rightsquigarrow C \mid \Psi)$ for every Ψ and every SCC C of $G_i(\Psi)$. Here, $\langle c \ ? \ x : y \rangle$ stands for a conditional operator that returns x if c is satisfied or y otherwise. When $C.\mathsf{f}$ is either an original TC Φ^t or $\bot, \top$, we use the q value; otherwise, we use the r value. Finally, lines 27–31 compute every $\mathsf{res}[v] = \Pr(S \rightsquigarrow v)$ using (10).

Here, we confirm the correctness of Algorithm 2.

▶ **Lemma 4.13.** *After executing Algorithm 2, $\mathsf{res}[v]$ equals $\Pr(S \rightsquigarrow v)$.*

Proof. First, we confirm $\mathsf{p}[\Psi] = \Pr(\Psi)$. Starting from $\mathsf{p}[\Psi_1(\emptyset)] = \Pr(\Psi_1(\emptyset)) = 1$ for $\Psi_1(\emptyset) \in \mathcal{M}_1'$, lines 17–19 compute every $\Pr(\Psi)$ value for $\mathcal{M}_2', \ldots, \mathcal{M}_m'$ according to (11).

Second, we confirm $\mathsf{q}[\Phi] = \Pr(S \rightsquigarrow t \mid \Phi)$, but this is much simpler; with the case analysis of whether e_i is present or absent, we have

$$\Pr(S \rightsquigarrow t \mid \Phi) = p_i \cdot \Pr(S \rightsquigarrow t \mid \mathsf{hi}^t(\Phi)) + (1 - p_i) \cdot \Pr(S \rightsquigarrow t \mid \mathsf{lo}^t(\Phi)). \tag{14}$$

Algorithm 2 Proposed algorithm for computing $\Pr(S \rightsquigarrow v)$ for every v.

```
1  M'_1 ← {Ψ_1(∅)}, M'_i ← {} (i ≥ 2), N'_i ← {} (i ≥ 1)
2  for i ← 1 to m do                                          // Diagram construction for Ψ
3      foreach Ψ ∈ M'_i do
4          foreach f ∈ {lo, hi} do
5              if f(Ψ) = ⊥ then Ψ.f ← ⊥
6              else M'_{i+1} ← M'_{i+1} ∪ {f(Ψ)}, Ψ.f ← f(Ψ) ∈ M'_{i+1}
7              foreach C: SCC of G_i(Ψ) do                    // Compute successive SCCs
8                  if f(C) ≠ ∅ then C.f ← f(C)                // Including the cases f(C) = ⊥, ⊤
9                  else                                       // Choose u ∈ C arbitrarily
10                     if Φ ← f^u(Φ^u(Ψ)) = ⊥ or ⊤ then C.f ← ⊥ or ⊤
11                 else N'_{i+1} ← N'_{i+1} ∪ {change^t(Φ)}, C.f ← change^t(Φ) ∈ N'_{i+1}
12     foreach Φ ∈ N'_i do                                    // Diagram construction for Φ^t
13         foreach f ∈ {lo, hi} do
14             if f^t(Φ) = ⊥ or ⊤ then Φ.f ← ⊥ or ⊤
15             else N'_{i+1} ← N'_{i+1} ∪ {f^t(Φ)}, Φ.f ← f^t(Φ) ∈ N'_{i+1}
16 p[Ψ] ← 1 for Ψ ∈ M'_1, p[Ψ] = 0 for Ψ ∈ M'_i (i ≥ 2), q[⊥] ← 0, q[⊤] ← 1
17 for i ← 1 to m do                                          // Top-down DP
18     foreach Ψ ∈ M'_i do                                    // p[Ψ] stores Pr(Ψ)
19         p[Ψ.lo] += (1 − p_{e_i}) · p[Ψ], p[Ψ.hi] += p_{e_i} · p[Ψ]
20 for i ← m to 1 do                                          // Bottom-up DP
21     foreach Φ ∈ N'_i do                                    // q[Φ] stores Pr(S ⇝ t | Φ)
22         q[Φ] ← (1 − p_{e_i}) · q[Φ.lo] + p_{e_i} · q[Φ.hi]
23     foreach Ψ ∈ M'_i do
24         foreach C: SCC of G_i(Ψ) do                        // r[Ψ, C] stores Pr(S ⇝ C | Ψ)
25             r[Ψ, C] ← (1 − p_{e_i}) · ⟨lo(C) ∈ {⊤, ⊥, ∅} ? q[C.lo] : r[Ψ.lo, C.lo]⟩
26                       + p_{e_i} · ⟨hi(C) ∈ {⊤, ⊥, ∅} ? q[C.hi] : r[Ψ.hi, C.hi]⟩
27 foreach v ∈ V \ S do                                       // Compute Pr(S ⇝ v) by using Eq. (10)
28     res[v] ← 0                                             // res[v] stores Pr(S ⇝ v)
29     foreach Ψ ∈ M'_{i*} do                                 // Choose i* arbitrarily s.t. v ∈ W_{i*}
30         res[v] += p[Ψ] · ⟨v ∈ W_{i*}^{S⇝}(Ψ) ? 1 : r[Ψ, C_Ψ(v)]⟩
31 return res
```

Starting from $\mathsf{q}[\bot] = \Pr(S \rightsquigarrow t \mid \bot) = 0$ and $\mathsf{q}[\top] = \Pr(S \rightsquigarrow t \mid \top) = 1$, lines 20–22 compute every $\Pr(S \rightsquigarrow t \mid \Phi)$ value for $\mathcal{N}'_m, \mathcal{N}'_{m-1}, \ldots$ according to (14).

Third, we confirm $\mathsf{r}[\Psi, C] = \Pr(S \rightsquigarrow C \mid \Psi)$. If there exists $u \in C$ such that $u \in W_{i+1}$, i.e., a successive SCC exists, we can apply (12). Otherwise (i.e., a successive SCC does not exist), we can apply (13). Here, without considering the pruning, $C.\mathsf{f}$ stores a pointer to $\mathsf{f}(C)$ if a successive SCC exists or, otherwise, to $\mathsf{change}^t(\mathsf{lo}^u(\Phi^u))$. Consequently, lines 23–26 correctly compute $\mathsf{r}[\Psi, C]$ according to (12) if a successive SCC exists or to (13) otherwise. For the latter case, we use Lemma 4.5; i.e., $\Pr(S \rightsquigarrow u \mid \mathsf{lo}^u(\Phi^u)) = \Pr(S \rightsquigarrow t \mid \mathsf{change}^t(\mathsf{lo}^u(\Phi^u))) = \mathsf{q}[\mathsf{change}^t(\mathsf{lo}^u(\Phi^u))]$.

Finally, by choosing $i^\star$ such that $v \in W_{i^\star}$, $\mathsf{res}[v] = \Pr(S \rightsquigarrow v)$ is correctly computed using (10) through lines 27–30. ◀

Treatment of degree 1 vertices. We finally describe how to deal with degree 1 vertices. By obtaining the $\Pr(S \rightsquigarrow v)$ values for all vertices with degree not less than 2, we can easily obtain the $\Pr(S \rightsquigarrow v)$ values for degree 1 vertices. Let $v \in V \setminus S$ be a vertex whose degree is 1. Let e_v be the unique edge incident to v. If $e_v = (v, w)$ for some $w \in V$, i.e., v is the tail of e_v, we have $\Pr(S \rightsquigarrow v) = 0$ because v has no incoming edges. Otherwise (i.e., v is the head of e_v), we have $\Pr(S \rightsquigarrow v) = p_{e_v} \cdot \Pr(S \rightsquigarrow w)$ because every path from some $s \in S$ to v must go through w, thus e_v. Since G is connected and thus w has degree at least 2 except for the trivial case where e_v is the only edge of G, we can obtain $\Pr(S \rightsquigarrow v)$ for v whose degree is 1 from the value of $\Pr(S \rightsquigarrow w)$.

4.4 Complexity

To analyze the complexity, we again use $\omega = \max_i |W_i|$. First, we consider the diagram construction. The key lemma is as follows.

▶ **Lemma 4.14.** *Given $\Phi = \Phi_i^v(E')$ for some $E' \subseteq E_{<i}$, we can compute the transitions $\mathsf{lo}^v(\Phi)$ and $\mathsf{hi}^v(\Phi)$ in $O((|W_i| + |W_{i+1}|)^2)$ time. Similarly, given $\Psi = \Psi_i(E')$ for some $E' \subseteq E_{<i}$, we can also compute $\mathsf{lo}(\Psi)$ and $\mathsf{hi}(\Psi)$ in $O((|W_i| + |W_{i+1}|)^2)$ time.*

The full proof is deferred to the full version; here, we describe the ideas for computing transitions. For TC Φ, we build a graph $G^\star = (W_i \cup \{S, v\}, E^\star)$, where S is regarded as an individual special vertex and $E^\star = \{(u, w) \mid \Phi_{uw} = 1\}$. By adding and removing edges and vertices according to the transition and computing the transitive closure, we can obtain the updated graph, whose adjacency matrix constitutes $\mathsf{hi}(\Phi)$ or $\mathsf{lo}(\Phi)$. The transition of STC Ψ can be computed in very similar way.

Similar to Φ_i^v, we can compute the transition lo, hi of Ψ in $O(\omega^2)$ by recomputing the transitive closure. Note that we can also compute the SCCs and their successive SCCs within this process, as described in the proof of Lemma 4.14. Thus, the total time for diagram construction is bounded by $O(\omega^2) \cdot O(\sum_i (|\mathcal{M}_i| + |\mathcal{N}_i|))$ time. Here, we can bound the number of possible patterns on Φ_i^v and Ψ_i, i.e., $|\mathcal{N}_i|$ and $|\mathcal{M}_i|$, as follows. Since the proofs of Lemmas 4.15 and 4.16 are much similar; the proof of Lemma 4.16 is put on the full version.

▶ **Lemma 4.15.** *$|\mathcal{M}_i| = |\{\Phi_i^v(E') \mid E' \subseteq E_{<i}\}|$ is bounded by $O(2^{|W_i|^2})$.*

Proof. If we fix $W_i^{S\rightsquigarrow}(E')$ and $W_i^{\rightsquigarrow v}(E')$, some entries of $\Phi_i^v(E')$ are determined as follows: $\Phi_i^v(E')_{Sw} = 1$ if $w \in W_i^{S\rightsquigarrow}(E')$ or 0 if $w \in W_i \setminus W_i^{S\rightsquigarrow}(E')$, and $\Phi_i^v(E')_{uv} = 1$ if $u \in W_i^{\rightsquigarrow v}(E')$ or 0 if $u \in W_i \setminus W_i^{\rightsquigarrow v}(E')$. Therefore, the possible patterns on $\Phi_i^v(E')$ is bounded by $O(2^{(|W_i| - |W_i^{S\rightsquigarrow}(E')|)(|W_i| - |W_i^{\rightsquigarrow v}(E')|)})$ because $\Phi_i^v(E')$ is a binary matrix of size $(|W_i| - |W_i^{S\rightsquigarrow}(E')| + 1) \times (|W_i| - |W_i^{\rightsquigarrow v}(E')| + 1)$, and the entries of one row and one column are already determined.

We conduct a case analysis of $k = |W_i^{S\rightsquigarrow}(E')| + |W_i^{\rightsquigarrow v}(E')|$ when fixing $W_i^{S\rightsquigarrow}(E')$ and $W_i^{\rightsquigarrow v}(E')$. When $k = 0$, the bound is $O(2^{|W_i|^2})$. When $k = 1$, the bound is $O(2^{|W_i|^2 - |W_i|})$, and the number of patterns on $(W_i^{S\rightsquigarrow}(E'), W_i^{\rightsquigarrow v}(E'))$ is bounded by $O(|W_i|)$. When $k \geq 2$, since $(|W_i| - |W_i^{S\rightsquigarrow}(E')|)(|W_i| - |W_i^{\rightsquigarrow v}(E')|) \leq (|W_i| - 1)(|W_i| - 1) = |W_i|^2 - 2|W_i| + 1$ (supposing that $|W_i| \geq 3$), the number of possible patterns is bounded by $O(2^{|W_i|^2 - 2|W_i|})$. The number of possible patterns on $(W_i^{S\rightsquigarrow}(E'), W_i^{\rightsquigarrow v}(E'))$ is bounded by $2^{|W_i|} \cdot 2^{|W_i|}$ because $W_i^{S\rightsquigarrow}, W_i^{\rightsquigarrow v} \subseteq W_i$. Thus, the total number of patterns on $\Phi_i^v(E')$ is bounded by $O(2^{|W_i|^2}) + O(|W_i| \cdot 2^{|W_i|^2 - |W_i|}) + O(2^{2|W_i|}) \cdot O(2^{|W_i|^2 - 2|W_i|}) = O(2^{|W_i|^2}) + O(2^{|W_i|^2 + \log|W_i| - |W_i|}) + O(2^{|W_i|^2}) = O(2^{|W_i|^2})$. ◀

▶ **Lemma 4.16.** *$|\mathcal{N}_i| = |\{\Psi_i(E') \mid E' \subseteq E_{<i}\}|$ is bounded by $O(2^{|W_i|^2})$.*

Therefore, $|\mathcal{M}_i| = |\mathcal{N}_i| = O(2^{\omega^2})$, and the total time complexity for diagram construction can be bounded by $O(m\omega^2 \cdot 2^{\omega^2})$.

Next, we consider the DP. Each value can be computed in constant time. The number of values to compute for Ψ_i and Φ_i^v is bounded by $O(|\mathcal{M}_i| + |W_i||\mathcal{M}_i| + |\mathcal{N}_i|) = O(\omega \cdot 2^{\omega^2} + 2^{\omega^2}) = O(\omega \cdot 2^{\omega^2})$. Thus, the total complexity is $O(m\omega \cdot 2^{\omega^2})$.

Finally, computing each $\Pr(S \rightsquigarrow v)$ using (10) requires $O(|\mathcal{M}_i|) = O(2^{\omega^2})$ time. Thus, it needs $O(n \cdot 2^{\omega^2})$ time in total for all $\Pr(S \rightsquigarrow v)$ values.

▶ **Proposition 4.17.** *Given directed graph G, seed set S, each edge's probability p_e of presence, and the edge ordering $e_1, \ldots, e_m$, the probability $\Pr(S \rightsquigarrow v)$ for every vertex v can be computed in $O((m + n) \cdot \omega^2 \cdot 2^{\omega^2})$ time in total.*

Since we can generate an edge ordering with $\omega \leq \omega_p$ from a path decomposition of width ω_p [9] in linear time, we achieve Theorem 1.2. Moreover, with the linear-time path-decomposition algorithm for a bounded-pathwidth graph [2], we have the following.

▶ **Theorem 4.18.** *Given directed graph G, seed set S, and each edge's probability p_e of presence, suppose that G's pathwidth is bounded by a constant. Then, $\Pr(S \rightsquigarrow v)$ for every vertex v can be computed in $O(m+n)$ time in total.*

The previous algorithm [15] requires $O(m\omega^2 \cdot 2^{\omega^2})$ time for computing $\Pr(S \rightsquigarrow v)$, as discussed in Section 3. Thus, it takes $O(nm\omega^2 \cdot 2^{\omega^2})$ time for computing $\sigma(S)$. In contrast, the proposed algorithm takes $O((n+m)\omega^2 \cdot 2^{\omega^2})$ time. Therefore, in terms of time complexity, our algorithm reduces the dependence on the graph size from $O(mn)$ to $O(m+n)$ while the dependence on $\omega = \max_i |W_i|$ is kept the same.

5 Discussion

Here, we point out the differences between the proposed algorithm and the existing algorithm that computes the probability of connection for undirected graphs [16]. These algorithms basically share the same idea: In computing the probability of reachability (or connectivity in the undirected graphs) of every vertex, the previous top-down style algorithms, like that in Section 3, generate similar diagrams for different vertices v, so they attempt to share equivalent structures. The algorithm for undirected graphs uses connected components (CCs) among frontier vertices to efficiently compute the conditional probabilities, similar to $\Pr(S \rightsquigarrow C \mid \Psi)$ in our algorithm. The difference comes from the computation of conditional probability when a successive SCC or CC does not exist. In undirected graphs, if a successive CC does not exist, the current CC is disconnected from any further vertex, so we immediately have the conditional probability 0 (or 1 in an exceptional case). In contrast, in directed graphs, even if a successive SCC does not exist, the current SCC may still reach or be reached from further vertices because it can reach or be reached from the frontier vertices. This makes the computation of the conditional probabilities difficult because we cannot immediately determine them when no successive SCC exists. We resolve this issue by focusing on the equivalence of lower levels of diagrams (Lemma 4.5). This enables us to compute the conditional probabilities using a constant number of diagrams (STCs Ψ and TCs Φ^t of imaginary vertex t) without breaking the linear complexity.

6 Conclusion

We proposed an algorithm for computing the influence spread under IC model exactly in $O((m+n)\omega_p^2 \cdot 2^{\omega_p^2})$ time. For graphs with bounded pathwidth ω_p, we improved the complexity from $O(mn)$ to $O(m+n)$, linear in the graph size, while the dependence on the pathwidth ω_p is kept the same.

Finally, we mention some future directions. First, a more efficient exact evaluation of influence spread, suitable for the greedy algorithm [10], should be investigated. Chen et al. [4] developed an algorithm that simultaneously approximates $\sigma(S \cup \{v\}) - \sigma(S)$ for all $v \in V \setminus S$, which corresponds to one iteration of the greedy algorithm. Nakamura et al. [17] developed an algorithm for undirected graphs that exactly and simultaneously computes the expected number of connected vertices for all vertices. This algorithm is promising for adoption with directed graphs as a way to simultaneously compute $\sigma(S \cup \{v\})$ for all $v \in V \setminus S$ exactly. Second, we should investigate whether a linear time algorithm for computing $\sigma(S)$ is possible for graphs with bounded *treewidth*. A roadmap for achieving this may be to fit

the diagram construction method based on tree decomposition [1] to the computation of influence spread and then observe the equivalences. However, such an improvement may not be straightforward due to the complicated nature of these algorithms.

References

1 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *Proc. of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 111:1–111:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.111.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2010)*, pages 1029–1038, 2010. doi:10.1145/1835804.1835934.

4 Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009)*, pages 199–208, 2009. doi:10.1145/1557019.1557047.

5 Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proc. of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001)*, pages 57–66, 2001. doi:10.1145/502512.502525.

6 Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12:211–223, 2001. doi:10.1023/A:1011122126881.

7 Amit Goyal, Wei Lu, and Laks V.S. Lakshmanan. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In *Proc. of the 20th International Conference Companion on World Wide Web (WWW 2011)*, pages 47–48, 2011. doi:10.1145/1963192.1963217.

8 Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-terminal network reliability measures with binary decision diagrams. *IEEE Transactions on Reliability*, 56(3):506–515, 2007. doi:10.1109/TR.2007.898572.

9 Yuma Inoue and Shin-ichi Minato. Acceleration of ZDD construction for subgraph enumeration via pathwidth optimization. Technical Report TCS-TR-A-16-80, Division of Computer Science, Hokkaido University, 2016.

10 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*, pages 137–146, 2003. doi:10.1145/956750.956769.

11 Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007)*, pages 420–429, 2007. doi:10.1145/1281192.1281239.

12 Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018. doi:10.1109/TKDE.2018.2807843.

13 Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. Real-time targeted influence maximization for online advertisements. *Proc. of the VLDB Endowment*, pages 1070–1081, 2015. doi:10.14778/2794367.2794376.

14 Chen Ling, Junji Jiang, Junxiang Wang, My T. Thai, Renhao Xue, James Song, Meikang Qiu, and Liang Zhao. Deep graph representation learning and optimization for influence maximization. In *Proc. of the 40th International Conference on Machine Learning (ICML 2023)*, volume 202 of *Proceedings of Machine Learning Research*, pages 21350–21361, 2023. URL: https://proceedings.mlr.press/v202/ling23b.html.

15 Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact computation of influence spread by binary decision diagrams. In *Proc. of the 26th International Conference on World Wide Web (WWW 2017)*, pages 947–956, 2017. doi:10.1145/3038912.3052567.

16 Kengo Nakamura, Takeru Inoue, Masaaki Nishino, and Norihito Yasuda. Efficient network reliability evaluation for client-server model. In *Proc. of the 2021 IEEE Global Communications Conference (GLOBECOM 2021)*, pages 1–6, 2021. doi:10.1109/GLOBECOM46510.2021.9685283.

17 Kengo Nakamura, Takeru Inoue, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato. A fast and exact evaluation algorithm for the expected number of connected nodes: an enhanced network reliability measure. In *Proc. of the 2023 IEEE International Conference on Computer Communications (INFOCOM 2023)*, pages 1–10, 2023. doi:10.1109/INFOCOM53939.2023.10228897.

18 Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *Proc. of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 138–144, 2014. doi:10.1609/aaai.v28i1.8726.

19 Shashank Sheshar Singh, Divya Srivastva, Madhushi Verma, and Jagendra Singh. Influence maximization frameworks, performance, challenges and directions on social network: A theoretical study. *Journal of King Saud University - Computer and Information Sciences*, 34(9):7570–7603, 2022. doi:10.1016/j.jksuci.2021.08.009.

20 Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Exact computation of strongly connected reliability by binary decision diagrams. In *Proc. of the 12th International Conference on Combinatorial Optimization and Applications (COCOA 2018)*, pages 281–295, 2018. doi:10.1007/978-3-030-04651-4_19.

21 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

22 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi:10.1137/0208032.

23 Peng Wu and Li Pan. Scalable influence blocking maximization in social networks under competitive independent cascade models. *Computer Networks*, 123:38–50, 2017. doi:10.1016/j.comnet.2017.05.004.

24 Mao Ye, Xingjie Liu, and Wang-Chien Lee. Exploring social influence for recommendation: a generative model approach. In *Proc. of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2012)*, pages 671–680, 2012. doi:10.1145/2348283.2348373.

Path-Reporting Distance Oracles for Vertex-Labeled Graphs

Ofer Neiman
Faculty of Computer and Information Science, Ben-Gurion University of the Negev

Alon Spector
Faculty of Computer and Information Science, Ben-Gurion University of the Negev

Abstract

Let $G = (V, E)$ be a weighted undirected graph, with n vertices. A distance oracle is a data structure that can quickly answer distance queries, with some stretch factor. A seminal work of [16], given an integer $k \geq 1$, provides such an oracle with stretch $2k - 1$, query time $O(k)$, and size $O(k \cdot n^{1+1/k})$. Furthermore, this oracle can also report a path in G corresponding to the returned distance.

In this paper we focus on vertex-labeled graphs, in which each vertex is given a label from a set L of size ℓ. A *vertex-label distance oracle* answers queries of the form (v, λ), where $v \in V$ and $\lambda \in L$, by reporting (an approximation to) the distance from v to the closest vertex of label λ. Following [9], it was shown in [2] that for any integer $k > 1$, there exists a vertex-label distance oracle with stretch $4k - 5$, query time $O(k)$, and size $O(k \cdot n \cdot \ell^{1/k})$.

This state-of-the-art result suffers from two main drawbacks: The stretch is roughly a factor of 2 larger than in [16], and it is not path-reporting. We address these concerns in this work, and provide the following results.

- First, we devise a *path-reporting* vertex-label distance oracle, at the cost of a slight increase in stretch and size. For any constant $0 < \epsilon < 1$, our oracle has stretch $(4k - 5) \cdot (1 + \epsilon)$, query time $O(k)$, and size $O(n^{1+o(1)} \cdot \ell^{1/k})$.
- Second, we show how to improve the stretch to the optimal $2k-1$, at the cost of mildly increasing the query time. Specifically, we devise a vertex-label distance oracle with stretch $2k - 1$, query time $O(\ell^{1/k} \cdot \log n)$, and size $O(k \cdot n \cdot \ell^{1/k})$.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Sparsification and spanners

Keywords and phrases Graph Algorithms, Shortest Paths, Distance Oracles

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.35

Funding *Ofer Neiman*: Partially supported by the Israel Science Foundation (grant No. 970/21). *Alon Spector*: Partially supported by the Lynn and William Frankel Center for Computer Sciences, and the Israel Science Foundation (grant No. 970/21).

1 Introduction

Let $G = (V, E)$ be an undirected graph, with positive weights on the edges $w : E \to \mathbb{R}_+$. An (approximate) *distance oracle* is a data structure that can (approximately) answer distance queries. The oracle has stretch t, if for every pair $u, v \in V$, the answer it provides $\tilde{d}(u, v)$ to a given query (u, v) satisfies

$$\mathrm{dist}(u, v) \leq \tilde{d}(u, v) \leq t \cdot \mathrm{dist}(u, v) \ ,$$

where $\mathrm{dist}(\cdot)$ is the distance in G with respect to the weights. The main interest is the tradeoff between the stretch t, the size of the oracle (number of words needed to store it), and the query time. In a seminal work, [16] showed for any n-vertex graph and any integer $k \geq 1$,

© Ofer Neiman and Alon Spector;
licensed under Creative Commons License CC-BY 4.0

a distance oracle with stretch $2k-1$, size $O(kn^{1+1/k})$ and query time $O(k)$. Subsequently, the size and query time were improved to $O(n^{1+1/k})$ and $O(1)$ respectively, [17, 3]. This state-of-the-art tradeoff is optimal (up to constants), assuming Erdős' girth conjecture.

A useful feature of a distance oracle is its ability to report not just an approximation $\tilde{d}(u,v)$ to the distance, but also provide a path in G between u,v of length $\tilde{d}(u,v)$. An oracle with this property is called *path-reporting*. The original construction of [16] provides a path-reporting oracle, albeit the improved one of [3] does not. Recently there has been a surge of interest in path-reporting distance oracles [5, 6, 1, 14, 7, 4], most of these were interested in obtaining (almost) linear size, as the [16] oracles always have size $\Omega(n \log n)$.

When stating the query time of path-reporting oracles, we will follow the convention of omitting the length of the returned path. That is, if the algorithm returns a path P in time $O(q+|P|)$, we will write that the query time is $O(q)$ (since reporting P will always incur time $O(|P|)$).

1.1 Vertex-Label Distance Oracles

In this paper we focus on distance oracles for *vertex-labeled* graphs, introduced by [9]. In this setting, there is a set $L=\{\lambda_1,\ldots,\lambda_\ell\}$ of labels, and each vertex receives a single label from L. We would like to answer *vertex-label* queries, which are queries of the form (v,λ), for $v \in V$ and $\lambda \in L$, and the goal is to return an approximation for $\mathrm{dist}(v,\lambda)$, which is the distance from v to the nearest vertex of label λ.

Vertex-label queries arise naturally in various applications, where there are multiple vertices that can provide a service to the source vertex v. For instance, in a network, the labels may indicate different types of servers that provide certain functionality, or in a road network, the labels may correspond to services such as gas, food, accommodation, etc. We would like to be able to answer queries such as "how close is the nearest gas station", rather than the distance to a particular one.

In [9] it was shown that the distance oracles of [16] can be adapted to the vertex-label query setting, with stretch $4k-5$ and size $O(kn^{1+1/k})$, regardless of the number of labels, ℓ. However, if $\ell < n^{1/k}$, then this size is larger than the trivial solution with stretch 1 and size $O(n \cdot \ell)$, that simply stores all distances between vertices and labels. To address this issue, Hermelin et al. [9] devised an oracle with improved size of $O(kn \cdot \ell^{1/k})$, at the cost of increasing the stretch to 2^k-1. In [2], the stretch was substantially improved to $4k-5$. Both results have query time $O(k)$.

The current state-of-the-art leaves two main concerns. The first is that unlike standard distance oracles, the vertex-label oracles of [9, 2] of size $\approx n \cdot \ell^{1/k}$ are not *path-reporting*. Indeed, for many applications the ability to report paths seems appealing (for instance, rather than obtaining just the distance to the nearest gas station, we would like to receive a path to it). The second concern is that the stretch is roughly a factor of 2 away from the conjectured optimum.

1.2 Our Results

In this paper we provide a positive answer to both concerns mentioned above. That is, by slightly increasing the stretch, we show how to obtain path-reporting vertex-label distance oracles. In addition, we show a construction that achieves the optimal stretch of $2k-1$, at the cost of mildly increasing the query time.

Specifically, for any n-vertex graph, given an integer $k > 1$ and constant $0 < \epsilon < 1$, we devise a path-reporting distance oracle, that answers vertex-label queries in $O(k)$ time, and has stretch $(4k-5)\cdot(1+\epsilon)$ and size $O(n^{1+o(1)}\cdot\ell^{1/k})$. (Alternatively, with stretch $O(k)$ and size $\tilde{O}(n\cdot\ell^{1/k})$.) [1]

Our second result achieves for any integer $k > 1$, a vertex-label distance oracle with stretch $2k-1$, size $O(k\cdot n\cdot\ell^{1/k})$, and query time $O(\ell^{1/k}\cdot\log n)$. (This oracle can also be made path-reporting, at the cost of increasing the stretch by a factor of $1+\epsilon$, and the size by a factor of $n^{o(1)}$.)

1.3 Overview of Techniques

Our construction of the path-reporting vertex-label distance oracle starts similarly to previous works [9, 2]. Given the integer parameter $k > 1$, we sample a sequence of sets $V = A_0 \supseteq A_1 \supseteq ... \supseteq A_{k-1} \supseteq A_k = \emptyset$, as follows. For each $0 \le i \le k-2$, every vertex in A_i is independently sampled to A_{i+1} with probability $\ell^{-1/k}$ (recall that $\ell = |L|$ is the number of labels). This sampling probability is different from the $n^{-1/k}$ used in [16], which allows the size improvement from $\approx n^{1+1/k}$ to $\approx n\cdot\ell^{1/k}$. However, the main issue is that the last set A_{k-1} is expected to be prohibitively large, $\frac{n}{\ell^{1-1/k}}$.

As in [16], for each $v \in V$ and $0 \le i \le k-1$ define *pivots*: $p_i(v)$ is the nearest vertex of A_i to v, and *bunches*: $B_i(v)$ contains all the vertices in A_i closer to v than $p_{i+1}(v)$, and $B(v) = \bigcup_i B_i(v)$. The *cluster* of $u \in V$, $C(u)$, consists of all vertices v such that $u \in B(v)$.

Path-reporting construction: The query algorithm of [2], given the query (v,λ), finds a pivot $p_i(v)$ such that there exists some vertex u of label λ with $p_i(v) \in B(u)$. It then returns the sum of distances from u, v to this pivot $p_i(v)$. We take a similar approach, and as long as $i < k-1$, we can apply the technique of [16], using clusters, to find a path from v to u via the cluster of $p_i(v)$. The issue arises when $i = k-1$, then the clusters of vertices in this last level A_{k-1} are too large to store. To overcome this hurdle, we apply a *pairwise* path-reporting distance oracle, that approximately preserves distances (and reports paths), only between a given set of pairs. Indeed, we show that the set of pairs containing all the possible required connections for vertices in A_{k-1} is sufficiently small, and the pairwise oracles of [7, 14] induce a slight increase in stretch and size.

Improved stretch: The query algorithm mentioned above, given a query (v,λ), tests only the pivots of v. This *one-sided testing* enables stretch of $4k-5$, similarly to the context of compact routing schemes [15], where we have access to the bunches of the source v, but not to the bunches of the destination u. In order to obtain the (conjectured) optimal stretch of $2k-1$ for vertex-label queries, we need *two-sided testing*, as in [16, 3], where we also inspect whether the pivots of the destination u are in $B(v)$. Alas, the destination u is not known, we are only given a label λ. In order to achieve two-sided testing, it seems that in the i-th iteration, one must check for each vertex u of label λ, whether $p_i(u) \in B(v)$. Unfortunately, this may result in a linear query time. To deal with this barrier, in the pre-processing phase, for each $\lambda \in L$, we store all the level i pivots of vertices with label λ, denoted $P_i(\lambda)$, in a hash table. Then, we can check if there exists a vertex of $B_i(v)$ in $P_i(\lambda)$. This will be much faster, since the size of $B_i(v)$, for $i < k-1$, is bounded by $\approx \ell^{1/k}$. As the last set of level $k-1$ can be quite large, we will need to treat it differently, which leads to an intricate case analysis when bounding the stretch.

[1] When writing $\tilde{O}(f(n))$ we mean $O(f(n)\cdot\log^{O(1)}(f(n)))$.

1.4 Related Work

In the context of planar graphs, [13] showed a vertex-label distance oracle with stretch $1+\epsilon$, size $O(n \log n)$ and query time $O(\log n \cdot \log \Delta)$, where $0 < \epsilon < 1$ is fixed and Δ is the hop-diameter of the graph. This result is somewhat inferior to state-of-the-art distance oracles for planar graph, where [12] obtained stretch $1+\epsilon$, size $O(n)$ and query time $O(1)$. In [8], a stretch 1 vertex-label distance oracle for planar graphs is shown, with parameters (size and query) that are at most polylogarithmically larger than the standard setting.

A closely related problem is the so-called *colored distance oracles*, which can answer label-to-label queries. These objects were studied by [11, 10], in the context of detecting patterns in texts, focusing on the $1+\epsilon$ stretch regime.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with nonnegative weights on the edges $w : E \to \mathbb{R}_+$. Given a set L of labels, each vertex is associated with a single label $\lambda \in L$. For $\lambda \in L$, the set of vertices of label λ is denoted by V_λ.

For vertices $u, v \in V$, define $\text{dist}(u, v)$ as the length of the shortest path between u, v with respect to the weights. If H is a subgraph of G, we denote $\text{dist}_H(u, v)$ the distance in H. For a vertex $v \in V$ and a set $A \subseteq V$, define $\text{dist}(v, A) = \min_{u \in A}\{\text{dist}(v, u)\}$. Additionally, let $\text{dist}(v, \emptyset) = \infty$.

For every vertex $v \in V$ and label $\lambda \in L$, we define $\lambda(v)$ as the closest node to v that has the label λ, that is, $\text{dist}(v, \lambda(v)) = \text{dist}(v, V_\lambda)$.

Pairwise distance oracles: Given a set of pairs $\mathcal{P} \subseteq V \times V$, a *pairwise* distance oracle is required to (approximately) answer distance queries only for pairs in $\mathcal{P}$. A pairwise oracle is called *path-reporting* if it can also return a path in G that achieves the approximated distance. We will use two results: the first from [7] that has stretch close to 1, and the second from [14] that has constant stretch, but smaller size. By plugging in $k = \log n$ in [7, Theorem 2] we get[2]

▶ **Theorem 1** ([7]). *Given an undirected weighted n-vertex graph $G = (V, E)$, a parameter $0 < \epsilon \leq 1$, and a set of pairs $\mathcal{P} \subseteq V \times V$, there exists a pairwise path-reporting distance oracle, with stretch $1+\epsilon$, query time $O(1)$ and size*

$$O\left(|\mathcal{P}| \cdot \left(\frac{\log\log n}{\epsilon}\right)^{\log_{4/3}\log n} + n \log\log n\right) .$$

By plugging in a constant $c > 1$ and $k = \log n$ in [14, Theorem 13] we get[3]

▶ **Theorem 2** ([14]). *Given an undirected weighted n-vertex graph $G = (V, E)$ and a set of pairs $\mathcal{P} \subseteq V \times V$, there exists a pairwise path-reporting distance oracle, with stretch $O(1)$, query time $O(1)$ and size*

$$\tilde{O}\left(|\mathcal{P}| + n\right) .$$

[2] Path-reporting oracles are a part of *interactive spanners* in [7].
[3] Path-reporting oracles are a part of *path-reporting spanners* in [14].

3 Path-Reporting Oracles for Labeled Graphs

In this section we prove the following theorem.

▶ **Theorem 3.** *For any n-vertex graph $G = (V, E)$ labeled by a set L of ℓ labels, any integer $k > 1$ and constant $0 < \epsilon < 1$, there exists a path-reporting distance oracle that can answer any vertex-label query in time $O(k)$, with stretch $(4k-5)\cdot(1+\epsilon)$ and size $O(n^{1+o(1)} \cdot \ell^{1/k})$. (Alternatively, with stretch $O(k)$ and size $\tilde{O}(n \cdot \ell^{1/k})$.)*

▶ Remark 4. The parameter ϵ can be nonconstant, in which case the size increases by a factor of $\epsilon^{-O(\log\log n)}$. The term $n^{o(1)}$ is in fact only sightly super-polylogarithmic: $(\log\log n)^{O(\log\log n)}$.

3.1 Preprocessing

Define a random sequence of sets: $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$, where for each $0 \le i \le k-2$, every vertex of A_i is selected for A_{i+1} independently with probability $\ell^{-1/k}$. For each vertex $v \in V$ and $0 \le i \le k-1$ define the i-th *pivot* $p_i(v) \in A_i$ as the nearest vertex[4] in A_i to v, i.e.

$$\mathrm{dist}(v, A_i) = \mathrm{dist}(v, p_i(v)) \ .$$

Define the i-th *bunch* of $v \in V$ by

$$B_i(v) = \{u \in A_i \ : \ \mathrm{dist}(v,u) < \mathrm{dist}(v, A_{i+1})\} \ .$$

Note that the distance to $A_k = \emptyset$ is ∞, so that $B_{k-1}(v) = A_{k-1}$ for every $v \in V$. Let

$$B(v) = \bigcup_{i=0}^{k-1} B_i(v) \ .$$

Additionally, we define $\widetilde{B}(v)$ as the bunch excluding the last level, that is,

$$\widetilde{B}(v) = \bigcup_{i=0}^{k-2} B_i(v) = B(v) \setminus A_{k-1}.$$

The definition of bunches extends naturally to labels, by defining for any $\lambda \in L$,

$$B(\lambda) = \bigcup_{v \in V_\lambda} B(v) \ ,$$

and similarly define $\widetilde{B}(\lambda) = B(\lambda) \setminus A_{k-1}$. For every $u \in \widetilde{B}(\lambda)$, let

$$\lambda_B(u) = \mathrm{argmin}_{x \in V_\lambda : u \in \widetilde{B}(x)} \{\mathrm{dist}(u,x)\}$$

in other words, this is the closest vertex x to u that caused u to enter $\widetilde{B}(\lambda)$; so x has label λ and also satisfies $u \in \widetilde{B}(x)$.

Clusters: Similarly to [16], we use the "inverses" of bunches, called *clusters*. For any $0 \le i \le k-1$ and $u \in A_i \setminus A_{i+1}$, define

$$C(u) = \{v \in V \ : \ \mathrm{dist}(u,v) < \mathrm{dist}(v, A_{i+1})\} \ .$$

[4] If there are ties, we use a consistent tie-breaking scheme: first, we prefer to take the vertex belonging to the highest set. If $\mathrm{dist}(v, A_i) = \mathrm{dist}(v, A_{i+1})$, then $p_i(v) \leftarrow p_{i+1}(v)$; Otherwise, we choose arbitrarily from the set $\{u \in A_i \mid \mathrm{dist}(v, A_i) = \mathrm{dist}(v,u)\}$.

Note that $v \in C(u)$ iff $u \in B(v)$. It was shown in [16] that one can efficiently compute a tree $T(u)$ rooted at u that spans $C(u)$, so that $\text{dist}_{T(u)}(u,v) = \text{dist}(u,v)$ for every $v \in C(u)$.

As pointed out in the introduction, the last set A_{k-1} is expected to be excessively large, of size $\frac{n}{\ell^{1-1/k}}$. So unlike [16] we cannot use clusters to derive paths for vertices in that set. Indeed, the main difference between previous works and our construction, is that we employ a path-reporting pairwise distance oracle for vertices in the last set A_{k-1}. Specifically, we define a set of required pairs $\mathcal{P}$ as follows: For every $v \in V$ we add the pair $(v, p_{k-1}(v))$ to $\mathcal{P}$, and for every $u \in A_{k-1}$ and every label $\lambda \in L$, we add the pair $(u, \lambda(u))$ to $\mathcal{P}$ (recalling that $\lambda(u)$ is the nearest vertex to u that has label λ). That is, define

$$\mathcal{P} = \{(v, p_{k-1}(v)) : v \in V\} \cup \{(u, \lambda(u)) : u \in A_{k-1},\ \lambda \in L\} ,$$

and let $O_{\mathcal{P}}$ be the pairwise path-reporting distance oracle from Theorem 1 (respectively, from Theorem 2) for the set of pairs $\mathcal{P}$.

The following claims will be useful:

▷ Claim 5. For all $v \in V$, and for all $0 \le i \le k-1$, $p_i(v) \in B(v)$.

Proof. We prove the claim by induction on i, proceeding downward. The base case is for $i = k-1$, which holds since $p_{k-1}(v) \in A_{k-1} \subseteq B(v)$ for every $v \in V$. Now, assume the claim holds for $i+1 \le k-1$, so that $p_{i+1}(v) \in B(v)$. If $\text{dist}(v, A_i) = \text{dist}(v, A_{i+1})$, then as we chose pivot by a consistent tie breaking scheme, preferring vertices from higher sets, it follows that $p_i(v) = p_{i+1}(v) \in B(v)$. Otherwise, $\text{dist}(v, p_i(v)) = \text{dist}(v, A_i) < \text{dist}(v, A_{i+1})$, which implies $p_i(v) \in B_i(v) \subseteq B(v)$. ◁

Next, we record some known facts about the expected size of the sets A_i and of the bunches $\widetilde{B}(v)$. For $0 \le i \le k-1$, to be in A_i, a vertex needs to be sampled i times. Since each sample is done independently with probability $\ell^{-1/k}$, we have that

$$\mathbb{E}[|A_i|] = \frac{n}{\ell^{i/k}} . \tag{1}$$

▷ Claim 6. For all $v \in V$, $\mathbb{E}[|\tilde{B}(v)|] \le (k-1) \cdot \ell^{1/k}$. Furthermore, with high probability it holds that for all $v \in V$, $|\tilde{B}(v)| = O(\ell^{1/k} \cdot \log n)$.

The proof is similar to [16, 2], and is deferred to Appendix A.

Data structure: Our path-reporting distance oracle will store in memory the following items:

1. For every $v \in V$, store in a hash table $\widetilde{B}(v)$, and $p_i(v)$ for all $0 \le i \le k-1$. Each vertex is stored along with its distance to v.
2. For every $u \in V \setminus A_{k-1}$, store $C(u)$ and the corresponding tree $T(u)$.
3. For every label $\lambda \in L$, store in a hash table $\widetilde{B}(\lambda)$. Every $u \in \widetilde{B}(\lambda)$ is stored with $\lambda_B(u)$.
4. The collection of pairs $\mathcal{P}$, and the pairwise distance oracle $O_{\mathcal{P}}$.

3.2 Size Analysis

Note that every vertex has k pivots, so by Claim 6, the total storage for the first item of our data structure, the bunches and pivots, is expected to be of $O(k \cdot n \cdot \ell^{1/k})$ words.

In order to bound the size of clusters stored, we recall that clusters are inverses of bunches, i.e., $C(u) = \{v \in V : u \in B(v)\}$. Furthermore, whenever $u \in V \setminus A_{k-1}$ and $u \in B(v)$, we have that $u \in \widetilde{B}(v)$, so

$$\sum_{u \in V \setminus A_{k-1}} |C(u)| = \sum_{u \in V \setminus A_{k-1}} \sum_{v \in V : u \in \widetilde{B}(v)} 1 = \sum_{v \in V} |\widetilde{B}(v)| , \tag{2}$$

So the total storage required for the second item, the clusters of vertices outside A_{k-1}, is expected to be $\mathbb{E}\left[\sum_{v\in V}|\widetilde{B}(v)|\right] = O(k\cdot n\cdot \ell^{1/k})$ as well.

Since $\{V_\lambda\}_{\lambda\in L}$ is a partition of V, every bunch $\widetilde{B}(v)$ is counted exactly once in one of the $\widetilde{B}(\lambda)$, so the expected size of third item is also bounded by $O(k\cdot n\cdot \ell^{1/k})$ words.

Finally, the size of the pairwise distance oracle $O_{\mathcal{P}}$ is at most:

$$O\left(|\mathcal{P}|\cdot\left(\frac{\log\log n}{\epsilon}\right)^{\log_{4/3}\log n} + n\log\log n\right)$$

if we use Theorem 1, and $\tilde{O}\left(|\mathcal{P}|+n\right)$ if we use Theorem 2. It remains to bound the expected size of $\mathcal{P}$. The first set of pairs, from each vertex to its pivot in A_{k-1}, consists of n pairs, so we focus now on the second set of pairs in $\mathcal{P}$: $\{(u,\lambda(u)) : u\in A_{k-1},\ \lambda\in L\}$. By (1), we expect $\frac{n}{\ell^{1-1/k}}$ vertices to be in A_{k-1}, each of them adds ℓ pairs to $\mathcal{P}$, one for each label. We get that

$$\mathbb{E}[|\mathcal{P}|] = n + \frac{n}{\ell^{1-1/k}}\cdot\ell = O(n\cdot\ell^{1/k})\ .$$

Thus, the expected size of the pairwise distance oracle $O_{\mathcal{P}}$ of Theorem 1 is at most

$$O\left(n\cdot\ell^{1/k}\cdot\left(\frac{\log\log n}{\epsilon}\right)^{\log_{4/3}\log n}\right) = O\left(n^{1+o(1)}\cdot\ell^{1/k}\right)\ ,$$

for constant $0<\epsilon<1$. Alternatively, it is at most $\tilde{O}(n\cdot\ell^{1/k})$ using Theorem 2.

3.3 Query Algorithm

Let (v,λ) be the query, for some $v\in V$ and $\lambda\in L$. The query algorithm consists of two phases.

Phase 1: For every $0\le i\le k-2$, check whether $p_i(v)\in\widetilde{B}(\lambda)$. If so, then let $x_i=\lambda_B(p_i(v))$ be the vertex in V_λ that is returned by our data structure when querying $\widetilde{B}(\lambda)$ (recall that this is the closest vertex of label λ to $p_i(v)$ such that $p_i(v)\in\widetilde{B}(x_i)$). Define $D_i=\mathrm{dist}(v,p_i(v))+\mathrm{dist}(p_i(v),x_i)$. Note that both of these distances are stored in the first item of our data structure.

In the case that $p_i(v)\notin\widetilde{B}(\lambda)$, then simply set $D_i=\infty$.

Phase 2: Extract $x_{k-1}=\lambda(p_{k-1}(v))$ from $\mathcal{P}$ (recall that this is the closest vertex of label λ to $p_{k-1}(v)$), and use the pairwise path-reporting distance oracle $O_{\mathcal{P}}$ to (approximately) compute $D_{k-1}=\mathrm{dist}(v,p_{k-1}(v))+\mathrm{dist}(p_{k-1}(v),x_{k-1})$.

Distance: Let $i^*=\mathrm{argmin}_{0\le i\le k-1}\{D_i\}$, and return D_{i^*} as the approximation to the distance from v to $\lambda(v)$.

Path-reporting: If $i^*<k-1$, then let $T=T(p_{i^*}(v))$. Note that $x_{i^*}\in C(p_{i^*}(v))$ and by Claim 5 also $v\in C(p_{i^*}(v))$, so both x_{i^*} and v are in T, and we return the path in T from x_{i^*} to v (by going from each of them to the root $p_{i^*}(v)$). Otherwise, if $i^*=k-1$, use the path-reporting oracle $O_{\mathcal{P}}$ to return the path from v to $p_{k-1}(v)$ concatenated with the path from $p_{k-1}(v)$ to x_{k-1}.

Query time: Let P be the returned path, then the running time of the query algorithm is $O(k+|P|)$, since in each of the k iterations we check $O(1)$ vertices and distances from the hash tables. If $i^*<k-1$, then the path P in the tree T can be recovered in $O(|P|)$ time. Otherwise, the oracle $O_{\mathcal{P}}$ recovers P, also in $O(|P|)$ time. As we state the query time omitting the length of the returned path, we indeed obtain query time $O(k)$.

3.4 Stretch Analysis

In what follows we assume that $A_{k-1} \neq \emptyset$ (which happens w.h.p.), still this is without loss of generality, as otherwise we can simply redefine k as the minimal for which $A_k = \emptyset$. The following lemma is very similar to [15, Lemma A.1], we give a proof for completeness in Appendix A.

▶ **Lemma 7.** *Fix any $u, v \in V$, and let $0 \leq j \leq k-1$ be the smallest index such that $p_j(v) \in B(u)$. Then,*

$$\mathrm{dist}(v, p_j(v)) + \mathrm{dist}(u, p_j(v)) \leq (4k-3) \cdot \mathrm{dist}(u, v)$$

Let (v, λ) be the given query, set $u = \lambda(v)$ the closest vertex to v that has label λ. We are now ready to analyze the stretch of the returned path, according to the following two cases. We stress that these cases are not analogous to the two phases of the algorithm, since in the analysis here we consider $\widetilde{B}(u)$, and u is not known to the query algorithm (which considers $\widetilde{B}(\lambda)$).

Case 1: There exists $0 \leq i \leq k-2$ such that $p_i(v) \in \widetilde{B}(u)$. Denote i as the minimal such index, and let $x_i = \lambda_B(p_i(v))$. Recall that x_i is the closest vertex to $p_i(v)$ that satisfies: 1) has label λ , and 2) contains $p_i(v)$ in its bunch $\widetilde{B}(\cdot)$. Since u satisfies both these conditions, we have that

$$\mathrm{dist}(p_i(v), x_i) \leq \mathrm{dist}(p_i(v), u) \ . \tag{3}$$

Using that i is the minimal such that $p_i(v) \in B(u)$, by Lemma 7 it follows that

$$D_i = \mathrm{dist}(v, p_i(v)) + \mathrm{dist}(p_i(v), x_i) \overset{(3)}{\leq} \mathrm{dist}(v, p_i(v)) + \mathrm{dist}(p_i(v), u) \leq (4k-3) \cdot \mathrm{dist}(v, u) \ .$$

So by definition, the returned distance $D_{i^*} \leq D_i$ is also a $4k-3$ approximation. Note that the returned path in the tree $T = T(p_{i^*}(v))$ incurs no additional stretch.

Case 2: For all $0 \leq i \leq k-2$, $p_i(v) \notin \widetilde{B}(u)$. Denote i as the minimal index such that $p_i(v) \in B(u)$. (There is such $0 \leq i \leq k-1$ because $B_{k-1}(u) = A_{k-1} \subseteq B(u)$.) As $p_i(v) \in B(u) \setminus \widetilde{B}(u)$ it follows that $p_i(v) \in A_{k-1}$ i.e., $p_i(v) = p_{k-1}(v)$ (see footnote 4). Recall that in phase 2 the algorithm sets $x_{k-1} = \lambda(p_{k-1}(v))$, the closest vertex of label λ to $p_{k-1}(v)$, so we have that

$$\mathrm{dist}(p_{k-1}(v), x_{k-1}) \leq \mathrm{dist}(p_{k-1}(v), u) \ . \tag{4}$$

Once again using Lemma 7 we get

$$\begin{aligned} D_{k-1} &= \mathrm{dist}(v, p_{k-1}(v)) + \mathrm{dist}(p_{k-1}(v), x_{k-1}) \\ &\overset{(4)}{\leq} \mathrm{dist}(v, p_{k-1}(v)) + \mathrm{dist}(p_{k-1}(v), u) \\ &\leq (4k-3) \cdot \mathrm{dist}(v, u) \ . \end{aligned}$$

Once again using Lemma 7 we get So in this case as well, the distance $D_{i^*} \leq D_{k-1}$ is also a $4k-3$ approximation. However, in this case the returned path is using the path-reporting pairwise distance oracle $O_{\mathcal{P}}$, which incurs an additional $1+\epsilon$ factor to the stretch if we used Theorem 1 for its construction, or an $O(1)$ factor if we used Theorem 2.

3.4.1 Improving the Stretch

To obtain the slightly improved stretch $4k-5$, we take a similar approach to [15]. The basic idea is to make a two-sided test in the first level, and one-sided tests in all other levels.[5] To implement this idea in our setting, we define the cluster of a label $\lambda \in L$ by

$$C(\lambda) = \bigcup_{u \in V_\lambda \setminus A_1} C(u) \ .$$

Note that we take a union only over first-level vertices in $A_0 \setminus A_1$. Therefore, by (2), we can store every $C(\lambda)$ in a hash table with expected space only $O(k \cdot n \cdot \ell^{1/k})$. Every vertex $v \in C(\lambda)$ is stored together with $u = \lambda_C(v)$, which is the closest vertex to v that satisfies $u \in V_\lambda \setminus A_1$ and $v \in C(u)$. In other words, this is the closest vertex to v that caused v to enter $C(\lambda)$.

The query algorithm will start by testing whether $v \in C(\lambda)$, and if so, it will return the path from $u = \lambda_C(v)$ to v in $T(u)$ (note that as $u \notin A_{k-1}$, this tree is indeed stored in our data structure). If $v \notin C(\lambda)$, continue with the query algorithm described in Section 3.3. The following claim ensures that we get stretch 1 if the query algorithm finds $v \in C(\lambda)$.

▷ Claim 8. If $v \in C(\lambda)$ and $u = \lambda_C(v)$, then $u = \lambda(v)$.

Proof. Since $u \in A_0 \setminus A_1$ and $v \in C(u)$, by definition it follows that $u \in B_0(v)$. Seeking contradiction, suppose that $u' = \lambda(v)$ is different than u. If $u' \in A_1$, then by definition of B_0, and of a pivot, we get that

$$\mathrm{dist}(v,u) < \mathrm{dist}(v,p_1(v)) \leq \mathrm{dist}(v,u') \ ,$$

contradiction. Otherwise, $u' \in A_0 \setminus A_1$, but then $u' \in V_\lambda \setminus A_1$ is closer to v than u, so $u' \in B_0(v)$ as well, contradicting the definition of $u = \lambda_C(v)$ as the closest such vertex. ◁

Next, we show an improved bound on the distance from v to its first pivot in the case that $v \notin C(\lambda)$.

▷ Claim 9. Suppose that $v \notin C(\lambda)$, then $\mathrm{dist}(v,p_1(v)) \leq \mathrm{dist}(v,\lambda(v))$.

Proof. Let $u = \lambda(v)$. If $u \in A_1$, then by definition of a pivot, we have that $\mathrm{dist}(v,u) \geq \mathrm{dist}(v,p_1(v))$. Otherwise, $u \in A_0 \setminus A_1$, but by the assertion of the lemma, $v \notin C(u)$, so $u \notin B_0(v)$, which implies that $\mathrm{dist}(v,u) \geq \mathrm{dist}(v,p_1(v))$. ◁

Recall that in the stretch analysis, (9) for $i = 1$ gave us the bound $\mathrm{dist}(v,p_1(v)) \leq 2\Delta$ (where $\Delta = \mathrm{dist}(v,\lambda(v))$). Now, with the improved bound of $\mathrm{dist}(v,p_1(v)) \leq \Delta$ from Claim 9, the stretch will be smaller by an additive 2 (see [15, Lemma A.2]), resulting in stretch $4k-5$ instead of $4k-3$.

4 Distance Oracles for Labeled Graphs with $2k-1$ Stretch

In this section we prove the following theorem.

▶ **Theorem 10.** *For any n-vertex graph $G = (V,E)$ labeled by a set L of ℓ labels, and any integer $k \geq 1$, there exists a distance oracle that can answer any vertex-label query in time $O(\ell^{1/k} \cdot \log n)$, with stretch $2k-1$ and size $O(k \cdot n \cdot \ell^{1/k})$.*

[5] In the context [15] were interested in, compact routing schemes, this amounts to storing the clusters of vertices in the first level $A_0 \setminus A_1$, and ensuring they are sufficiently small.

▶ Remark 11. For any $0 < \epsilon < 1$, the oracle of Theorem 10 can be made path-reporting as well, at the cost of increasing the stretch to $2k - 1 \cdot (1 + \epsilon)$, and the size to

$$O\left(\left(\frac{\log\log n}{\epsilon}\right)^{O(\log\log n)} \cdot n \cdot \ell^{1/k}\right)$$

4.1 Preprocessing

We will use a similar data structure as in Section 3, excluding the pairwise distance oracle $O_{\mathcal{P}}$ and the clusters. Additionally, for every $0 \leq i \leq k-2$ and every $\lambda \in L$, we will store a set containing all the i-th level pivots of vertices with label λ. Formally,

$$P_i(\lambda) = \{p_i(v) : v \in V_\lambda\}$$

For every $y \in P_i(\lambda)$, let $\lambda_{P_i}(y) = \text{argmin}_{x \in V_\lambda : p_i(x) = y}\{\text{dist}(x, y)\}$, in other words, this is the closest vertex x to y that caused y to enter $P_i(\lambda)$; so x has label λ and also satisfies $p_i(x) = y$.

We will also store for all vertices in the last level A_{k-1}, the distances to the closest vertex of label λ, for every $\lambda \in L$.

Data structure: Our distance oracle will store in memory the following items:

1. For every $v \in V$, store in a hash table $\widetilde{B}(v)$, and $p_i(v)$ for all $0 \leq i \leq k-1$. Each vertex is stored along with its distance to v.
2. For every label $\lambda \in L$, store in a hash table $\widetilde{B}(\lambda)$. Every $u \in \widetilde{B}(\lambda)$ is stored with $\lambda_B(u)$ and the distance between them.
3. For every label $\lambda \in L$ and for every $0 \leq i \leq k-2$, store in a hash table $P_i(\lambda)$. Every $y \in P_i(\lambda)$ is stored with $\lambda_{P_i}(y)$ and the distance between them.
4. For every vertex $v \in A_{k-1}$ and every label $\lambda \in L$, store $\lambda(v)$ and the distance $\text{dist}(v, \lambda(v))$.

4.2 Size Analysis

In section 3.2 it was shown that the total expected storage required for the first two items is $O(k \cdot n \cdot \ell^{1/k})$ words. We show next that the third item requires only $O(k \cdot n)$ space. To see this, note that for any $0 \leq i \leq k-2$ and $\lambda \in L$, it holds that $|P_i(\lambda)| \leq |V_\lambda|$ (since every vertex in V_λ adds one of its pivots to this set). Therefore,

$$\sum_{i=0}^{k-2} \sum_{\lambda \in L} |P_i(\lambda)| \leq k \cdot \sum_{\lambda \in L} |V_\lambda| = k \cdot n \ .$$

For the last item, recall that by (1) we have $\mathbb{E}[|A_{k-1}|] = \frac{n}{\ell^{1-1/k}}$. As every vertex in A_{k-1} stores $O(\ell)$ words, the expected space required for the last item is $O(n \cdot \ell^{1/k})$.

Thus, the total expected size of our data structure is $O(k \cdot n \cdot \ell^{1/k})$ words.

4.3 Query Algorithm

Let (v, λ) be the query, for some $v \in V$ and $\lambda \in L$. The query algorithm consists of three phases. Denote $\tilde{B}(v) = \{v_1, v_2, \ldots, v_{|\tilde{B}(v)|}\}$.

Phase 1: For every $0 \leq i \leq k-2$, check whether $p_i(v) \in \widetilde{B}(\lambda)$. If so, then let $x_i = \lambda_B(p_i(v))$ be the vertex in V_λ that is returned by our data structure when querying $\widetilde{B}(\lambda)$ (recall that this is the closest vertex of label λ to $p_i(v)$ such that $p_i(v) \in \widetilde{B}(x_i)$). Define $D_i = \text{dist}(v, p_i(v)) + \text{dist}(p_i(v), x_i)$. Note that both of these distances are stored in the first item of the data structure.

In the case that $p_i(v) \notin \widetilde{B}(\lambda)$, then simply set $D_i = \infty$.

Phase 2: For every $1 \leq j \leq |\tilde{B}(v)|$, let $i = i(j)$ be the unique index such that $v_j \in A_i \setminus A_{i+1}$, noting that $0 \leq i \leq k-2$ because $\tilde{B}(v)$ does not contain vertices in A_{k-1}. Then, check whether $v_j \in P_i(\lambda)$. If so, then let $y_j = \lambda_{P_i}(v_j)$ be the vertex in V_λ that is returned by our data structure when querying $P_i(\lambda)$ (recall that this is the closest vertex of label λ to v_j such that $v_j = p_i(y_j)$). Define $E_j = \text{dist}(v, v_j) + \text{dist}(v_j, y_j)$. Again, both of these distances are stored in the first item of the data structure.

In the case that $v_j \notin P_i(\lambda)$, then simply set $E_j = \infty$.

Phase 3: Extract $x_{k-1} = \lambda(p_{k-1}(v))$ from the data structure (recall that this is the closest vertex of label λ to $p_{k-1}(v)$). Compute $D_{k-1} = \text{dist}(v, p_{k-1}(v)) + \text{dist}(p_{k-1}(v), x_{k-1})$. These distances are stored in the first (respectively, fourth) item of the data structure.

Distance: Let $i^* = \text{argmin}_{0 \leq i \leq k-1}\{D_i\}$, and $j^* = \text{argmin}_{1 \leq j \leq |\tilde{B}(v)|}\{E_j\}$. We return $\min\{D_{i^*}, E_{j^*}\}$ as the approximation to the distance from v to $\lambda(v)$.

Query time: The running time of the query algorithm is $O(k)$ for phase 1, $O(|\tilde{B}(v)|)$ for phase 2, and $O(1)$ for phase 3, so in total it is $O(k + |\tilde{B}(v)|)$. By Claim 6, with high probability for every $v \in V$ we have $|\tilde{B}(v)| \leq O(\ell^{1/k} \cdot \log n)$. As $k \leq \log \ell \leq \log n$, we obtain query time $O(\ell^{1/k} \cdot \log n)$.

4.3.1 Path-Reporting

The query algorithm described above can be converted to a path-reporting one, by incorporating the technique of Section 3. To this end, we will add to the data structure the following items, for a given a parameter $0 < \epsilon < 1$.

- For every $u \in V \setminus A_{k-1}$, store $C(u)$ and the corresponding tree $T(u)$.
- The collection of pairs $\mathcal{P}$, and the pairwise distance oracle $O_{\mathcal{P}}$.

As proved in Section 3.2, this will increase the size of our data structure by a factor of $\left(\frac{\log \log n}{\epsilon}\right)^{O(\log \log n)}$. Note that the distance returned in phase 1 or 2 of the query algorithm is the sum of distances from a pivot $p \in V \setminus A_{k-1}$, to two vertices in $C(p)$. Therefore, the tree $T(p)$ which is stored in our data structure, contains the corresponding paths, and they can be recovered in linear time in the length of the paths. Finally, if the distance returned is from phase 3, we can use $O_{\mathcal{P}}$ to report the paths, increasing the stretch by a factor of $1 + \epsilon$.

4.4 Stretch Analysis

As in section 3.4, we assume that $A_{k-1} \neq \emptyset$. The following lemma is similar to [16, Lemma 3.3], we give a proof for completeness in Appendix A.

▶ **Lemma 12.** *Fix any $u, v \in V$, and let $0 \leq i \leq k-1$ be the smallest index such that $p_i(v) \in B(u)$ or $p_i(u) \in B(v))$. Then:*

$$dist(v, p_i(v)) + dist(p_i(v), u) \leq (2i+1) \cdot dist(u, v)$$

Let (v, λ) be the given query, set $u = \lambda(v)$ the closest vertex to v that has label λ. Denote $i_u = \text{argmin}_{0 \leq i \leq k-1}\{p_i(u) \in B(v)\}$ as the minimal index i such that $B(v)$ contains the i-th pivot of u (note that i_u is well defined, since $p_{k-1}(u) \in A_{k-1} \subseteq B(v)$). Likewise, let $i_v = \text{argmin}_{0 \leq i \leq k-1}\{p_i(v) \in B(u)\}$. We are now ready to analyze the stretch of the returned distance, according to the following cases. As before, these cases are not analogous to the phases of the algorithm.

Case 1: $i_v \leq i_u$ and $p_{i_v}(v) \notin A_{k-1}$. Let $i = i_v$, and note that $p_i(v) \in \tilde{B}(u)$. In phase 1. of the query, we take x_i which is the closest vertex to $p_i(v)$ that satisfies: 1) has label λ, and 2) contains $p_i(v)$ in its bunch $\widetilde{B}(\cdot)$. Since u satisfies both these conditions, we have that

$$\text{dist}(p_i(v), x_i) \leq \text{dist}(p_i(v), u) \ . \tag{5}$$

By the minimality of $i = i_v \leq i_u$, we may use Lemma 12 and get that

$$D_i = \text{dist}(v, p_i(v)) + \text{dist}(p_i(v), x_i) \overset{(5)}{\leq} \text{dist}(v, p_i(v)) + \text{dist}(p_i(v), u) \leq (2i+1) \cdot \text{dist}(v, u) \ ,$$

and as $i \leq k-1$ we conclude that

$$D_i \leq (2k-1) \cdot \text{dist}(v, u) \ .$$

Case 2: $i_u < i_v$ and $p_{i_u}(u) \notin A_{k-1}$. Let $i = i_u$, and note that there exists $v_j \in \tilde{B}(v)$ such that $v_j = p_i(u)$. Let i' be the unique index such that $v_j \in A_{i'} \setminus A_{i'+1}$ (observe that $i \leq i' \leq k-2$, since $v_j \in A_i$ but $v_j \notin A_{k-1}$). In phase 2. of the query we take $y_j = \lambda_{P_{i'}}(v_j)$, which is the closest vertex to v_j that satisfies: 1) has label λ , and 2) v_j is its level i' pivot. We would like to show that u satisfies both these conditions: clearly it has label λ, and by our consistent choice of pivots, since v_j is the level i pivot of u, it is also its level i' pivot (see Footnote [4]). We conclude that

$$\text{dist}(v_j, y_j) \leq \text{dist}(v_j, u) \ . \tag{6}$$

By the minimality of $i = i_u < i_v$, we may use Lemma 12 (switching the roles of u, v). Recalling that $v_j = p_i(u)$, we get that

$$E_j = \text{dist}(v, v_j) + \text{dist}(v_j, y_j) \overset{(6)}{\leq} \text{dist}(v, v_j) + \text{dist}(v_j, u) \leq (2k-1) \cdot \text{dist}(v, u) \ .$$

Case 3: $i_v \leq i_u$ and $p_{i_v}(v) \in A_{k-1}$. Let $i = i_v$, and note that by our consistent choice of pivots, $p_i(v) = p_{k-1}(v)$. Recall that in phase 3., the algorithm sets $x_{k-1} = \lambda(p_{k-1}(v))$, the closest vertex of label λ to $p_{k-1}(v)$, so we have that

$$\text{dist}(p_{k-1}(v), x_{k-1}) \leq \text{dist}(p_{k-1}(v), u) \ . \tag{7}$$

Again using the minimality of $i = i_v$, by Lemma 12 we get

$$\begin{aligned} D_{k-1} &= \text{dist}(v, p_{k-1}(v)) + \text{dist}(p_{k-1}(v), x_{k-1}) \\ &\overset{(7)}{\leq} \text{dist}(v, p_{k-1}(v)) + \text{dist}(p_{k-1}(v), u) \\ &= \text{dist}(v, p_i(v)) + \text{dist}(p_i(v), u) \\ &\leq (2k-1) \cdot \text{dist}(v, u) \ . \end{aligned}$$

Case 4: $i_u < i_v$ and $p_{i_u}(u) \in A_{k-1}$. Set $i = i_u$ and again by the consistent choice of pivots, $p_i(u) = p_{k-1}(u)$. By definition of a pivot,

$$\text{dist}(v, p_{k-1}(v)) \leq \text{dist}(v, p_{k-1}(u)) = \text{dist}(v, p_i(u)) \ . \tag{8}$$

Using the triangle inequality, and the minimality of i, we have

$$\begin{aligned} D_{k-1} &= \mathrm{dist}(v, p_{k-1}(v)) + \mathrm{dist}(p_{k-1}(v), x_{k-1}) \\ &\overset{(7)}{\leq} \mathrm{dist}(v, p_{k-1}(v)) + \mathrm{dist}(p_{k-1}(v), u) \\ &\leq \mathrm{dist}(v, p_{k-1}(v)) + \mathrm{dist}(p_{k-1}(v), v) + \mathrm{dist}(v, u) \\ &\overset{(8)}{\leq} 2 \cdot \mathrm{dist}(v, p_i(u)) + \mathrm{dist}(v, u) \\ &\leq \mathrm{dist}(v, p_i(u)) + \mathrm{dist}(u, p_i(u)) + 2 \cdot \mathrm{dist}(v, u) \\ &\overset{\text{Lemma 12}}{\leq} (2 \cdot i + 3) \cdot \mathrm{dist}(v, u) \ . \end{aligned}$$

Finally, note that as $i < i_v \leq k-1$, it follows that $i \leq k-2$, therefore

$$D_{k-1} \leq (2 \cdot (k-2) + 3) \cdot \mathrm{dist}(v, u) \qquad = (2k-1) \cdot \mathrm{dist}(v, u) \ .$$

In each of these four cases we demonstrated that one of the D_i or E_j is a $2k-1$ approximation to $\mathrm{dist}(v, u)$, so we conclude the same for returned distance $\min\{D_{i^*}, E_{j^*}\}$.

References

1. Ittai Abraham, Shiri Chechik, Michael Elkin, Arnold Filtser, and Ofer Neiman. Ramsey spanning trees and their applications. *ACM Trans. Algorithms*, 16(2):19:1–19:21, 2020. doi:10.1145/3371039.
2. Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings 20*, pages 325–336. Springer, 2012. doi:10.1007/978-3-642-33090-2_29.
3. Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015. doi:10.1145/2746539.2746562.
4. Shiri Chechik and Tianyi Zhang. Path-reporting distance oracles with logarithmic stretch and linear size. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.42.
5. Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016. doi:10.1016/J.TCS.2016.07.038.
6. Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016. doi:10.1145/2888397.
7. Michael Elkin and Idan Shabat. Path-reporting distance oracles with logarithmic stretch and size o(n log log n). In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2278–2311. IEEE, 2023. doi:10.1109/FOCS57990.2023.00141.
8. Jacob Evald, Viktor Fredslund-Hansen, and Christian Wulff-Nilsen. Near-optimal distance oracles for vertex-labeled planar graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.23.
9. Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July*

4-8, 2011, Proceedings, Part II, volume 6756 of *Lecture Notes in Computer Science*, pages 490–501. Springer, 2011. doi:10.1007/978-3-642-22012-8_39.

10 Noam Horowicz and Tsvi Kopelowitz. Color distance oracles and snippets: Separation between exact and approximate solutions. *CoRR*, abs/2507.04578, 2025. doi:10.48550/arXiv.2507.04578.

11 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPIcs*, pages 24:1–24:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.24.

12 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 363–374. IEEE, 2021. doi:10.1109/FOCS52979.2021.00044.

13 Mingfei Li, Chu Chung Christopher Ma, and Li Ning. $(1+\epsilon)$-distance oracles for vertex-labeled planar graphs. In T.-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan, editors, *Theory and Applications of Models of Computation, 10th International Conference, TAMC 2013, Hong Kong, China, May 20-22, 2013. Proceedings*, volume 7876 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2013. doi:10.1007/978-3-642-38236-9_5.

14 Ofer Neiman and Idan Shabat. On the size overhead of pairwise spanners. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 83:1–83:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ITCS.2024.83.

15 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10, 2001. doi:10.1145/378580.378581.

16 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

17 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 539–549. SIAM, 2013. doi:10.1137/1.9781611973105.39.

A Missing Proofs

A.1 Proof of Claim 6

▷ Claim 6. For all $v \in V$, $\mathbb{E}[|\tilde{B}(v)|] \leq (k-1) \cdot \ell^{1/k}$. Furthermore, with high probability it holds that for all $v \in V$, $|\tilde{B}(v)| = O(\ell^{1/k} \cdot \log n)$.

Proof. Fix any $0 \leq i \leq k-2$. As shown in [16], $|B_i(v)|$ is stochastically dominated by a geomteric random variable X_i with parameter $p = \ell^{-1/k}$. It follows that

$$\mathbb{E}[|\tilde{B}(v)|] = \sum_{i=0}^{k-2} \mathbb{E}[|B_i(v)|] \leq \sum_{i=0}^{k-2} \mathbb{E}[X_i] = \sum_{i=0}^{k-2} p^{-1} = (k-1) \cdot \ell^{1/k} \ .$$

Furthermore, the variables X_i are independent, which implies that for any $s > 0$, if Y is a binomial random variable with parameters s and p, then

$$\Pr[|\tilde{B}(v)| > s] = \Pr\left[\sum_{i=0}^{k-2} |B_i(v)| > s\right] \leq \Pr\left[\sum_{i=0}^{k-2} X_i > s\right] = \Pr[Y < k].$$

The final equality was observed in [16] (in the first s attempts there were at most $k-1$ successes). Setting $s = 16 \cdot \ell^{1/k} \cdot \ln n$, let $\mu = \mathbb{E}[Y] = s \cdot p = 16 \ln n$, and apply a Chernoff bound,

$$\Pr[Y < (1-\delta)\mu] < \exp(-\mu\delta^2/2) \ ,$$

with $\delta = 1/2$. Noticing that $k \leq \log \ell < 8 \cdot \ln n$, it follows that

$$\Pr[Y < k] \leq \Pr[Y < 8 \cdot \ln n] \leq \exp\left(-\frac{\mu}{8}\right) = \frac{1}{n^2}.$$

Applying the union bound over all vertices $v \in V$, we get that

$$\Pr[\exists v \in V : |\tilde{B}(v)| > 16 \cdot \ell^{1/k} \cdot \ln n] \leq n \cdot \frac{1}{n^2} = \frac{1}{n} \ . \qquad \lhd$$

A.2 Proof of Lemma 7

▶ **Lemma 7.** *Fix any $u, v \in V$, and let $0 \leq j \leq k-1$ be the smallest index such that $p_j(v) \in B(u)$. Then,*

$$\mathrm{dist}(v, p_j(v)) + \mathrm{dist}(u, p_j(v)) \leq (4k-3) \cdot \mathrm{dist}(u, v)$$

Proof. Denote $\Delta = \mathrm{dist}(u, v)$. We will prove by induction on $0 \leq i \leq j$, that

$$\mathrm{dist}(v, p_i(v)) \leq 2i \cdot \Delta \ . \tag{9}$$

For the base case $i = 0$, we have $p_0(v) = v$, so $\mathrm{dist}(v, p_0(v)) = 0$, which satisfies (9).

For the inductive step, assume that (9) holds for $0 \leq i < j$, and we prove for $i+1$. By the minimality of j, we have $p_i(v) \notin B(u)$, so by definition of a bunch,

$$\mathrm{dist}(u, p_{i+1}(u)) \leq dist(u, p_i(v)) \ . \tag{10}$$

Since $p_{i+1}(v)$ is the closest vertex to v in A_{i+1}, and using the triangle inequality, we have

$$\begin{aligned}
\mathrm{dist}(v, p_{i+1}(v)) &\leq \mathrm{dist}(v, p_{i+1}(u)) \\
&\leq \mathrm{dist}(v, u) + \mathrm{dist}(u, p_{i+1}(u)) \\
&\overset{(10)}{\leq} \Delta + \mathrm{dist}(u, p_i(v)) \\
&\leq \Delta + \mathrm{dist}(u, v) + \mathrm{dist}(v, p_i(v)) \\
&\overset{(9)}{\leq} 2\Delta + 2i \cdot \Delta \\
&= (2i+2) \cdot \Delta \ ,
\end{aligned}$$

which concludes the proof of (9). Note that by the triangle inequality, for any $0 \leq i \leq j$ we also have that

$$\mathrm{dist}(u, p_i(v)) \leq \mathrm{dist}(u, v) + \mathrm{dist}(v, p_i(v)) \overset{(9)}{\leq} \Delta + 2i \cdot \Delta = (2i+1) \cdot \Delta \tag{11}$$

Combining (9) and (11), and using that $j \leq k-1$, we conclude that

$$\mathrm{dist}(v, p_j(v)) + \mathrm{dist}(u, p_j(v)) \leq (2j + 2j + 1) \cdot \Delta = (4j+1) \cdot \Delta \leq (4k-3) \cdot \Delta \ . \qquad \blacktriangleleft$$

A.3 Proof of Lemma 12

▶ **Lemma 12.** *Fix any $u, v \in V$, and let $0 \leq i \leq k-1$ be the smallest index such that $p_i(v) \in B(u)$ or $p_i(u) \in B(v))$. Then:*

$$dist(v, p_i(v)) + dist(p_i(v), u) \leq (2i+1) \cdot dist(u, v)$$

Proof. Denote $\Delta = dist(u, v)$. We prove by induction on $0 \leq j \leq i$ that

$$dist(v, p_j(v)) \leq j \cdot \Delta \ . \tag{12}$$
$$dist(u, p_j(u)) \leq j \cdot \Delta \ . \tag{13}$$

For the base case $j = 0$, we have $p_0(v) = v$ and $p_0(u) = u$, so both $dist(v, p_0(v)) = dist(u, p_0(u)) = 0$, as required. For the inductive step, assume (12) and (13) hold for $0 \leq j < i$, and we prove for $j+1$. By the minimality of i, we have that both $p_j(v) \notin B(u)$ and $p_j(u) \notin B(v)$. By definition of a bunch, it follows that

$$dist(u, p_{j+1}(u)) \leq dist(u, p_j(v)) \ , \tag{14}$$

and

$$dist(v, p_{j+1}(v)) \leq dist(v, p_j(u)) \ , \tag{15}$$

By the induction hypothesis and the triangle inequality, it follows that

$$dist(v, p_{j+1}(v)) \overset{(15)}{\leq} dist(v, p_j(u)) \leq \mathrm{dist}(v, u) + dist(u, p_j(u)) \overset{(13)}{\leq} (j+1) \cdot \Delta \ ,$$

and symmetrically,

$$dist(u, p_{j+1}(u)) \overset{(14)}{\leq} dist(u, p_j(v)) \leq \mathrm{dist}(u, v) + dist(v, p_j(v)) \overset{(12)}{\leq} (j+1) \cdot \Delta \ ,$$

which completes the proof of (12) and (13). Finally, we conclude that

$$dist(v, p_i(v)) + dist(p_i(v), u) \leq dist(v, p_i(v)) + dist(p_i(v), v) + \Delta \leq (2i+1) \cdot \Delta \ . \qquad \blacktriangleleft$$

Exact Subquadratic Algorithm for Many-To-Many Matching on Planar Point Sets with Integer Coordinates

Seongbin Park
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Eunjin Oh
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Abstract

In this paper, we study the many-to-many matching problem on planar point sets with integer coordinates: Given two disjoint sets $R, B \subset [\Delta]^2$ with $|R|+|B| = n$, the goal is to select a set of edges between R and B so that every point is incident to at least one edge and the total Euclidean length is minimized. In the general case that R and B are point sets in the plane, the best-known algorithm for the many-to-many matching problem takes $\tilde{O}(n^2)$ time. We present an exact $\tilde{O}(n^{1.5} \log \Delta)$ time algorithm for point sets in $[\Delta]^2$. To the best of our knowledge, this is the first subquadratic exact algorithm for planar many-to-many matching under bounded integer coordinates.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry

Keywords and phrases Edge cover, many-to-many matching, similarity, geometric matching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.36

Related Version *Full Version*: https://arxiv.org/abs/2604.16921

Funding Supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00358505).

1 Introduction

Measuring the similarity between two point sets is a fundamental problem in computational geometry and pattern recognition [1, 22, 23]. A widely used measure is the *Earth Mover's Distance* (EMD), which captures the global distribution of point sets by finding an optimal correspondence [18, 19]. Formally, for two sets R and B of the same cardinality, the EMD is defined as the minimum cost of a perfect matching: $\min_{\sigma:R\to B} \sum_{r\in R} \|r - \sigma(r)\|$, where σ is a bijection. While EMD is effective at preserving the holistic structure of the sets, its definition is inherently restricted to cases where $|R| = |B|$. To accommodate sets of different cardinalities, one might consider *pointwise* heuristics such as the Chamfer distance: $\sum_{r\in R} \min_{b\in B} \|r - b\| + \sum_{b\in B} \min_{r\in R} \|b - r\|$. It provides a simple way to handle unequal set sizes as it treats each connection independently and sums $|R| + |B|$ distances without any global coordination. However, this often fails to capture the underlying structural correspondence, as it lacks the "assignment" nature that makes EMD robust.

A more principled approach to handling different sized sets, instead of resorting to pointwise measures, is to generalize the matching framework of EMD itself. By relaxing the strict one-to-one matching requirement to a *many-to-many* correspondence, we arrive at the *minimum link measure* introduced by [7]. Formally, a many-to-many matching is a set of edges $M \subseteq R \times B$ such that every point in $R \cup B$ is incident to at least one edge in M.

© Seongbin Park and Eunjin Oh;
licensed under Creative Commons License CC-BY 4.0

The cost is the total weight $\sum_{(r,b)\in M} \|r-b\|$. The minimum link measure is the minimum cost of a many-to-many matching between R and B. This "setwise" approach preserves the global structural integrity of EMD by seeking an optimal set of edges that covers both sets simultaneously. As a single edge in M can satisfy the coverage requirement for points in both R and B, the minimum link measure maintains a mutual correspondence that is more stable and geometrically intuitive than independent pointwise distances.

In this paper, we consider the problem of computing a minimum-cost many-to-many matching, also called the minimum link measure, between two point sets in Euclidean space. This problem was originally introduced by Eiter and Mannila [7]. While an efficient, optimal algorithm exists for points on a line, the problem becomes significantly more difficult in higher dimensions. Even for planar point sets, the best-known exact algorithm still suffers from a quadratic bottleneck [3], mirroring the challenges found in various other geometric bipartite matching problems [11]. To bridge this gap, we consider the case where the inputs lie on an *integer grid* as an intermediate step toward the general planar case.

Our results. In this paper, we present the first subquadratic-time exact algorithm for the many-to-many matching problem for point sets on an integer grid. Our main result is summarized in the following theorem.

▶ **Theorem 1.** *Given two disjoint sets*[1] R *and* B *of points in* $[\Delta]^2 = \{1, 2, \ldots, \Delta\} \times \{1, 2, \ldots, \Delta\}$*, we can compute a minimum-cost many-to-many matching between* R *and* B *in* $\tilde{O}(n^{1.5} \log \Delta)$ *time,*[2] *where* $n = |R| + |B|$.

A key technical challenge in achieving this bound is efficiently handling subproblems where points can be either matched to the opposite set or remain unmatched by paying a certain cost. To this end, we introduce the *minimum-cost matching with penalties* problem: Given two point sets R and B where each point $p \in R \cup B$ is associated with a real-valued penalty $\pi(p) \geq 0$, we seek a matching $M \subseteq R \times B$ that minimizes $\sum_{(r,b)\in M} \|r-b\| + \sum_{p\in \mathrm{free}(M)} \pi(p)$, where $\mathrm{free}(M)$ denotes the set of points in $R \cup B$ not incident to any edge in M. To the best of our knowledge, this problem has not been explicitly studied before. Although this can be solved in $O(n^3)$ time via standard reductions to the minimum-cost perfect matching problem for bipartite graphs, the cubic-time algorithm is too slow for our purpose. As a subroutine for our main algorithm, we provide an optimal algorithm for this penalty variant in one dimension, which we believe is of independent interest, stated as follows.

▶ **Theorem 2.** *Given two disjoint sets* R *and* B *of points on a real line, where each point has a real-valued penalty, a minimum-cost matching with penalties can be computed in* $O(n \log n)$ *time, where* $n = |R| + |B|$.

Related works. The study of the minimum-cost many-to-many matching problem was introduced by Eiter and Mannila [7]. They established that the problem is solvable in $O(n^3)$ time by reducing it to the minimum-cost perfect matching problem on graphs [7]. For the one-dimensional case where points lie on a real line, Colannino et al. [5] provided an optimal $O(n \log n)$-time algorithm, improving upon previous $O(n^2)$ results [6]. In the Euclidean plane, the best-known exact algorithm runs in $\tilde{O}(n^2)$ time [3]. On the other hand, in higher dimensions, no non-trivial algorithm is known for this problem while an $O(n^3)$-time algorithm

[1] With a slight modification, we can deal with the non-disjoint case without increasing the running time.
[2] The $\tilde{O}$-notation hides polylogarithmic factors in n.

can be obtained easily by reducing it to a minimum-cost perfect matching of a graph of complexity. Given the challenge of computing exact solutions in subquadratic time, several approximation algorithms have been explored. The Chamfer distance serves as a simple 2-approximation for the many-to-many matching cost [3]. Furthermore, $(1+\varepsilon)$-approximate solutions can be computed in $(1/\varepsilon)^{O(d)} \cdot n \log n$ time for d-dimensional Euclidean space, or any metric space with a constant doubling dimension [2, 4].

As EMD is a more classical setting, there are numerous results on computing the EMD between two points (also known as the minimum-cost perfect matching). By implementing the Hungarian algorithm [15] using a dynamic weighted nearest neighbor data structure [12], one can compute the exact EMD of two point sets in the plane in $\tilde{O}(n^2)$ time [11]. While the exact bipartite matching problem in the plane suffers from a quadratic bottleneck, the problem becomes easier if there are constraints on the input points. For instance, if the points in B and R are drawn independently and identically from a fixed distribution that is not known to the algorithm, the exact EMD between B and R can be computed in $O(n^{7/4} \log \Phi)$ time, where Φ denotes the spread of $B \cup R$ (i.e, the ratio of the maximum distance to the minimum distance). If B and R come from $[\Delta]^2$, Sharathkumar [20] presented an $\tilde{O}(n^{1.5} \log \Delta)$-time algorithm.[3]

2 Preliminaries and Algorithm Overview

In this section, we first describe an alternative view for the problem and then provide an overview of our algorithm. An alternative way is to view the many-to-many matching problem as a geometric graph problem. Consider a complete bipartite graph $G = ((R, B), E)$ with vertex set $V = R \cup B$ and with edge set $E = R \times B$ where the cost of an edge $(r, b) \in E$ is the Euclidean distance $\|r - b\|$ between their endpoints. An *edge cover* of G is a subset of E such that every vertex of V is incident to at least one edge of it. The cost of an edge cover of G is defined as the sum of the costs of its edges. Then the many-to-many matching problem on R and B is equivalent to the problem of finding an edge cover of G with the minimum cost.

For a general graph with n vertices and m edges, a minimum-cost edge cover can be solved in $O(n(n \log n + m))$ time via a reduction to the minimum-cost perfect matching [8, 9, 13]. Since the complete bipartite graph G has complexity $\Theta(n^2)$, we cannot use these graph algorithms directly to obtain a subquadratic-time algorithm. Nevertheless, as our approach extensively adapts several combinatorial principles from these classical algorithms, we introduce the following graph-theoretic terminology to be used throughout this paper.

Terminology for matching. Let H be a bipartite graph with edge costs. A *matching* M in H is a set of pairwise vertex-disjoint edges, and its cost defined as the sum of its edge costs. A matching M in H is *perfect* if every vertex is incident to exactly one edge of M. A vertex of the graph is *free* (with respect to M) if no edge of M is incident to it. An *alternating path* (with respect to M) is a path in the graph that begins from a free vertex and whose edges alternately belong to M and not to M. An *augmenting path* (with respect to M) is an alternating path whose two endpoints are both free. Augmenting M along an augmenting path P by taking symmetric difference $(M \oplus P)$ increases its cardinality by

[3] The paper [20] states that their algorithm takes $O(n^{1.5+\delta} \log(n\Delta))$ time for a constant δ, but by replacing a nearest neighbor data structure used in this paper with the most recent one [12], the factor n^{δ} can be replaced with $\operatorname{polylog} n$.

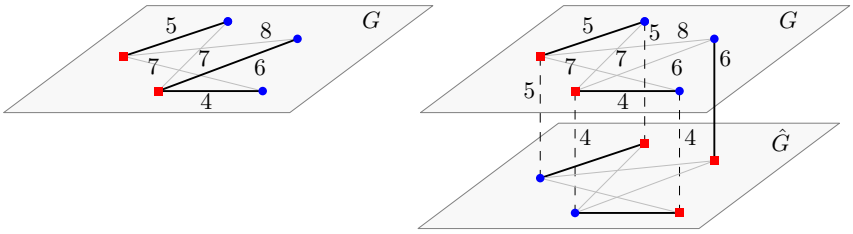


Figure 1 Illustration of the prism graph (right) of the original graph (left). The thick segments in the left figure show the min-cost many-to-many matching, and the thick segments in the right figure show the min-cost perfect matching.

one. A matching M has maximum cardinality if no augmenting path exists. Classically, maximum-cardinality (or minimum-cost) matchings are computed by repeatedly augmenting M along paths with certain properties until none remain.

2.1 Reduction to the Min-Cost Perfect Matching

In this section, we describe a reduction from the minimum-cost many-to-many matching problem to the minimum-cost perfect matching problem. While this is identical to the folklore reduction, we refer to the resulting graph as the *prism graph* for notational convenience. It preserves the original geometric embedding and induces independent vertex penalties.

Let R and B be two point sets in $[\Delta]^2$, and let G be the complete bipartite graph between R and B, where the cost of each edge is the Euclidean distance between its endpoints. The reduced instance is called the *prism graph* of G, denoted by $\tilde{G} = (\tilde{V}, \tilde{E})$, with an associated edge cost $\tilde{c} : \tilde{E} \to \mathbb{R}$.

Prism graph. The prism graph $\tilde{G}$ is a two-layered bipartite graph consisting of an "upper" layer, which is the original graph G, and a "lower" layer, which is a mirrored copy $\hat{G}$ of G. For each vertex v in the upper layer, we add a *link edge* between v and its corresponding copy $\hat{v}$ in the lower layer. Note that $\tilde{G}$ remains bipartite, as its vertex set can be partitioned into $R \cup \{\hat{v} \mid \hat{v} \text{ is the copy of } v \in B\}$ and $B \cup \{\hat{v} \mid \hat{v} \text{ is the copy of } v \in R\}$. See Figure 1. For an edge $e = (u, v)$ in $\tilde{G}$, the cost $\tilde{c}(e)$ is defined as follows:

- If u and v are both in the upper layer, $\tilde{c}(e) = \|u - v\|$.
- If u and v are both in the lower layer, $\tilde{c}(e) = 0$.
- If e is a link edge $(v, \hat{v})$, $\tilde{c}(e) = \mu(v)$, where $\mu(v)$ is the distance from v to its nearest neighbor in the opposite set (R or B).

Note that an optimal edge cover M of G consists of disjoint stars. For a star of M of size larger than two, let c be the center of the star. Then for every vertex of the star other than c, its closest neighbor in the opposite set is c. Otherwise, we can reconnect it with its closest neighbor without violating the feasibility of the solution. This reduces the problem into the minimum-cost matching problem with penalties where the penalty of each point is $\mu(\cdot)$. The construction of the prism graph is based on this intuition. Matching a vertex via a link edge to the lower layer simulates leaving that vertex uncovered by the primary matching in the upper layer, instead paying the penalty of connecting it to its nearest neighbor.

The values $\mu(v)$ for all $v \in R \cup B$ can be computed in $O(n \log n)$ time by constructing the Voronoi diagrams of R and B and using a planar point location data structure [14]. Although $\tilde{G}$ has $\Theta(n^2)$ edges in the worst case, we can store it using complexity of $O(n)$ by maintaining only the vertices and the link edges explicitly. The edges in G and $\hat{G}$ can be represented implicitly as they form complete bipartite graphs.

The following lemma establishes the equivalence between the minimum-cost edge cover of G and the minimum-cost perfect matching of $\tilde{G}$. For illustration, see Figure 1.

▶ **Lemma 3** ([3]). *Given a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$, a minimum-cost edge cover of G can be computed in $O(n)$ time.*

Due to the lack of space, some proofs are omitted. The missing proofs can be found in the full version of this paper.

2.2 Overview of Our Algorithm

Due to Lemma 3, the problem reduces to computing a minimum-cost perfect matching of $\tilde{G}$. The algorithm consists of two phases.

In the first phase, we compute an approximate perfect matching of $\tilde{G}$ with its corresponding dual weights. We use the approximation algorithm of Bandyapadhyay et al. [3] which is based on the scaling algorithm [10], with dynamic additively weighted nearest-neighbor data structures [12] to obtain these dual weights in $\tilde{O}(n^{1.5} \log \Delta)$ time.

In the second phase, we leverage these dual weights to construct a sparse candidate subgraph $\tilde{G}_{\mathsf{cand}}$ of $\tilde{G}$ containing an optimal perfect matching of $\tilde{G}$. Here, a crucial property is that it is *almost* planar. Specifically, the upper layer of $\tilde{G}_{\mathsf{cand}}$ possesses a planar *skeleton*, meaning no two edges of $\tilde{G}_{\mathsf{cand}}$ cross unless their endpoints are collinear, and the lower layer of $\tilde{G}_{\mathsf{cand}}$ is a mirrored copy of its upper layer. We then compute the exact minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$ by applying a divide-and-conquer strategy on a balanced separator of this planar skeleton.

While this overall strategy comes from the subquadratic-time algorithm for the one-to-one setting on integer grids [20], we address two major technical hurdles unique to the many-to-many setting.

- During the construction of $\tilde{G}_{\mathsf{cand}}$ and within the recursion step, we must efficiently solve subproblems where points can either be matched to the opposite set or left unmatched by paying a penalty. To handle this, we develop an $O(n \log n)$-time algorithm for the 1D minimum-cost matching problem with penalties (Section 3), replacing the simple greedy approach used for standard 1D perfect matching.
- In the one-to-one setting [20], the candidate graph $\tilde{G}_{\mathsf{cand}}$ is strictly planar, allowing direct application of the separator-based algorithm in [17]. In our case, $\tilde{G}_{\mathsf{cand}}$ is not strictly planar. We overcome this by applying the balanced separator exclusively to the planar skeleton of $\tilde{G}_{\mathsf{cand}}$ (Section 4.2). A tricky part is to handle the vertices of $\tilde{G}_{\mathsf{cand}}$ not appearing on the planar skeleton. We address this using their 1D nature: these vertices are fully contained in the open intervals (edges) of the planar skeleton, which allows us to process them efficiently using our 1D matching subroutine.

In the following, some proofs and details are omitted due to page limits. All missing proofs and details can be found in the full version of this paper.

3 Subroutine: 1D Minimum Cost Bipartite Matching with Penalties

In this section, we prove Theorem 2, that is, we present an $O(n \log n)$-time algorithm for the 1D minimum-cost bipartite matching problem with penalties. As input, we are given two sets B and R lying on a horizontal line where each point $p \in B \cup R$ is associated with a penalty w_p. For a matching $M \subseteq B \times R$ between B and R, its cost $c(M)$ is defined as $\sum_{(r,b)\in M} \|r - b\| + \sum_{p\in(R\cup B)\setminus V(M)} w_p$. That is, its cost is the total edge length of M plus the total penalty of the unmatched vertices. The goal is to compute a matching between R and B of minimum total cost.

To the best of our knowledge, there is no work explicitly studying the minimum-cost bipartite matching problem with penalties even for the one-dimensional case. However, there are several ways to address this problem. First, this problem reduces to the min-cost flow problem on planar directed graphs. Also, if points of $B \cup R$ have integer coordinates, and the penalties w_p are all integers, then the resulting instance has integer costs. In this case, we can solve the problem in $\tilde{O}(n \log N)$ time, where N is the maximum of the maximum cost and the diameter of $B \cup R$. However, the penalty w_p can be an arbitrary real number in our case, which makes the aforementioned algorithm inapplicable in our setting. Another simple way is to use dynamic programming, which leads to an $O(n^2)$-time algorithm. Since our main algorithm is based on this dynamic programming algorithm, we also describe it here.

3.1 Quadratic-Time DP Algorithm

Let ℓ be the line containing B and R. Without loss of generality, assume that ℓ is horizontal. For a point $x \in \ell$, let B_x and R_x be the sets of points in B and R, respectively, lying to the left of x (including the points lying on x). For any point $x \in \ell$ and a positive integer k, we let $F_x(k)$ denote the minimum matching cost with penalties between $\bar{B}_x$ and R_x, where $\bar{B}_x$ is the set obtained from B_x by adding k points at x with penalty ∞. For a negative integer k, we let $F_x(k)$ denote the minimum matching cost with penalties between B_x and $\bar{R}_x$, where $\bar{R}_x$ is the set obtained from R_x by adding $|k|$ points at x with penalty ∞. Then the minimum matching cost between B and R is $F_\infty(0)$. Therefore, the problem reduces to computing $F_x(\cdot)$ for all points x in ℓ and for all values k with $-n \le k \le n$.

▶ **Lemma 4.** *For any value k and any two points x and x' of ℓ such that no point of $R \cup B$ lies between x and x' (including x and x'), we have $F_x(k) = F_{x'}(k) + |k| \cdot \|x - x'\|$.*

Thus it is sufficient to compute $F_p(k)$ for all points $p \in R \cup B$ and all values $-n \le k \le n$. But to avoid a degeneracy issue, for each point $p \in R \cup B$, we compute $F_{\bar{p}}(k)$ for a conceptual point $\bar{p}$ located infinitesimally to the right of p. For all distance calculations, we treat $\|p - \bar{p}\| = 0$. As initialization, let x_0 be any point lying strictly to the left of the first point in $B \cup R$. Then $F_{x_0}(0) = 0$ and $F_{x_0}(k) = \infty$ for any $k \neq 0$. Starting from this base case, we process the points of $R \cup B$ from left to right along ℓ. We show how to handle a point $p \in R \cup B$. Without loss of generality, assume that $p \in R$. The other case can be handled symmetrically. Let p' be the point of $B \cup R$ lying immediately to the left of p along ℓ, and let $d = \|p - p'\|$. Then we have the following recurrence relation. There are two possibilities: either p is matched with a blue point in $\bar{B}_{\bar{p}}$, or p is not used in the matching by paying the penalty. We consider both cases, and take the one with minimum cost.

$$F_{\bar{p}}(k) = \min\Big(\underbrace{F_{\bar{p}'}(k-1) + |k-1|d}_{\text{Match } p}, \quad \underbrace{F_{\bar{p}'}(k) + |k|d + w_p}_{\text{Pay Penalty}}\Big) \tag{1}$$

The recurrence relation immediately shows how to compute $F_{\bar{p}}(\cdot)$ for all points $p \in B \cup R$. Each table entry can be computed in $O(1)$ time, yielding an overall running time of $O(n^2)$.

In the following, to make the description easier, we simply let $F_p(k) := F_{\bar{p}}(k)$ as the following analysis is based only on the recurrence relation (1).

3.2 Convexity of the Cost Function

In this subsection, we show that the function $F_x(\cdot)$ is *convex* for any fixed point $x \in \ell$. We say $f : \mathbb{Z} \to \mathbb{R}$ is *convex* if $f(i-1) + f(i+1) \ge 2f(i)$ for every integer i. If $f(i+1) = \infty$ or $f(i-1) = \infty$, the inequality holds immediately. We let $\mathrm{dom}(f)$ be the set of integers k such that $f(k)$ is finite. Let $\Delta f(k) := f(k) - f(k-1)$.

▶ **Lemma 5.** *The function $F_x(\cdot)$ is convex for any fixed point $x \in R \cup B$.*

Proof. We use induction on the points of $B \cup R$ encountered from left to right. Let x_0 be a point to the left of the leftmost point of $B \cup R$. We define $F_{x_0}(0) = 0$ and $F_{x_0}(k) = \infty$ for $k \neq 0$, which is trivially convex.

Let $p \in B \cup R$ be the current point and p' be the previous point in $B \cup R \cup \{x_0\}$. Let $d = \|p - p'\|$. Without loss of generality, assume that $p \in B$. By the induction hypothesis, $F_{p'}(\cdot)$ is convex. Note that $|k-1|d$ is a convex function of k. Since the sum of two convex functions is also convex, $F_{p'}(k-1) + |k-1|d$ is convex. Similarly, $F_{p'}(k) + |k|d + w_p$ is convex. Although the minimum of two convex functions is not convex in general, it is convex in our setting. This holds due to the following claim with $f(k) = F_{p'}(k) + |k|d$.

▷ Claim 6. Let $f : \mathbb{Z} \to \mathbb{R}$ be a convex function. Then $F_p(k) := \min\{f(k-1), f(k) + w\}$ is also convex for any value w.

Therefore, the lemma holds for any fixed point $x \in R \cup B$. ◀

As we showed in the proof of Claim 6, the following corollary holds.

▶ **Corollary 7.** *Let k^* be the largest integer with $\Delta f(k) \leq -w_p$. Then for $k \leq k^*$, we have $F_p(k) = f(k) + w_p$, and for $k > k^*$, we have $F_p(k) = f(k-1)$.*

3.3 Near-Linear-Time Algorithm

In this subsection, we present an $O(n \log n)$-time algorithm for the 1D minimum-cost bipartite matching problem with penalties. For each $p \in B \cup R$, let $\mathrm{dom}(F_p) = [L_p, R_p]$ be the maximal interval of integers for which $F_p(k) < \infty$. By the recurrence, $\mathrm{dom}(F_x)$ is always a contiguous interval. Basically, we compute $F_p(k)$ for all integers $k \in \mathrm{dom}(F_p)$ and all points $p \in B \cup R$. By maintaining $F_p(k)$ using a data structure for a fixed p, we show that $F_p(k)$ can be computed in $O(\log n)$ time for all integers k for a fixed p assuming that we have $F_{p'}(\cdot)$, where p' is the point of $B \cup R$ lying immediately to the left of p.

Representation of the cost function. Fix $x \in \ell$, and let $\Delta F_x(k) := F_x(k) - F_x(k-1)$. We maintain $F_x(0)$ and $\Delta F_x(k)$ for all indices k. Note that they fully represent $F_x(\cdot)$. To make the update efficiently, we store $\Delta F_x(\cdot)$ using two balanced binary search trees. Let H_x^+ and H_x^- be two sets where

$$H_x^+ = \{\Delta F_x(k) \mid 0 < k \leq R_x\} \text{ and } H_x^- = \{\Delta F_x(k) \mid L_x < k \leq 0\}.$$

Since $F_x(\cdot)$ is convex, $\Delta F_x(k)$ is non-decreasing with respect to k. We maintain the two sets using two binary search trees T_x^+ and T_x^- and offsets o^+ and o^-. The offsets allow us to shift all stored values uniformly without updating each element explicitly. Each leaf of a tree of T_x^+ (and T_x^-) corresponds to exactly one index $k > 0$ (and $k < 0$), and the leaf stores the value $\Delta F_x(k) - o^+$ (and $\Delta F_x(k) - o^-$). Thus, the actual value of $\Delta F_x(k)$ is obtained by adding the corresponding offset o^+ or o^-. Each internal node additionally stores the size of its subtree and the sum of the stored values in its subtree (excluding the offset). Hence, the true sum of the values in any subtree can be obtained by adding the offset multiplied by the subtree size. The leaves of T_x^+ (and T_x^-) are stored in increasing order of their indices k. Since $F_x(\cdot)$ is convex, the stored values are non-decreasing along this order. Therefore, T_x^+ and T_x^- can be implemented as binary search trees augmented with subtree sizes and subtree sums, supporting insertion, deletion, and prefix-sum queries in $O(\log n)$ time. We call the tuple of $F_x(0)$, o^+, o^-, T_x^+ and T_x^- the *representation* of $F_x(\cdot)$. Here, since the roots store the size of the trees, we do not need to explicitly maintain $\mathrm{dom}(F_p)$.

▶ **Observation 8.** *Given the representation of $F_x(\cdot)$, we can compute $F_x(k)$ for any integer k in $O(\log n)$ time.*

Proof. We show how to compute $F_x(k)$ for any integer $k > 0$. For $k = 0$, we maintain $F_x(0)$ explicitly, and for $k < 0$, we can do this symmetrically. By telescoping, we have

$$F_x(k) = F_x(0) + \sum_{i=1}^{k} \Delta F_x(i).$$

Thus, it suffices to compute the prefix sum of the first k values in H_x^+.

Since $\Delta F_x(k)$ is non-decreasing in k, the values stored in T_x^+ are already ordered by index. We augment each node of T_x^+ with the size of its subtree and the sum of the stored values in its subtree (excluding the offset o^+). Using standard binary search tree operations, we can retrieve the sum of the smallest k elements in T_x^+ in $O(\log n)$ time. Adding this sum and the accumulated offset $k \cdot o^+$ to $F_x(0)$ yields $F_x(k)$. Therefore, given the representation of $F_x(\cdot)$, we can compute $F_x(k)$ for any integer k in $O(\log n)$ time. ◀

Update of the representation. Assume that we are given the representation of $F_{p'}(\cdot)$, and we wish to compute the representation of $F_p(\cdot)$, where p is the next point to the right of p' on ℓ. We describe the case $p \in B$; the case $p \in R$ is symmetric. Let $d := \|p - p'\|$.

Step 1. We first consider the transportation cost. Let $f(k) := F_{p'}(k) + |k|d$. Thus, before handling the matching/penalty decision at p, we first transform the representation of $F_{p'}$ into the representation of f. For $k > 0$, we have $\Delta f(k) = \Delta F_{p'}(k) + d$, and for $k \leq 0$, we have $\Delta f(k) = \Delta F_{p'}(k) - d$. Hence, this update can be performed by increasing the offset o^+ by d, and decreasing the offset o^- by d, while keeping the trees unchanged. Also, $F_{p'}(0)$ remains unchanged since $|0|d = 0$. After this step, the representation corresponds to $f(\cdot)$.

Step 2. We then find the largest integer k^* with $\Delta f(k) \leq -w_p$ by a binary search over the ordered trees $T_{p'}^+$ and $T_{p'}^-$ in $O(\log n)$ time. Since $\Delta f(k)$ is non-decreasing in k, the threshold k^* can be found by searching for the largest index whose discrete derivative is at most $-w_p$. Recall that $F_p(k) = \min\{f(k-1),\, f(k) + w_p\}$ from (1). By Corollary 7, for $k \leq k^*$, we have $F_p(k) = f(k) + w_p$, and for $k > k^*$, we have $F_p(k) = f(k-1)$.

Step 3. We now update the representation of $f(\cdot)$ to obtain that of $F_p(\cdot)$. We first compute $F_p(0)$. Since $F_p(0) = \min\{f(-1),\, f(0) + w_p\}$, this value can be computed in $O(\log n)$ time using Observation 8.

We next determine the new discrete derivatives. For $k \leq k^*$, both $F_p(k)$ and $F_p(k-1)$ lie in the same case, and hence

$$\Delta F_p(k) = F_p(k) - F_p(k-1) = f(k) - f(k-1) = \Delta f(k).$$

For $k > k^* + 1$, we have

$$\Delta F_p(k) = F_p(k) - F_p(k-1) = f(k-1) - f(k-2) = \Delta f(k-1).$$

At the boundary $k = k^* + 1$, we have

$$\Delta F_p(k^* + 1) = F_p(k^* + 1) - F_p(k^*) = f(k^*) - (f(k^*) + w_p) = -w_p.$$

Therefore, the sequence $\Delta F_p(\cdot)$ is obtained from $\Delta f(\cdot)$ by inserting the single value $-w_p$ (minus the corresponding offset) at position $k^* + 1$. As a result, all indices strictly larger than k^* are shifted by one position to the right.

Since the discrete derivatives are stored in a binary search tree, this transformation can be implemented by a single rank-based insertion at index $k^* + 1$. If $k^* < 0$, the insertion shifts the element originally at index 0 to index 1. Because index 0 belongs to $T^-_{p'}$ and index 1 belongs to $T^+_{p'}$, we must extract the maximum element from $T^-_{p'}$ and insert it as the minimum element in $T^+_{p'}$ (after adjusting the values with respect to the offsets for the two trees). All subtree sizes and subtree sums are updated along the search path. The insertion and any necessary boundary shifts take $O(\log n)$ time. Thus, the entire update from the representation of $F_{p'}(\cdot)$ to that of $F_p(\cdot)$ takes $O(\log n)$ time.

▶ **Lemma 9.** *Given the representation of $F_{p'}(\cdot)$, we can compute the representation of $F_p(\cdot)$ in $O(\log n)$ time.*

We consider the points of $B \cup R$ from left to right. The initialization takes $O(n)$ time, and each update takes $O(\log n)$ time. Therefore, the total running time is $O(n \log n)$.

4 Minimum-Cost Perfect Matching for Prism Graphs

In this section, we present an exact algorithm for computing a minimum-cost perfect matching of the prism graph $\tilde{G}$. To simplify the exposition, we identify the vertices in the *upper layer* of the prism graph $\tilde{G}$ directly with the original points in R and B. Specifically, we let the upper layer vertex set be $R \cup B$. We then define the *lower layer* as a mirrored copy of the upper layer, where each vertex $v \in R \cup B$ has a corresponding copy $\hat{v}$ in the lower layer.

Our strategy involves two main phases: computing an approximate minimum-cost perfect matching of $\tilde{G}$ along with its associated dual weights, and utilizing these dual weights to derive the exact solution. We begin by establishing the properties of an approximate solution and the associated dual weights.

1-optimality. Given a *scaling factor* $\theta > 0$, let $\tilde{G}_\theta = (\tilde{V}, \tilde{E})$ denote the *θ-scaled graph* of $\tilde{G}$ with the edge-cost function $\tilde{c}_\theta : \tilde{E} \to \mathbb{Z}_{\geq 0}$, defined as $\tilde{c}_\theta(e) = \lceil \tilde{c}(e)/\theta \rceil$ for all $e \in \tilde{E}$. Let $y(v)$ denote the *dual weight* of a vertex $v \in \tilde{V}$. A matching $\tilde{M}$ in $\tilde{G}_\theta$ and the associated dual weights $y(\cdot)$ are said to be *1-feasible* if

$$y(u) + y(v) \leq \tilde{c}_\theta(e) + 1 \quad \text{for all } e = (u, v) \in \tilde{E}, \tag{2}$$

$$y(u) + y(v) = \tilde{c}_\theta(e) \quad \text{for all } e = (u, v) \in \tilde{M}. \tag{3}$$

We refer to (2) and (3) as the *1-feasibility conditions*. An edge $e = (u, v) \in \tilde{E} \setminus \tilde{M}$ is called *admissible* if $y(u) + y(v) = \tilde{c}_\theta(e) + 1$. The *admissible graph* of $\tilde{G}_\theta$ is the subgraph of $\tilde{G}_\theta$ consisting of all admissible edges and the edges in $\tilde{M}$. If a matching $\tilde{M}$ and dual weights $y(\cdot)$ satisfy the 1-feasibility conditions and $\tilde{M}$ is a perfect matching, we call the pair $(\tilde{M}, y)$ *1-optimal*. Sharathkumar and Agarwal [21] showed that a 1-optimal matching of $\tilde{G}_\theta$ provides an approximate minimum-cost perfect matching of $\tilde{G}$ with an additive error ε of at most $3n\theta$.

We now outline our algorithm for computing an exact minimum-cost perfect matching of $\tilde{G}$. First, we compute a 1-optimal matching $\tilde{M}^*_1$ and associated dual weights $y(\cdot)$ of $\tilde{G}_\theta$ using a scaling factor of $\theta = 1/(n\Delta^{33})$, following the procedure described in the full version of this paper. In Section 4.1, we utilize these dual weights to identify a set of *eligible edges* E_{elig} of the upper layer of $\tilde{G}$ and exploit their geometric properties to construct a set P of non-crossing line segments having endpoints at $B \cup R$. Using P, we systematically extract a

sparse candidate subgraph $\tilde{G}_{\mathsf{cand}}$ of size $O(n)$ that is guaranteed to contain a minimum-cost perfect matching of $\tilde{G}$. Finally, Section 4.2 presents a divide-and-conquer algorithm to compute an exact optimal solution in $\tilde{G}_{\mathsf{cand}}$ by applying the planar separator theorem to the graph induced by P.

4.1 Construction of the Candidate Subgraph

We construct a *candidate subgraph* $\tilde{G}_{\mathsf{cand}} = (\tilde{V}, \tilde{E}_{\mathsf{cand}})$ of $\tilde{G}$ that is guaranteed to contain an optimal matching of $\tilde{G}$. We begin by utilizing the 1-optimal matching $\tilde{M}_1^*$ and the 1-optimal dual weights of $\tilde{G}_\theta$ with a scaling factor of $\theta = 1/(n\Delta^{33})$ to characterize the *eligible edges* in the upper layer that can participate in an optimal solution. This characterization reveals a crucial geometric property of these edges. Using this property, we apply the technique of Sharathkumar [20] to construct a planar skeleton (planar straight-line graph) P. Finally, we use P to generate the set of *candidate edges* $\tilde{E}_{\mathsf{cand}}$.

The overall construction comes from Sharathkumar [20]: It applies the approach to a Euclidean complete bipartite graph while we apply it to the upper layer of the prism graph $\tilde{G}$. As the upper layer is a Euclidean complete bipartite graph, the analysis immediately follows from [20], but the detailed implementation should be tailored for our setting.

Eligible edge decomposition and its induced planar skeleton. We define the *scaled dual weight* of a vertex $v \in \tilde{V}$ as $y_\theta(v) = \theta \cdot y(v)$. The following lemma establishes a lower bound on the sum of scaled dual weights for the edges participating in an optimal matching of $\tilde{G}$.

▶ **Lemma 10.** *Let $y_\theta(\cdot)$ be the scaled dual weights associated with a 1-optimal matching of $\tilde{G}_\theta$ using the scaling factor $\theta = 1/(n\Delta^{33})$. Let $\tilde{M}^*$ be a minimum-cost perfect matching of $\tilde{G}$. Then, for every edge $(u, v) \in \tilde{M}^*$ in the upper layer,*

$$y_\theta(u) + y_\theta(v) > \|u - v\| - \frac{2}{\Delta^{33}}. \tag{4}$$

The proof is same as the argument in [20] and is thus omitted. We define the set of *eligible edges*, denoted by E_{elig}, as the set of all edges in the upper layer of $\tilde{G}$ satisfying the condition (4). Lemma 10 implies that for any optimal matching $\tilde{M}^*$ of $\tilde{G}$, all edges of $\tilde{M}^*$ that belong to the upper layer are contained in E_{elig}. Therefore, even if we remove all non-eligible edges in the upper layer from the prism graph, the optimal solution remains the same.

The following lemma shows that if no four vertices of $R \cup B$ are collinear, then the graph obtained from the upper layer of the prism graph by removing all non-eligible edges is planar. Thus our main focus is to handle collinear points in the following.

▶ **Lemma 11** ([20])**.** *Let $Q \subset [\Delta]^2$ be a set of points. Let $a, b, c, d \in Q$ be four distinct points that are not collinear. If the line segments $\overline{ab}$ and $\overline{cd}$ intersect, then*

$$\|a - d\| + \|b - c\| + \frac{1}{\Delta^{32}} < \|a - b\| + \|c - d\|.$$

Therefore, for any two distinct edges (a, b) and (c, d) in E_{elig} such that their four endpoints are not collinear, the line segments $\overline{ab}$ and $\overline{cd}$ do not intersect.

We show that the non-crossing property enables us to compute an *eligible edge decomposition* $P = \{(p_1, q_1), \ldots, (p_k, q_k)\}$, which is a set of non-crossing line segments, such that for every edge of E_{elig}, it is contained in a segment of P. Note that P induces a plane graph whose vertices come from $R \cup B$.

Since the definition of eligible edges in our setting is identical to that in Sharathkumar [20], we can adopt their algorithm for computing the eligible edge decomposition. However, a minor technical hurdle arises in the initialization phase. It uses an approximate *perfect* matching M obtained from the first phase, but in our setting, an approximate matching we have is not necessarily perfect if we look at the upper layer only. More specifically, it first computes all point-line pairs (p, ℓ) with $p \in R \cup B$ such that ℓ contains an eligible edge incident to p. The number of such pairs is $O(n)$. For a point $p \in R \cup B$, one can compute all eligible edges incident to it in $O(\text{polylog}\, n)$ time per edge using a dynamic weighted nearest neighbor data structure. However, it is possible that a single line ℓ contains many different eligible edges incident to p. Then a single line can be discovered multiple times, which leads to quadratic running time in the worst case. To avoid this, as initialization, it computes just one line ℓ for each point p in $R \cup B$ containing an eligible edge incident to p in $O(n \,\text{polylog}\, n)$ time in total using M. Later, it considers each point p in $B \cup R$ one by one and computes all pairs (p, ℓ) such that ℓ is not found in the initialization. Even in this case, a single line ℓ can be discovered multiple times, but the total number of pairs (p, ℓ) found in this way is $O(n)$, which leads to near-linear running time. The perfect matching M can be used in the initialization; since an edge (u, v) of M is eligible, its extension contains an eligible edge incident to u (and v). But we can perform this initialization using a dynamic weighted nearest-neighbor data structure, and this does not increase the overall running time. The full details of this modified construction are deferred to the full version of this paper.

Construction of the candidate subgraph. We now construct the candidate edge set $\tilde{E}_{\mathsf{cand}}$ of size $O(n)$ using the eligible edge decomposition P. As mentioned earlier, if no four points of $R \cup B$ are collinear, we can simply construct $\tilde{E}_{\mathsf{cand}}$ by adding all eligible edges (the edges of P), their mirrored edges in the lower layer, and all link edges between the layers. However, in the general case, the number of eligible edges can be $\Theta(n^2)$, making it impossible to include all of them.

Instead, we select a restricted subset of eligible edges as follows. For each pair $(p, q) \in P$, let I be the set of points in $R \cup B$ lying on the open segment between p and q. For a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$, any edge incident to a vertex $v \in I$ must have its other endpoint in $I \cup \{\hat{v}, p, q\}$. Consequently, for the endpoints p and q, there are four possible configurations based on whether p and q are matched with points in I. For each configuration, we can identify the specific edges of $\tilde{M}^*$ incident to points in I in $O(|I| \log |I|)$ time without computing the entire matching. More specifically, let $I' \subseteq \{p, q\}$ be the set of points matched with elements in I under $\tilde{M}^*$. By solving the local matching problem for $I \cup I'$, we extract the necessary edges to be included in $\tilde{E}_{\mathsf{cand}}$, ensuring the total size remains $O(n)$. For the local matching problem, since the vertices of $I \cup I'$ are collinear, the problem is restricted to one dimension. This problem reduces to the minimum-cost matching problem on $I \cup I'$, where leaving a vertex unmatched incurs a penalty of $\mu'(\cdot)$, where $\mu'(v) = \mu(v)$ for a vertex $v \in I$ and $\mu'(v) = \infty$ for a vertex $v \in I'$. Therefore, the local matching problem can be solved in $O(|I| \log |I|)$ time by Theorem 2. We do this for all the four configurations, and then take the union of the resulting edges. Then $\tilde{E}_{\mathsf{cand}}$ is defined as the union of all such edges in P, their mirrored edges in the lower layer, and all the link edges. Then the size of $\tilde{E}_{\mathsf{cand}}$ is $O(n)$, and it can be computed in $O(n \log n)$ time.

We show that $\tilde{E}_{\mathsf{cand}}$ contains a minimum-cost perfect matching of $\tilde{G}$. For this, we use the following technical lemma justifying that focusing on the upper layer during the construction of $\tilde{E}_{\mathsf{cand}}$ is sufficient. We say a matching M is *symmetric* if an edge (u, v) in the upper layer is contained in M if and only if its mirrored edge $(\hat{u}, \hat{v})$ is contained in M.

▶ **Lemma 12.** *Let $\tilde{H}$ be a subgraph of $\tilde{G}$ induced by edges in the upper layer, their mirrored edges in the lower layer, and all link edges. There exists a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{H}$ that is symmetric.*

Proof. Let $\tilde{M}^*$ be an arbitrary minimum-cost perfect matching of $\tilde{H}$. Let m be the number of edges in $\tilde{M}^*$ that belong to the upper layer. Each such edge (u, v) covers one vertex in R and one in B. Because $\tilde{M}^*$ is a perfect matching, any vertex in the upper layer not matched with a vertex in the upper layer must be matched to its corresponding copy in the lower layer via a link edge. This implies that exactly m vertices in the lower layer are not covered by link edges. To satisfy the perfect matching requirement, these m vertices must be matched to each other using $m/2$ edges in the lower layer.

Now, consider the cost of these $m/2$ edges. By construction, the cost of any edge in the lower layer is zero. Therefore, we can replace the existing lower-layer edges in $\tilde{M}^*$ with the "mirrored" edges of the upper-layer matching without changing the total cost of the matching. This substitution results in a new perfect matching M' that remains optimal and satisfies the symmetry condition: (u, v) is in M' if and only if $(\hat{u}, \hat{v})$ is in M'. Therefore, the lemma holds. ◀

▶ **Lemma 13.** *There exists a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$ such that $\tilde{M}^* \subseteq \tilde{E}_{\mathsf{cand}}$.*

Proof. Let $\tilde{M}^*$ be a minimum-cost perfect matching of $\tilde{G}$ such that (i) all edges of $\tilde{M}^*$ belonging to the upper layer are contained in E_{elig}, and (ii) $\tilde{M}^*$ is symmetric, which always exists by Lemma 10 and by Lemma 12.

To show that $\tilde{M}^* \subseteq \tilde{E}_{\mathsf{cand}}$, we consider the eligible edge decomposition P. Every edge $(u, v) \in \tilde{M}^*$ in the upper layer must be contained within some segment $\overline{pq} \in P$. For each such segment, let I be the set of points of $B \cup R$ lying in the open interval between p and q. By the construction of P, any edge in $\tilde{M}^*$ incident to a vertex in I must have its other endpoint in $I \cup \{\hat{v}, p, q\}$. This local constraint implies that the matching within each segment $\overline{pq}$ can be optimized independently based on the four possible configurations of the endpoints p and q. Since $\tilde{E}_{\mathsf{cand}}$ is constructed by taking the union of the optimal local matchings for all four configurations across all segments in P, it follows that the edges of $\tilde{M}^*$ are included in $\tilde{E}_{\mathsf{cand}}$. Thus, the subgraph of $\tilde{G}$ induced by $\tilde{E}_{\mathsf{cand}}$ contains a minimum-cost perfect matching of $\tilde{G}$. ◀

Therefore, the candidate edge set $\tilde{E}_{\mathsf{cand}}$ can be computed in $\tilde{O}(n)$ time in total, and the problem reduces to computing a minimum-cost perfect matching on the subgraph $\tilde{G}_{\mathsf{cand}}$ of $\tilde{G}$ induced by $\tilde{E}_{\mathsf{cand}}$. For an illustration of $\tilde{G}_{\mathsf{cand}}$, see Figure 2.

It is worth noting the structural distinction between our construction and the one-to-one setting addressed by Sharathkumar [20]. In the one-to-one case, for each segment $(p, q) \in P$, it is sufficient to consider at most *two* configurations of the endpoints p and q, which makes the resulting graph planar. This is because the difference between the numbers of points in R and of points in B restricts the possible configurations of p and q. In the many-to-many setting, however, this difference does not inherently reduce the number of configurations. Consequently, we must consider the union of local optimal matchings for all *four* configurations, which renders the resulting candidate graph $\tilde{G}_{\mathsf{cand}}$ non-planar.

4.2 Divide-and-Conquer Algorithm via Planar Separators

In this section, we present a divide-and-conquer algorithm to compute a minimum-cost perfect matching of the candidate graph $\tilde{G}_{\mathsf{cand}} = (\tilde{V}, \tilde{E}_{\mathsf{cand}})$. Our algorithm recursively partitions $\tilde{G}_{\mathsf{cand}}$ into disjoint vertex sets by removing a subset of vertices. We first compute

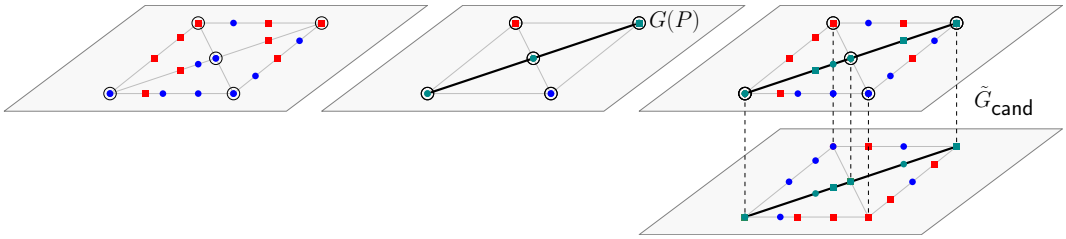


Figure 2 Illustration of the upper layer of $\tilde{G}_{\mathsf{cand}}$ (left), the planar skeleton $G(P)$ (middle), and the candidate subgraph $\tilde{G}_{\mathsf{cand}}$ (right). Red squares and blue disks denote vertices in $R \cup \hat{B}$ and vertices in $B \cup \hat{R}$, respectively. In the upper layer of $\tilde{G}_{\mathsf{cand}}$, the segment endpoints represent skeleton vertices, while intermediate points on the segments represent interior vertices. The green vertices in $G(P)$ constitute the balanced separator of $G(P)$, and those in $\tilde{G}_{\mathsf{cand}}$ form the prism separator $\tilde{S}$.

the optimal matching for the subgraph induced by the disjoint sets. Subsequently, in the conquer phase, we reintroduce the removed vertices and gradually update the matching to obtain a minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$ using augmenting paths.

We begin by recalling the planar separator theorem, which serves as the foundation for our graph partitioning strategy. Lipton and Tarjan [16] established the following result for planar graphs.

▶ **Theorem 14** ([16]). *Let H be a planar graph with n_H vertices. The vertices of H can be partitioned in $O(n_H)$ time into three disjoint sets X, Y, S such that (i) no edge connects a vertex in X with a vertex in Y, (ii) $|X|, |Y| \leq 2n_H/3$, and (iii) $|S| \leq 2\sqrt{2}\sqrt{n_H}$.*

We call S a *balanced separator* of H. While $\tilde{G}_{\mathsf{cand}}$ is not necessarily planar, its inherent planar structure is derived from the eligible edge decomposition P. Let $V(P)$ be the set of endpoints of the edges (segments) in P, and let $G(P) = (V(P), P)$ be the planar skeleton defined by P, that is, it is the planar graph induced by the edges of P.

We apply the planar separator theorem to $G(P)$ to induce a separation of $\tilde{G}_{\mathsf{cand}}$. Let X, Y, S be a partition of $V(P)$ obtained by applying Theorem 14 to $G(P)$, where S is a balanced separator of $G(P)$. Let $\hat{X}, \hat{Y}$ and $\hat{S}$ denote the mirrored sets of X, Y and S, respectively, in the lower layer. We make the following observation regarding the sizes of these sets.

▶ **Observation 15.** *Let $\hat{V}(P)$ be the mirrored set of $V(P)$ in the lower layer. Then, $|X \cup \hat{X}|, |Y \cup \hat{Y}| \leq 2/3 \cdot (|V(P)| + |\hat{V}(P)|)$, and $|S \cup \hat{S}| \leq 2\sqrt{2}(|V(P)| + |\hat{V}(P)|)^{1/2}$.*

Using the partition (X, Y, S) of $V(P)$, we partition the vertex set of $\tilde{G}_{\mathsf{cand}}$ into $\tilde{X}, \tilde{Y}$ and $\tilde{S}$ with $X \subseteq \tilde{X}, Y \subseteq \tilde{Y}$ and $S \subseteq \tilde{S}$ as follows. For this, it suffices to partition the vertices of $\tilde{G}_{\mathsf{cand}}$ contained in the open segments of P. We call such vertices *interior vertices*. For each interior vertex v of $\tilde{G}_{\mathsf{cand}}$, we assign it to the set based on the endpoints of the segment of P containing v. Specifically, if the segment of P containing v has an endpoint in X (in Y), we put v and $\hat{v}$ to $\tilde{X}$ (to $\tilde{Y}$). Otherwise, both endpoints of the segment are contained in S. In this case, we put v and $\hat{v}$ to $\tilde{S}$. Here, notice that no segment of P has endpoints both in X and Y since S separates X and Y.

▶ **Lemma 16.** *No edge in $\tilde{E}_{\mathsf{cand}}$ connects a vertex in $\tilde{X}$ to a vertex in $\tilde{Y}$.*

Proof. We prove this by examining the three types of edges e in $\tilde{E}_{\mathsf{cand}}$. In the first case that e belongs to the upper layer, every eligible edge in the upper layer is contained in a segment s in P. If an endpoint of s is contained in X (or in Y), both endpoints of e are in $\tilde{X}$ (or $\tilde{Y}$), or one belongs to $\tilde{X}$ (or $\tilde{Y}$) and the other to $\tilde{S}$. If both endpoints of s are contained in S, both endpoints of e are contained in $\tilde{S}$. Thus in any case, the lemma holds.

In the other cases, the lemma holds due to the symmetric construction of $\tilde{E}_{\mathsf{cand}}$ following Lemma 12. Specifically, a vertex $v \in B \cup R$ and its copy $\hat{v}$ belong to the same set among $\tilde{X}, \tilde{Y}$ and $\tilde{S}$. Therefore, in the case that e belongs to the lower layer, or e is a link edge, the lemma holds immediately. ◀

We refer to $\tilde{S}$ as the *prism separator* of $\tilde{G}_{\mathsf{cand}}$. We classify each vertex of the prism separator $\tilde{S}$ into two types. We call a vertex of $S \cup \hat{S}$ a *skeleton vertex* of $\tilde{S}$ and a vertex of $\tilde{S} \setminus (S \cup \hat{S})$ an *interior vertex* of $\tilde{S}$. See Figure 2.

Divide-and-Conquer Algorithm. We now present a detailed divide-and-conquer algorithm for computing a minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$. The algorithm first computes the partition $(\tilde{X}, \tilde{Y}, \tilde{S})$ of the vertex set in linear time and recursively finds minimum-cost perfect matchings for the subgraphs induced by $\tilde{X}$ and $\tilde{Y}$. Note that the subgraph induced by $\tilde{X}$ (and $\tilde{Y}$) has a perfect matching due to its prism structure. Since no edges connect $\tilde{X}$ and $\tilde{Y}$, the union of their optimal matchings is a minimum-cost perfect matching for the subgraph induced by $\tilde{X} \cup \tilde{Y}$. To obtain the optimal matching for the entire graph, one might consider extending this matching by adding all vertices of the separator $\tilde{S}$ one by one using the following lemma. Here, a *minimum-cost maximum-cardinality matching* is defined as the matching with the minimum total cost among all matchings of maximum possible cardinality.

▶ **Lemma 17** ([17]). *Let H be a graph with n_H vertices and m_H edges associated with edge costs, and let v be a vertex of H. Given a minimum-cost maximum-cardinality matching of $H - \{v\}$, a minimum-cost maximum-cardinality matching of H can be computed in $O(m_H \log n_H)$ time.*

However, a naive application of Lemma 17 to all vertices in $\tilde{S}$ would take $O(|\tilde{S}| \cdot n \log n)$ time, which is inefficient since $|\tilde{S}|$ can be as large as $\Theta(n)$. To address this, we observe that $\tilde{S}$ consists of $O(\sqrt{n})$ *skeleton vertices* and a potentially large number of *interior vertices*. Our strategy is to handle the skeleton vertices using Lemma 17, while the interior vertices are handled efficiently by exploiting their 1D nature and pre-computed optimal matchings.

The algorithm processes the subgraph $\tilde{G}'$ of $\tilde{G}_{\mathsf{cand}}$ in each recursive step as follows. Using the subgraph of $G(P)$ induced by $V(P) \cap V(\tilde{G}')$, we compute a balanced separator S and a partition $(\tilde{X}, \tilde{Y}, \tilde{S})$ of $\tilde{G}'$. As mentioned earlier, we compute a minimum-cost perfect matching of $\tilde{X} \cup \tilde{Y}$ in linear time. Now consider the internal vertices of $\tilde{S}$. We decompose them into subsets such that each subset consists of the internal vertices lying on the same segment of P. For each subset W, we retrieve its optimal matching of $W \cup \hat{W}$ using the pre-computed 1D matching from Section 4.1, where $\hat{W}$ is the copy of W in the lower layer. An optimal matching of $W \cup \hat{W}$ can be obtained from a minimum-cost matching of W with penalties, which can be found in near-linear time. Since no edge of $\tilde{G}'$ connects two vertices from different subsets, the union M' of the all these matchings is a minimum-cost maximum-cardinality matching of $\tilde{X} \cup \tilde{Y} \cup \{\text{internal vertices of } \tilde{S}\}$. Finally, we reintroduce the $O(\sqrt{n'})$ skeleton vertices of $\tilde{S}$ into the matching using Lemma 17, where $n' = |V(\tilde{G}') \cap V(P)|$. We iteratively apply Lemma 17 for each skeleton vertex of $\tilde{S}$ to update $\tilde{M}'$. In this way, each recursion step takes $\tilde{O}(|V(\tilde{G}')|\sqrt{n'})$ time in total.

In the base case that the number of (skeleton) vertices of $V(P) \cap V(\tilde{G}')$ is $O(1)$, we compute a minimum-cost maximum-cardinality matching of the (non-skeleton) vertices of $V(\tilde{G}') \setminus V(P)$ using their 1D nature in $\tilde{O}(|V(\tilde{G}')|)$ time. Then we reintroduce the $O(1)$ skeleton vertices with Lemma 17. This takes $\tilde{O}(|V(\tilde{G}')|)$ time in total.

Complexity Analysis. To bound the overall time complexity, let $\eta = O(n)$ be the number of vertices of $V(P)$. At depth d of the recursion, let $H_{d,1}, H_{d,2}, \dots$ denote the disjoint subgraphs, where each $H_{d,i}$ contains $m_{d,i}$ edges and $\eta_{d,i}$ skeleton vertices along with their copies. By Theorem 14 and Observation 15, the number of skeleton vertices shrinks by a factor of at least 2/3 per level, yielding $\eta_{d,i} \leq (2/3)^d \eta$. Furthermore, since the subgraphs at any depth are edge-disjoint, $\sum_i m_{d,i} \leq |\tilde{E}_{\mathsf{cand}}| = O(n)$.

In the conquer phase, merging the sub-solutions for $H_{d,i}$ requires $O(\sqrt{\eta_{d,i}})$ augmentations. Since each augmentation takes $O(m_{d,i} \log n)$ time by Lemma 17, the total time for depth d is:

$$\sum_i O\left(\sqrt{\eta_{d,i}} \cdot m_{d,i} \log n\right) \leq O\left(\sqrt{(2/3)^d \eta} \log n\right) \sum_i m_{d,i} \leq O\left(\left(\sqrt{2/3}\right)^d \cdot n^{1.5} \log n\right).$$

Because $\sqrt{2/3} < 1$, the time for each level decreases geometrically. The total running time of the divide-and-conquer algorithm is thus dominated by the root level, which is $O(n^{1.5} \log n)$.

Overall complexity and proof of Theorem 1. We compute an approximate matching and its dual weights for the scaled graph with a suitable scaling factor in $\tilde{O}(n^{1.5} \log \Delta)$ time. The candidate graph can be computed in $O(n \log n)$ time, and the divide-and-conquer algorithm takes $O(n^{1.5} \log n)$ time. Therefore, the total time complexity of our algorithm remains $\tilde{O}(n^{1.5} \log \Delta)$.

5 Conclusion

In this paper, we presented an exact $\tilde{O}(n^{1.5} \log \Delta)$-time algorithm for the many-to-many matching problem on planar point sets with integer coordinates. Our approach relies on a reduction to perfect matching on a prism graph, a scaling framework, and a planar separator-based divide-and-conquer strategy.

Extending this framework to non-integer coordinates or higher dimensions ($d \geq 3$) poses fundamental challenges. In particular, our framework relies on Lemma 11. In continuous domains, the additive error in the inequality in Lemma 11 is no longer bounded, and thus it seems hard to apply our framework to continuous domains. For higher dimensions, geometric graphs lack planarity, preventing the use of a balanced vertex separator of $O(\sqrt{n})$ size. Moreover, in higher dimensions, maintaining the dynamic weighted nearest-neighbor data structures required in the scaling step becomes substantially more difficult. We leave the problem of designing a subquadratic-time algorithm for many-to-many matching on higher dimensions or continuous domains as an open problem.

References

1 Helmut Alt, Kurt Mehlhorn, Hubert Wagener, and Emo Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete & Computational Geometry*, 3(3):237–256, 1988. doi:10.1007/BF02187910.

2 Shinwoo An, Eunjin Oh, and Jie Xue. Approximation algorithms for the geometric multimatching problem. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025)*, pages 2317–2328, 2025. doi:10.1145/3717823.3718147.

3 Sayan Bandyapadhyay, Anil Maheshwari, and Michiel Smid. Exact and approximation algorithms for many-to-many point matching in the plane. In *Proceedings on the 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *LIPIcs*, pages 44:1–44:14, 2021. doi:10.4230/LIPIcs.ISAAC.2021.44.

4 Sayan Bandyapadhyay and Jie Xue. An $O(n \log n)$-time approximation scheme for geometric many-to-many matching. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, pages 12:1–12:15, 2024. doi:10.4230/LIPIcs.SOCG.2024.12.

5 Justin Colannino, Mirela Damian, Ferran Hurtado, Stefan Langerman, Henk Meijer, Suneeta Ramaswami, Diane L. Souvaine, and Godfried Toussaint. Efficient many-to-many point matching in one dimension. *Graphs and Combinatorics*, 23:169–178, 2007. doi:10.1007/S00373-007-0714-3.

6 Justin Colannino and Godfried Toussaint. A faster algorithm for computing the link distance between two point sets on the real line. Technical report, Technical Report SOCS-TR-2005.5, McGill University, School of Computer Science, 2005.

7 Thomas Eiter and Heikki Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997. doi:10.1007/S002360050075.

8 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

9 Harold N. Gabow. A data structure for nearest common ancestors with linking. *ACM Trans. Algorithms*, 13(4):45:1–45:28, 2017. doi:10.1145/3108240.

10 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989. doi:10.1137/0218069.

11 Akshaykumar Gattani, Sharath Raghvendra, and Pouyan Shirzadian. A robust exact algorithm for the Euclidean bipartite matching problem. *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 36:51706–51718, 2023.

12 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

13 Judith Keijsper and Rudi Pendavingh. An efficient algorithm for minimum-weight bibranching. *Journal of Combinatorics Theory, Series B*, 73(2):130–145, 1998. doi:10.1006/JCTB.1998.1817.

14 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. doi:10.1137/0212002.

15 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

16 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

17 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980. doi:10.1137/0209046.

18 Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM 2004)*, pages 208–217, 2004. doi:10.1145/1031171.1031213.

19 David Mumford. Mathematical theories of shape: Do they model perception? In *Geometric methods in computer vision*, volume 1570, pages 2–10, 1991.

20 R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proceedings of the 29th Annual Symposium on Computational Geometry (SoCG 2013)*, pages 9–16, 2013. doi:10.1145/2462356.2480283.

21 R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 306–317, 2012. doi:10.1137/1.9781611973099.29.

22 Ranjith Unnikrishnan and Martial Hebert. Measures of similarity. In *Proceedings of the 7th IEEE Workshops on Applications of Computer Vision (WACV/MOTION 2005)*, volume 1, pages 394–394. IEEE, 2005.

23 Remco C. Veltkamp and Michiel Hagedoorn. State of the art in shape matching. *Principles of visual information retrieval*, pages 87–119, 2001. doi:10.1007/978-1-4471-3702-3_4.

Faster Linear-Space Data Structures for Path Frequency Queries

Ovidiu Rața
University of Copenhagen, Denmark

Abstract

We present linear-space data structures for several frequency queries on trees, namely: path mode, path least frequent element, and path α-minority queries. We present the first linear-space data structures, requiring $O(n\sqrt{nw})$ preprocessing time, that can answer path mode and path least frequent element queries in $O(\sqrt{n/w})$ time. This improves upon the best previously known bound of $O(\log\log n\sqrt{n/w})$ achieved by Durocher et al. [10] in 2016.

For the path α-minority problem, where α is specified at query time, we reduce the query time of the linear-space data structure of Durocher et al. [10] from $O(\alpha^{-1}\log\log n)$ down to $O(\alpha^{-1})$ by employing a simple randomized algorithm with a success probability $\geq 1/2$.

We also present the first linear-space data structure supporting "Path Maximum g-value Color" queries in $O(\sqrt{n/w})$ time, requiring $O(n\sqrt{nw})$ preprocessing time. This general framework encapsulates both path mode and path least frequent element queries. For our data structures, we consider the word-RAM model with $w \in \Omega(\log n)$, where w is the word size in bits.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Data structure, Range query, Mode, Minority, Least frequent element, Trees, Linear-space, Path query

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.37

Related Version *Full Version*: `https://arxiv.org/abs/2604.18667` [14]

Funding Supported by VILLUM Foundation grant 54451, Basic Algorithms Research Copenhagen (BARC).

Acknowledgements I wish to thank Professor Mikkel Thorup for his valuable support as my Master's Thesis supervisor. I also wish to thank the anonymous reviewers of SWAT 2026 for their meaningful feedback, which helped improve the paper.

1 Introduction

According to Chan et al. [6], given a multiset S, the frequency of an element x in S, denoted by $\text{freq}_S(x)$, is the number of occurrences of the element x in S. A mode of S is an element $a \in S$ such that for all $x \in S$, $\text{freq}_S(x) \leq \text{freq}_S(a)$. Given an array A of n integers, a range mode of $A[i:j]$ is a mode of the multiset determined by the range $A[i:j]$. Alternatively, one least frequent element in a multiset A is an element of minimum multiplicity in A.

Generalizing the range mode query problem to trees, we study the problem of building a data structure that, given two nodes (i, j), can efficiently find the mode of the multiset of nodes of the path $P(i, j)$. An alternative to the path mode query is the path least frequent element query, which asks to report **any single** element on the path $P(i, j)$ of minimal multiplicity.

Historically, there has always been a gap between the query times for the array and tree versions of the mode and least frequent element problems. Chan et al. [6] developed the fastest known linear-space data structure with $O(\sqrt{n/w})$ query time for the array mode query problem, while the time for tree path queries remained $O(\log n\sqrt{n})$. Later, in 2016, Durocher et al. [10] reduced the time on trees down to $O(\log\log n\sqrt{n/w})$, for both mode

© Ovidiu Rața;
licensed under Creative Commons License CC-BY 4.0

and least frequent element queries. In this paper, we bridge this gap by reducing the time for both path mode queries and least frequent element queries on trees down to $O(\sqrt{n/w})$, in the context of linear-space data structures.

We also consider a new, more general problem, which we call the **Range Maximum *g*-value Color** query. By abstracting specific frequency conditions into a generic function g, we provide a unified algorithmic framework that simultaneously solves both the mode and least frequent element problems, while also capturing more complex frequency-based metrics. An array A of n positive integers (the "array of colors") and a function g are given. We require g to satisfy several properties, which are also satisfied by the range mode function. A detailed formal definition of these properties is provided in Section 2.

The **Range Maximum *g*-value Color** problem is to preprocess A and the function g to efficiently answer queries of the following form:

Given two indices i, j of A, determine: $\arg\max_{c \in A[i \ldots j]}\{g(i, j, c)\}$

In other words, the query asks to report any single color c corresponding to the maximum value of $g(i, j, c)$.

We study the generalization of this problem to trees, which we call **Path Maximum *g*-value Color**, and prove that there exists a linear-space data structure supporting these queries in $O(\sqrt{n/w})$ time, requiring $O(n\sqrt{nw})$ preprocessing time. Because this problem acts as a generalization, it directly reduces the $O(\log\log n\sqrt{n/w})$ worst-case query time required by the previous linear-space data structures for path mode and path least frequent element queries [10], down to $O(\sqrt{n/w})$.

We conclude our work by presenting an improvement over the path α-minority problem, studied by Durocher et al. [10]. The path α-minority query for a tree path (u, v) asks to report **any single** color that occurs at most an α fraction of the time on the path between nodes u and v. We prove that by applying a simple randomized algorithm with success probability $\geq 1/2$, their linear-space data structure can support α-minority queries in $O(\alpha^{-1})$ time (where α is specified at query time), improving on the previous $O(\alpha^{-1}\log\log n)$ query time.

We assume the Word RAM model of computation using words of $w = \Omega(\log n)$ bits, where $n = |A|$.

1.1 Related Work and Contribution

Krizanc et al. [13] presented $O(n)$-space data structures that support range mode queries in $O(\sqrt{n}\log\log n)$ time on arrays and $O(\sqrt{n}\log n)$ time on trees. Chan et al. [6] achieved $o(\sqrt{n})$ query time with an $O(n)$-space data structure that supports queries in $O(\sqrt{n/w}) \subseteq O(\sqrt{n/\log n})$ time on arrays. Durocher et al. [10] present an $O(n)$ space data structure supporting path mode queries on trees in $O(\log\log n\sqrt{n/w})$ time. Durocher et al. [10] ask whether the $\log\log n$ gap between the path mode query time on trees and the range mode query time on arrays can be closed. In this paper, we present the first $O(n)$ space data structure that supports path mode queries in $O(\sqrt{n/w}) \subset o(\log\log n\sqrt{n/w})$ query time, thus eliminating the gap between path mode and range mode query times.

In the context of the least frequent element problem, Chan et al. [7] presented an $O(n)$-space data structure that supports range least frequent element queries on arrays in $O(\sqrt{n})$ time. Durocher et al. [10] improve the range least frequent element query time down to $O(\sqrt{n/w})$, and designed the first path least frequent element query data structure for trees, with an $O(\log\log n\sqrt{n/w})$ query time. In our paper, we present the first linear-space data structure that supports path least frequent element queries on trees in $O(\sqrt{n/w})$ time, thus

also closing the $\log \log n$ gap between the array and tree query times for this problem. A history of previous best query times, achieved by linear-space data structures for mode queries and least frequent element queries, is presented in Tables 1, 2.

Table 1 History of Mode Queries (linear-space).

Authors	Array Query Time	Tree Query Time	Source
Krizanc et al.	$O(\sqrt{n} \log \log n)$	$O(\sqrt{n} \log n)$	[13]
Chan et al.	$O(\sqrt{n/w})$	–	[6]
Durocher et al.	–	$O(\log \log n \sqrt{n/w})$	[10]
This Paper	–	$\mathbf{O(\sqrt{n/w})}$	–

Table 2 History of Least Frequent Element Queries (linear-space).

Authors	Array Query Time	Tree Query Time	Source
Chan et al.	$O(\sqrt{n})$	–	[7]
Durocher et al.	$O(\sqrt{n/w})$	$O(\log \log n \sqrt{n/w})$	[10]
This Paper	–	$\mathbf{O(\sqrt{n/w})}$	–

In their work, Durocher et al. [10] present a linear-space data structure that can answer path α-minority queries on trees in $O(\alpha^{-1} \log \log n)$ time, with α specified at query time. The path α-minority query for a tree path (u, v) asks to report any color that occurs at most an α fraction of the tree path between nodes u and v. We prove that by applying a simple randomized algorithm with success probability $\geq 1/2$, their linear-space data structure can support α-minority queries in $O(\alpha^{-1})$ time, improving on the previous $O(\alpha^{-1} \log \log n)$ query time. A history of query times achieved by linear-space data structures for α-minority is given in Table 3.

Table 3 History of α-Minority Queries (linear-space).

Authors	Array Query Time	Tree Query Time	Source
Chan et al.	$O(1/\alpha)$	–	[7]
Durocher et al.	–	$O(\alpha^{-1} \log \log n)$	[10]
This Paper	–	$\mathbf{O(1/\alpha)}$	(Randomized)

2 Problem Presentation

We begin with a detailed formalization of the main problem studied in this paper, namely, the **Path Maximum g-value Color** problem. We formulated this problem to generalize the path mode query problem as broadly as possible. We also note that we present the first efficient linear-space data structure for the range version of the maximum g-value color problem, as this general problem has not been studied previously.

Our study of this problem and the methods used to solve it were inspired by the domain of retrieval/counting problems and array range query problems. A description of retrieval/counting problems is given by Chazelle [8], and some interesting instances of array range query problems can be found in Skala et al. [15]. For clarity, we first present the problem in the context of arrays, and subsequently generalize it to trees.

2.1 The Range Maximum g-value Color Problem

An instance of our problem consists of an array of integers A (referred to as the "array of colors") and an integer function g. Let $n = |A|$. We assume that $1 \leq A[i] \leq n$. If A does not fulfill this requirement, A can be transformed in $O(n \log n)$ time to an array A' with values compressed to the interval $\{1, \ldots, n\}$.

The starting point of our study was the range mode function. We developed our data structure using only a subset of the properties of the range and path mode functions. We require g to satisfy the following constraints, all of which are shared by the range mode function:

1. **Domain:** The function $g(i, j, c)$ takes three integer arguments, where $1 \leq i \leq j \leq n$ are indices of A, and c is a color present in the subarray $A[i \ldots j]$.
2. **Output:** g produces an integer output that fits into a single machine word.
3. **Range Contraction:** For any valid input (i, j, c),

   $$g(i, j, c) = g(i', j', c)$$

   where i' and j' are the minimum and maximum indices, respectively, such that $i \leq i' \leq j' \leq j$ and $A[i'] = A[j'] = c$. In other words, the value of g depends only on the range spanned by the occurrences of c within the query interval.
4. **Oracle Complexity:** We assume g can be computed efficiently given a contracted range. Specifically, computing $g(i, j, c)$ such that $A[i] = A[j] = c$ takes $O(1)$ time. This computation may use an additional $O(n)$-space static data structure.

The **Range Maximum g-value Color** problem is to preprocess A and the function g to efficiently answer queries of the following form:

Given two indices i, j of A, determine: $\arg\max_{c \in A[i \ldots j]}\{g(i, j, c)\}$

In other words, determine any single color c corresponding to the maximum value of $g(i, j, c)$.

2.2 Generalization to trees

This problem generalizes naturally to trees. Instead of an array, we are given a **rooted**, node-colored tree $\mathcal{T}$ and a function g defined on tree paths. An input to g consists of a triple (u, v, c), where u, v are nodes in $\mathcal{T}$ and c is a color occurring on the simple path $P(u, v)$. From now on, we will consider $\mathcal{T}$ to be rooted at node 1.

The properties of g remain analogous. Crucially, the **Range Contraction** property (Property 3) translates as follows:

$$g(u, v, c) = g(u', v', c)$$

where u' is the node on $P(u, v)$ closest to u such that $\mathcal{T}[u'] = c$, and v' is the node on $P(u, v)$ closest to v such that $\mathcal{T}[v'] = c$, where $\mathcal{T}[u]$ denotes the color of node u in the tree $\mathcal{T}$.

We refer to this generalized problem as the **Path Maximum g-value Color** problem. This is the main problem we focus on in our work. In the following sections, we present a linear-space data structure supporting path maximum g-value color queries in $O(\sqrt{n/w})$ time, requiring $O(n\sqrt{nw})$ preprocessing time.

2.3 Applications of the path maximum g-value color problem

To motivate why this abstract definition is interesting in its own right, we introduce the **Range Maximum Sum Color** problem. Given a color array A and a cost array V, we wish to find the color in a range $A[i \dots j]$ that maximizes the sum of its associated costs. Formally:

$$\arg \max_{c \in A[i \dots j]} \left\{ \sum_{k \in [i,j]: A[k]=c} V[k] \right\}$$

This is a specific instance of the maximum g-value color problem, as the summation function satisfies all properties of g (specifically, the sum of costs for color c in range $[i, j]$ is identical to the sum in the contracted range $[i', j']$).

This problem, again, naturally generalizes to trees; thus, we obtain the **Path Maximum Sum Color** problem. By using the **virtual tree** data structure, and Lemma 2 presented in Section 3, we can easily verify that:

$$g(u_c, v_c, c) = \begin{cases} g(u_c, r_c, c) + g(v_c, r_c, c) - 2 \cdot g(lca, r_c, c) & lca \notin P(u_c, v_c) \\ g(u_c, r_c, c) + g(v_c, r_c, c) - 2 \cdot g(lca, r_c, c) + V[lca] & lca \in P(u_c, v_c) \end{cases} \quad (1)$$

where lca is the lowest common colored ancestor of u_c and v_c (which requires $\mathcal{T}$ to be rooted), u_c, v_c are both c-colored nodes, and r_c is the root of the virtual tree of color c. We can verify that $g(u_c, v_c, c)$ can be computed in $O(1)$ time using relation 1 if we store for each u_c the values $g(u_c, r_c, c)$ in a table of size $O(n)$.

Both on arrays and on trees, the **Maximum Sum Color** problem acts as a generalization for both the **Mode** problem (where $V[k] = 1$ for all k) and the **Least Frequent Element** problem (where $V[k] = -1$ for all k, allowing us to maximize the negated sum).

2.3.1 Practical Example: Financial Time Series

As an additional motivation towards the independent study of the **Range Maximum *g*-value Color** problem, consider a log of chronologically ordered stock records: "Stock c changed by value v". Storing stock IDs in A and changes in V, the Maximum Sum Color problem answers: *"Which stock had the highest net increase/decrease between time i and j?"*

Furthermore, our general g-value framework allows for more complex queries, such as identifying the stock with the **maximum mean fluctuation** per record within a time period, provided the mean is calculated over the stock's records. This demonstrates that the g-value framework extends beyond standard frequency queries to elegantly handle cumulative weights and penalties.

2.4 Hierarchy and Contribution

We conclude this section by outlining the difficulty hierarchy. We focus on solving the most general variant: the Path Maximum g-value Color problem on trees. We present a linear-space data structure for this problem with a query time of $O(\sqrt{n/w})$. Remarkably, this matches the query time of the fastest known data structure for the range mode query problem (Chan et al. [6]), which corresponds to the simplest case in our hierarchy.

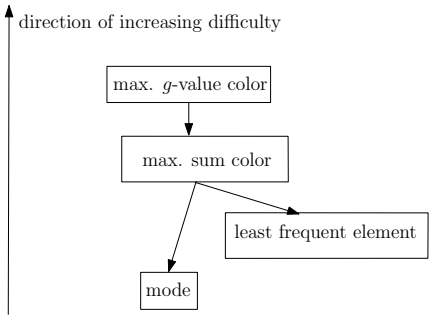


Figure 1 The hierarchy of problems ordered by increasing difficulty and generality. The **max. g-value color** problem generalizes the **max. sum color** problem, which covers both the **mode** and **least frequent element** problems.

3 Prerequisites

We mention that we consider the colored tree $\mathcal{T}$ given as input to be rooted at node 1. Prior to presenting our solution, we introduce several prerequisite results and definitions. We rely on the following standard results for tree data structures:

▶ **Lemma 1** (Bender and Farach-Colton [3]; Berkman et al. [5]; Dietz [9]). *There exists a linear-space data structure supporting level ancestor queries on trees in $O(1)$ time, requiring $O(n)$ preprocessing time.*

▶ **Lemma 2** (Bender and Farach-Colton [2]; Berkman and Vishkin [4]). *There exists a linear-space data structure supporting lowest common ancestor (LCA) queries on trees in $O(1)$ time, requiring $O(n)$ preprocessing time.*

3.1 Virtual Trees

We define **virtual trees**, which are essential for our data structure.

▶ **Definition 3.** *Given a colored tree $\mathcal{T}$ and a color c, the **virtual tree of color c**, denoted $\mathcal{V}_c$, is a rooted tree defined over the set of nodes $\{v \in \mathcal{T} \mid color(v) = c\}$.*

The structure of $\mathcal{V}_c$ is defined by the following parent relationship: for any node $v \in \mathcal{V}_c$, the parent of v in $\mathcal{V}_c$ is the nearest proper ancestor of v in $\mathcal{T}$ that also has color c.

To ensure $\mathcal{V}_c$ is a single connected tree (rather than a forest), we introduce a virtual root node r_c for each color c. If a node v of color c has no ancestors of color c in $\mathcal{T}$, we set its parent in $\mathcal{V}_c$ to be r_c.

It is possible to construct all virtual trees $\mathcal{V}_c$ for all colors c in $O(n)$ total time using a single depth-first traversal of $\mathcal{T}$. Since the node sets for each color form a partition of the colored nodes in $\mathcal{T}$, the total space required is linear, $O(n)$. We defer the algorithm for computing all the virtual trees $\mathcal{V}_c$ to the full version of this paper, available as a preprint on arXiv [14].

By applying standard LCA and Level Ancestor structures (Lemmas 1 and 2) to these virtual trees, we obtain the following corollary:

▶ **Corollary 4.** *There exists a linear-space data structure, constructed in $O(n)$ time, that answers the following queries in $O(1)$ time:*

1. *Given two nodes $u, v \in \mathcal{T}$ of the same color c, determine the lowest common ancestor of u and v **with respect to color** c (i.e., the LCA of u and v in $\mathcal{V}_c$).*
2. *Given a node u of color c and an integer k, determine the k-th ancestor of u **of color** c (i.e., the k-th ancestor of u in $\mathcal{V}_c$).*

Proof. For each distinct color c present in $\mathcal{T}$, we explicitly construct the virtual tree $\mathcal{V}_c$. Over each $\mathcal{V}_c$, we build instances of the data structures described in Lemmas 1 and 2.

Let $n_c = |V(\mathcal{V}_c)|$. The construction and preprocessing for a specific color c takes $O(n_c)$ time and space. Summing over all colors, the total preprocessing time is $\sum_c O(n_c) = O(n)$, and the total space is similarly $\sum_c O(n_c) = O(n)$. The queries are answered by querying the specific structure for $\mathcal{V}_c$ in $O(1)$ time. ◀

4 Blocking Technique

Krizanc et al. [13], Chan et al. [6, 7], and Durocher et al. [10] successfully employed blocking techniques to solve non-trivial frequency queries on arrays and trees. We adapt their framework to develop a multi-level blocking strategy, allowing us to achieve the desired query time for path maximum g-value color queries on trees.

▶ **Lemma 5** (Durocher et al. [10]). *Let $\mathcal{T}$ be a tree with n nodes, and let $t < n$ be an integer parameter (the blocking factor). There exists a subset of marked nodes $M_t \subseteq V(\mathcal{T})$ and an associated rooted tree $\mathcal{T}_t$ with vertex set M_t, satisfying the following properties:*

1. ***Size:*** *$|M_t| = O(n/t)$;*
2. ***LCA Closure:*** *M_t is closed under the lowest common ancestor operation (i.e., for any $u, v \in M_t$, the lowest common ancestor of u and v in $\mathcal{T}$ is also in M_t);*
3. ***Gap Size:*** *Any simple path in $\mathcal{T}$ consisting entirely of unmarked nodes has length at most t.*

Two nodes $u, v \in M_t$ are adjacent in $\mathcal{T}_t$ if and only if the simple path between them in $\mathcal{T}$ contains no other nodes from M_t. The set M_t and the tree $\mathcal{T}_t$ can be constructed in $O(n \log n)$ time.

▶ **Lemma 6** (Durocher et al. [10]). *Given a tree $\mathcal{T}$ with n nodes, an integer $t < n$, and a set of marked nodes M_t (obtained according to Lemma 5), there exists a partition of the vertex set $V(\mathcal{T})$, denoted by $\mathcal{B}_t = \{B_1, \ldots, B_k\}$, satisfying the following properties:*

1. ***Block Count:*** *The number of blocks is $k = O(n/t)$;*
2. ***Block Size:*** *For each block $B \in \mathcal{B}_t$, the number of nodes is $|B| = O(t)$;*
3. ***Connectivity:*** *For each block $B \in \mathcal{B}_t$, the subgraph of $\mathcal{T}$ induced by the nodes in B is connected.*

The partition $\mathcal{B}_t$ can be computed in $O(n \log n)$ time.

We defer the full details of the proofs of Lemmas 5 and 6 to the full version of this paper [14]. Further, for our approach, we require a multilevel partition of the nodes of $\mathcal{T}$. From Lemma 6, we derive the following corollary, which formalizes the construction of a hierarchical partition.

▶ **Corollary 7.** *Given a colored tree $\mathcal{T}$, two blocking factors $t_1 < t_2$, and the set of marked nodes M_{t_1} with its associated tree $\mathcal{T}_{t_1}$ (obtained according to Lemma 5), there exists a subset $M_{t_2} \subseteq M_{t_1}$ and a partition $\mathcal{B}_{t_2/t_1}$ of M_{t_1}, and a tree $\mathcal{T}_{t_2}$.*

These can be computed in $O(|M_{t_1}| \log |M_{t_1}|)$ time and satisfy the following properties:

1. ***Consistency:*** *The set M_{t_2} is a valid set of marked nodes for $\mathcal{T}$ with blocking factor t_2 (satisfying all properties of Lemma 5).*
2. ***Relative Gap:*** *Any simple path in the compressed tree $\mathcal{T}_{t_1}$ consisting entirely of nodes from $M_{t_1} \setminus M_{t_2}$ has length $O(t_2/t_1)$.*
3. ***Partition Structure:*** *The partition $\mathcal{B}_{t_2/t_1}$ divides the nodes of M_{t_1} into $O(|M_{t_1}| \cdot \frac{t_1}{t_2})$ blocks, each of size $O(t_2/t_1)$, such that the subgraph of $\mathcal{T}_{t_1}$ induced by each block is connected.*

Proof. We apply the construction procedures from Lemma 5 and Lemma 6 directly to the compressed tree $\mathcal{T}_{t_1}$, treating it as the input tree with a blocking factor of $t' = t_2/t_1$. Since $|V(\mathcal{T}_{t_1})| = |M_{t_1}|$, the construction requires $O(|M_{t_1}| \log |M_{t_1}|)$ time. The resulting marked nodes M_{t_2} and partition $\mathcal{B}_{t_2/t_1}$ satisfy the size and connectivity properties with respect to $\mathcal{T}_{t_1}$ by definition. Furthermore, since M_{t_1} preserves the LCA closure property of $\mathcal{T}$, and M_{t_2} is constructed to preserve LCA closure on $\mathcal{T}_{t_1}$, M_{t_2} maintains the necessary topological properties for $\mathcal{T}$. ◀

Crucially, the partition $\mathcal{B}_{t_2/t_1}$ of M_{t_1} implicitly defines a coarser partition $\mathcal{B}_{t_2}$ of the original vertex set $V(\mathcal{T})$. A high-level block $B \in \mathcal{B}_{t_2}$ consists of a set of connected low-level blocks from $\mathcal{B}_{t_1}$. To enumerate the nodes of a high-level block $B \in \mathcal{B}_{t_2}$, we identify the corresponding block of marked nodes $B' \in \mathcal{B}_{t_2/t_1}$ (where $B' \subseteq M_{t_1}$) and iterate through the low-level block $B''_u \in \mathcal{B}_{t_1}$ associated with each $u \in B'$.

▶ **Corollary 8.** *Given a colored tree $\mathcal{T}$, two blocking factors $t_1 < t_2$, and the respective partitions into blocks computed according to Lemma 6 and Corollary 7, we can compute in $O(|M_{t_1}|)$ time, for each block $B_2 \in \mathcal{B}_{t_2/t_1}$, a tree $\mathcal{T}_{t_2/t_1}[B_2]$, which is the subtree of T_{t_1} induced by the nodes in B_2 (which are also present in M_{t_1}).*

Proof. By Property 3 of Corollary 7, the subgraph of $\mathcal{T}_{t_1}$ induced by any block $B \in \mathcal{B}_{t_2/t_1}$ is connected. Since $\mathcal{T}_{t_1}$ is a tree, any connected subgraph is also a tree. To construct these explicit tree representations, we iterate through the edges of $\mathcal{T}_{t_1}$. For each edge $(u, v) \in E(\mathcal{T}_{t_1})$, if both u and v belong to the same block $B \in \mathcal{B}_{t_2/t_1}$, we add the edge to $\mathcal{T}_{t_2/t_1}[B]$. This traversal visits every edge exactly once, resulting in a total runtime of $O(|V(\mathcal{T}_{t_1})|) = O(|M_{t_1}|)$. ◀

We refer to $\mathcal{T}_{t_2/t_1}[B_2]$ as the **block tree** of B_2. We depict an example of a 2-level block partition in Figure 2.

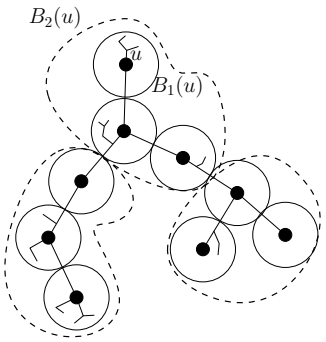


Figure 2 A 2-level partition of a tree. The representatives of a block are depicted with black dots. t_1-blocks are depicted by circles, and t_2-blocks by dashed splinoids. An example representative node u is chosen, and its respective blocks $B_1(u)$ and $B_2(u)$ are depicted.

We define the notions of home blocks for nodes and node representatives for blocks.

▶ **Definition 9** (Home Blocks and Representatives)**.** *For any node $u \in V(\mathcal{T})$ and level k, let $B_k(u)$ denote the unique block in the partition $\mathcal{B}_{t_k}$ that contains u. We assume each block $B \in \mathcal{B}_{t_k}$ is associated with a unique representative node in M_{t_k}.*

This definition allows us to examine a nested structure around a node u. The node u resides in a local block $B_1(u)$, which is nested within $B_2(u)$, which is further nested within $B_3(u)$ and so on.

4.1 Parameter Selection

For our specific data structure, we employ a 3-level partition of $\mathcal{T}$. We assume the first blocking factor t_1 is provided as a parameter. We then define the higher-level factors as follows:

$$t_2 = t_1 \log\log n, \quad t_3 = t_2\sqrt{\log n}$$

During the preprocessing phase, we compute the associated structures M_{t_i}, $\mathcal{T}_{t_i}$, and $\mathcal{B}_{t_i}$ for $1 \le i \le 3$. Specifically, we construct the full block partitions $\mathcal{B}_{t_i}$ using the hierarchical method described in Corollary 7. Based on the established results, this entire preprocessing step requires $O(n \log n)$ time.

5 Overview and Strategy

Having established the prerequisites, we now provide a high-level overview of our query answering strategy. Our approach refines the blocking technique of Durocher et al. [10]. In their solution, the query algorithm identifies a set of $O(\sqrt{n/w})$ candidate nodes. For every distinct color appearing in this set of nodes, they perform a frequency verification step taking $O(\log\log n)$ time.

To improve upon this, we introduce a hierarchical candidate selection process. Instead of a single set, we identify two distinct sets of **candidate nodes**, denoted S_1 and S_2. These sets act as proxies for the colors we need to verify:

- **Primary Candidate Nodes (S_1):** A small set of nodes ($|S_1| = O(t_1)$). For every color c appearing in S_1, we perform a standard verification taking $O(\log\log n)$ time.
- **Secondary Candidate Nodes (S_2):** A larger set of nodes ($|S_2| = O(t_2)$). For every color c appearing in S_2, we store a triple (c, l, r), where l and r are the first occurrences of c, when traversing the path in the direction $i \to j$ and $j \to i$, respectively. We need only $O(1)$ time to compute $g(c, l, r)$ for one such color.

This split allows us to process a larger total number of candidate nodes (and thus potential colors) without degrading the overall query complexity.

5.1 Decomposition into Disjoint Subtasks

To efficiently identify these candidate nodes, we decompose the query based on the hierarchy of blocks defined in Section 4. We decompose the problem into disjoint subtasks by partitioning the set of distinct colors present on the query path. Each subtask identifies a unique subset of colors based on their presence or absence in the hierarchical blocks surrounding i and j.

Let $C_{path} = C(P(i, j))$ denote the set of all distinct colors present on the query path. For any block B, let $C(B)$ denote the set of distinct colors present in that block.

We define the complement of a block's color set with respect to the **entire tree**. Let $\mathcal{U}$ be the universe of all distinct colors in the tree $\mathcal{T}$. We define:

$$\overline{C(B)} = \mathcal{U} \setminus C(B)$$

This represents the set of all colors in the tree that are *not* present in block B.

The set of path colors C_{path} is partitioned into the following disjoint subsets $C_1, \ldots, C_{10}$ using intersection operations on these color sets:

- **1. The Global Colors:** $C_1 = C_{path} \cap \overline{C(B_3(i))} \cap \overline{C(B_3(j))}$.
  Colors on the path that are not present in the Level 3 blocks of either endpoint.

- **2. The Level 3 Ascent:** $C_2 = C_{path} \cap C(B_3(i)) \cap \overline{C(B_2(i))} \cap \overline{C(B_3(j))}$.
  Colors on the path that are in i's Level 3 block, but not in i's Level 2 block or j's Level 3 block.
- **3. The Level 2 Ascent:** $C_3 = C_{path} \cap C(B_2(i)) \cap \overline{C(B_1(i))} \cap \overline{C(B_3(j))}$.
  Colors on the path that are in i's Level 2 block, but not in i's Level 1 block or j's Level 3 block.
- **4. The Level 3 Descent:** $C_4 = C_{path} \cap C(B_3(j)) \cap \overline{C(B_2(j))} \cap \overline{C(B_3(i))}$.
  Symmetric to C_2.
- **5. Level 3 Intersection:** $C_5 = C_{path} \cap C(B_3(i)) \cap C(B_3(j)) \cap \overline{C(B_2(i))} \cap \overline{C(B_2(j))}$.
  Colors on the path present in both Level 3 blocks, but in neither Level 2 block.
- **6. Mixed Intersection:** $C_6 = C_{path} \cap C(B_2(i)) \cap C(B_3(j)) \cap \overline{C(B_1(i))} \cap \overline{C(B_2(j))}$.
  Colors on the path present in i's Level 2 block and j's Level 3 block, excluding inner blocks.
- **7. The Level 2 Descent:** $C_7 = C_{path} \cap C(B_2(j)) \cap \overline{C(B_1(j))} \cap \overline{C(B_3(i))}$.
  Symmetric to C_3.
- **8. Mixed Intersection:** $C_8 = C_{path} \cap C(B_3(i)) \cap C(B_2(j)) \cap \overline{C(B_2(i))} \cap \overline{C(B_1(j))}$.
  Symmetric to C_6.
- **9. Level 2 Intersection:** $C_9 = C_{path} \cap C(B_2(i)) \cap C(B_2(j)) \cap \overline{C(B_1(i))} \cap \overline{C(B_1(j))}$.
  Colors on the path present in both Level 2 blocks, excluding both Level 1 blocks.
- **10. Local Colors:** $C_{10} = C_{path} \cap \Big(C(B_1(i)) \cup C(B_1(j))\Big)$.
  Colors on the path present in the smallest (Level 1) blocks of either endpoint.

In the subsequent analysis, we will omit the subtasks marked as symmetric to others (e.g., C_4, C_7, C_8), as their solutions are algorithmically identical to their counterparts, differing only in orientation.

For each of the subtasks 4-9, we will identify sets of candidate nodes of size $O(t_2)$, together with the corresponding endpoints l, r, and add them to the set S_2. For subtasks $1-3$ and 10, we will identify sets of candidates of size $O(t_1)$ or $O(1)$, and add them to S_1. At the end, we process the sets S_2 and S_1, in time $O(|S_2|) = O(t_2)$ and $O(|S_1| \cdot \log\log n) = O(t_1 \cdot \log\log n)$, respectively, and return the color with the largest g-value. Since $t_1 \log\log n = t_2$, processing S_1 and S_2 takes $O(t_2)$ in total.

5.1.1 Proof of Completeness

We prove that these subtasks partition C_{path}. Consider any color $c \in C_{path}$. Based on the block hierarchy, we can determine the membership of c in the color sets of the blocks surrounding i and j. Let P_u be the proposition $c \in C(B_3(u))$ and Q_u be the proposition $c \in C(B_2(u))$ for $u \in \{i, j\}$. Since $B_2(u) \subseteq B_3(u)$, we have $Q_u \implies P_u$. The definitions of C_1 through C_9 correspond precisely to the disjoint logical conjunctions of these propositions (e.g., C_2 corresponds to $P_i \wedge \neg Q_i \wedge \neg P_j$). The set C_{10} captures the remaining cases where the color is present in $C(B_1(i))$ or $C(B_1(j))$. Thus, every color $c \in C_{path}$ satisfies exactly one of these conditions, ensuring that $\bigcup_{k=1}^{10} C_k = C_{path}$ and all C_k are disjoint.

6 Solving the Subtasks

In this section, we present the algorithmic solutions for the subtasks defined previously. While the query procedures are concise, the precomputation strategies vary in complexity across subtasks; we sketch the data structures here, deferring full construction details to the full version of this paper [14]. We first state a key result from Durocher et al. [10] that enables efficient verification for our primary candidate set S_1.

▶ **Lemma 10** (Durocher et al. [10]). *There exists an $O(n)$-space data structure that supports lowest colored ancestor queries on trees in $O(\log \log n)$ time.*

To verify any $c \in S_1$, we must evaluate $g(i, j, c)$, which requires identifying the first and last occurrences of c on $P(i, j)$. Using the lowest colored ancestor structure from Lemma 10, we locate these endpoints in $O(\log \log n)$ time, and compute g in $O(1)$.

We analyze the space complexity, query time, and preprocessing time for each subtask. Note that any candidate color identified in these subtasks is added to the candidate set S_1 (verification taking $O(\log \log n)$ time) or S_2 (verification taking $O(1)$ time), as per the strategy in Section 5.

6.1 Solving Subtask 1: The Global Region

We precompute a table T of size $O\left((n/t_3)^2\right)$, indexed by the identifiers of the blocks in $\mathcal{B}_{t_3}$. For every pair of blocks $B_u, B_v \in \mathcal{B}_{t_3}$ with representatives u, v, the entry $T[u][v]$ stores the color c that maximizes $g(u, v, c)$, restricted to colors appearing on the path $P(u, v)$ but not in $C(B_u) \cup C(B_v)$. During a query, we identify the blocks $u = B_3(i)$ and $v = B_3(j)$, retrieve the candidate $c = T[u][v]$ in $O(1)$ time, and add it to the primary candidate set S_1. The table construction requires $O(n \cdot (n/t_3))$ preprocessing time and $O\left((n/t_3)^2\right)$ space, measured in words.

Complexity. Preprocessing $O(n \cdot (n/t_3))$; Space $O((n/t_3)^2)$ words; Query Time $O(1)$.

6.2 Solving Subtask 2: The Level 3 Ascent

We employ a lookup table T of size $O(\frac{n}{t_2} \cdot \frac{n}{t_3})$, indexed by pairs (u, v) representing a Level 2 block and a Level 3 block. The entry $T[u][v]$ stores the relative index (using $O(\log \log n)$ bits) of a specific Level 1 block (or t_1-node) inside $B_3(i)$ that contains at least one occurrence of the optimal color. During a query, we iterate through every distinct color c' in this retrieved t_1-node's subtree and add each to S_1.

Complexity. Preprocessing $O(n \cdot (n/t_3))$; Space $O((n/t_3)(n/t_2) \log \log n)$ bits; Query Time $O(t_1)$.

6.3 Solving Subtask 3: The Level 2 Ascent

This solution is analogous to Subtask 2. We use a table T of size $O(\frac{n}{t_1} \cdot \frac{n}{t_3})$. For a pair of blocks B_u, B_v (Level 1 and Level 3), $T[u][v]$ stores the index of a t_1-node inside $B_2(u)$ containing the optimal color. All colors in that t_1-node are added to S_1.

Complexity. Preprocessing $O(n \cdot (n/t_3))$; Space $O((n/t_3)(n/t_1) \log \log n)$ bits; Query Time $O(t_1)$.

6.4 Prerequisites for Subtasks 5, 6 and 9

To efficiently solve the intersection subtasks, we require advanced data structures that allow us to locate specific color occurrences within the block hierarchy in constant time. We refer to trees with size bounded by $s = \frac{\log n}{8 \log \log n}$ as **small trees**.

▶ **Lemma 11.** *There exists an $O(n)$-space data structure, constructible in $O(n)$ time, that answers the following query in $O(1)$ time: Given a small tree τ, a binary string σ of length $|V(\tau)|$, and a node $u \in V(\tau)$, return the lowest ancestor of u marked with 1 in σ. If no such ancestor exists, return a marked node v' such that no ancestor of v' (other than potentially u) is marked.*

Proof. We employ the tabulation method. We precompute answers for all possible canonical instances of small trees. The number of distinct rooted trees with size up to s is bounded by $O(2^{s \log s})$. For each tree topology τ, there are $2^{|V(\tau)|} \leq 2^s$ possible bitstrings σ. Thus, the total number of distinct query inputs (τ, σ, u) is bounded by:

$$O(2^{s \log s} \cdot 2^s \cdot s) = O(2^{2 \log \log n \frac{\log n}{8 \log \log n}} \cdot \log n) = O(n^{1/4} \log n) \ll O(n).$$

We construct a lookup table T indexed by the canonical ID of τ, the bitstring σ, and the node index u. We populate T using a standard DFS traversal for each instance. During the query phase, we map the input small tree to its canonical ID and retrieve the answer from T in $O(1)$ time. The total space and preprocessing time are sublinear. ◀

▶ **Lemma 12.** *Given a colored tree $\mathcal{T}$ and blocking factors $t_1 < t_2$ with $t_2/t_1 = O(\log n)$, there exists a linear-space perfect hashing structure (preprocessed in $O(n)$ expected time) that answers the following in $O(1)$ time: Given color c and t_2-block B, return a binary string σ of length t_2/t_1 indicating which t_1-blocks inside B contain c.*

Proof. We construct a dictionary of valid pairs. Let S be the set of triples (c, B, σ), where c is a color present in the t_2-block B, and σ is the bitmask of length t_2/t_1 representing the presence of c in the constituent t_1-blocks. The size of S is bounded by $O(n)$ because each node in $\mathcal{T}$ belongs to exactly one t_1-block, contributing exactly one bit to one mask in the entire structure. We store S using a static perfect hashing scheme (e.g., Fredman, Komlós and Szemerédi [11], Hagerup and Torben [12], or Belazzougui, Botelho and Dietzfelbinger [1]), allowing $O(1)$ worst-case lookups to retrieve σ for a pair (c, B), or returning null if $c \notin C(B)$. ◀

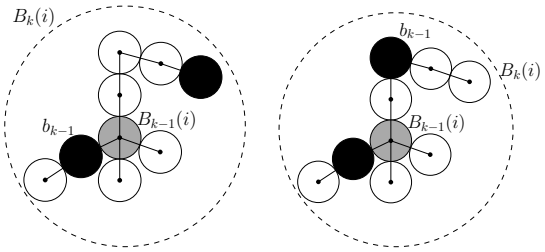
Combining these lemmas with the block tree definitions, we obtain the following result for locating path endpoints.

▶ **Corollary 13.** *There exists a linear-space data structure that, given endpoints i, j, blocks $B_k(i), B_{k'}(j)$ $(k, k' \in \{2, 3\})$, and a color c present in the blocks but not in their predecessors $B_{k-1}(i), B_{k'-1}(j)$, determines the first occurrence of c on the path $P(i, j)$ in $O(1)$ time.*

Proof. We assume the data structures from Lemmas 11 and 12 are constructed. Without loss of generality, we focus on finding the first occurrence of c in $B_k(i)$ closest to i.

First, we query the hashing structure from Lemma 12 with $(c, B_k(i))$ to retrieve the bitmask σ. Treating the hierarchy of t_{k-1}-blocks inside $B_k(i)$ as a small tree τ, we use the structure from Lemma 11 with start node $u = B_{k-1}(i)$ to find the lowest marked ancestor block b_{k-1}. This block is the t_{k-1}-block closest to $B_{k-1}(i)$ that contains c.

We distinguish two cases based on the topological relationship between $B_{k-1}(i)$ and b_{k-1}, as illustrated in Figure 3.



(a) Case 1 (b) Case 2

Figure 3 The two possible cases for the node b_{k-1}. The block marked with gray is the block $B_{k-1}(i)$. Blocks containing color c are marked with black, while the blocks that don't contain c are marked with white. The block representatives are marked as a point in the center of a block circle. In (a), Case 1 is presented, where b_{k-1} is not an ancestor of $B_{k-1}(i)$. In (b), Case 2 is presented, where b_{k-1} is an ancestor of $B_{k-1}(i)$.

Case 1: b_{k-1} is not an ancestor of $B_{k-1}(i)$. Here, the path to c must pass through the LCA of the blocks. We maintain a precomputed table storing the *topmost* occurrence of every color c in every t_{k-1}-block. Let this node be v_{top}. We compute the immediate colored ancestor of v_{top}, denoted c_{anc}. If c is a proper ancestor of $\text{LCA}(i, j)$, we locate the first occurrence by finding the occurrence on the path from j (say c') and performing a level ancestor query for the node at distance $\text{depth}(c') - \text{depth}(c_{anc}) - 1$. To solve this case, we require an additional linear-space perfect hashing data structure to determine the topmost occurrence of a color c inside a block.

Case 2: b_{k-1} is an ancestor of $B_{k-1}(i)$. Here, the path traverses intermediate blocks that do not contain c. By the LCA Closure property, the relevant occurrence lies on an edge connecting blocks. We maintain a secondary perfect hashing structure storing pairs (c, β), where β is a block that does *not* contain c, but the edge connecting β to its parent block contains an occurrence of c. If b_{k-1} is the found block, we compute the depth difference δ between $B_{k-1}(i)$ and b_{k-1}, jump to the $(\delta - 1)$-th ancestor block b'_{k-1} (the child of b_{k-1} on the path), and query the secondary hash table for (c, b'_{k-1}) to find the occurrence on the entering edge.

The perfect hashing data structure for determining the occurrence of color c on the edge from a block β to its parent block, requires additional $O(n + \sum_{c \in \mathcal{T}} count_c)$ words of space, because each occurrence of a color c, appears in at most one path connecting β to its parent block, according to the LCA Closure property. ◀

We must note that the same approach, with slight modifications, can be used to determine the desired nodes at which a color c occurs, even in the degenerate case, where the block $B_k(i)$ is an ancestor of $B_{k'}(j)$ or vice versa.

6.5 Solving Subtask 5: Level 3 Intersection

We apply a stratified blocking approach. We define a sequence of blocking factors $s_0, \ldots, s_{|s|}$ such that $s_0 = t_3$, $s_{|s|} = t_2$, and $s_{k+1} = t_2 \cdot (\log(s_k/t_2))^2$. Note that $|s| = O(\log^*(t_3/t_2))$.

For each blocking factor s_k, let $\mathcal{B}_{s_k}$ be the set of s_k blocks which partitions $V(\mathcal{T})$. For an s_k block $B \in \mathcal{B}_{s_k}$, let L_B denote the list of all nodes from $V(\mathcal{T})$, contained inside block B, ordered in increasing order of the node index. We construct tables T_k for each level s_k. For every configuration of blocks $(B_{s_k}(i), B_{s_k}(j))$, we store the approximate position (using the most significant $2\log(s_k/t_2)$ bits) of the optimal color c within the linearized list of nodes of the blocks $B_{s_k}(i), B_{s_k}(j)$. The total space required is:

$$\sum_{k=0}^{|s|}\left(\frac{n}{s_k}\right)^2\cdot 2\log\left(\frac{s_k}{t_2}\right)=\left(\frac{n}{t_2}\right)^2\sum_{k=0}^{|s|}\frac{2\log r_k}{r_k^2}=O\left(\left(\frac{n}{t_2}\right)^2\right)\text{ bits,}$$

where $r_k = s_k/t_2$. The query iterates through $O(t_2)$ candidate colors suggested by these tables. For each candidate c, we verify it is in $C(B_3)\cap\overline{C(B_2)}$ using Lemma 12 in $O(1)$ time, find its endpoints l, r using Corollary 13, and add (c, l, r) to S_2.

6.6 Solving Subtasks 6, 9, and 10

Subtask 6 (Mixed Level 2/3). We iterate through all $c \in C(B_2(i))$. We verify membership in the target set C_6 using Lemma 12 and find endpoints (l, r) using Corollary 13, all in $O(1)$ per color. Valid triplets are added to S_2. The complexity is dominated by the size of $B_2(i)$, i.e., $O(t_2)$.

Subtask 9 (Level 2 Intersection). Analogous to Subtask 6; runtime is $O(t_2)$.

Subtask 10 (Local). We iterate through every color in $C(B_1(i)) \cup C(B_1(j))$ and add them directly to S_1 (verified in $O(\log\log n)$). The cost is $O(t_1)$.

Finally, to determine the actual answer to the query, we need to iterate through the colors inside S_1, find their endpoints on the path $P(i, j)$ by using the $O(\log\log n)$ time query from Lemma 10, and compute the associated value of g. Then, we iterate through the triplets $(c, l, r) \in S_2$, and compute $g(c, l, r)$. The answer to the query for the path $P(i, j)$ is the color corresponding to the maximum value of g computed in this step.

Processing S_2 will take $O(|S_2|) = O(t_2)$ time, and processing S_1 will take $O(|S_1| \cdot \log\log n) = O(t_1 \cdot \log\log n)$ time. Since $t_1 \log\log n = t_2$, the total time of processing the sets of candidates S_2 and S_1 is $O(t_2)$.

We will present the preprocessing algorithms and analyze their respective runtime for each data structure in the full version of this paper [14]. Now we only mention that a preprocessing runtime of $O(n(n/t_2)) = O(n \cdot \frac{n}{t_1 \log\log n})$ can be achieved. The total space required by our data structure is $O((n/t_2)^2)$ bits. By substituting t_2 by t, we formalize our result as follows.

▶ **Lemma 14.** *Given a colored tree $\mathcal{T}$, of n nodes, a function g respecting the properties from Section 2, and a blocking factor t, there exists a data structure, supporting path maximum g-value color queries in $O(t)$ time, requiring $O((\frac{n}{t})^2)$ space in bits, and an $O(n(n/t))$ preprocessing time.*

By setting $t_1 = \frac{\sqrt{n/w}}{\log\log n}$, while keeping the values of t_2 and t_3 the same as a function of t_1, we conclude this section with our main result directly following Lemma 14.

▶ **Theorem 15.** *Given a colored tree $\mathcal{T}$, of n nodes, a function g respecting the properties from Section 2, there exists a linear-space data structure, supporting path maximum g-value color queries in $O(\sqrt{n/w})$ time, requiring an $O(n\sqrt{nw})$ preprocessing time.*

7 Faster Path α-Minority Queries on Trees

An α-minority in a multiset A, for some parameter $\alpha \in [0, 1]$, is defined as an element that occurs at least once in A but constitutes no more than an α fraction of the total size of A. The path α-minority query for a tree path (u, v) asks to report any single color that occurs at most an α fraction of the tree path between nodes u and v. Durocher et al. [10]

introduced a linear-space data structure capable of answering path α-minority queries on trees in $O(\alpha^{-1} \log \log n)$ time, where the parameter α is specified at query time, and can differ for every query. In this section, we show that by incorporating randomness, the query time can be improved to $O(\alpha^{-1})$ for a Monte Carlo algorithm (success probability $\geq 1/2$) and to expected $O(\alpha^{-1} + \log \log n)$ for a Las Vegas algorithm (success probability 1).

7.1 $O(\alpha^{-1})$ Time Algorithm with Success Probability $\geq 1/2$

We use the following result from Durocher et al. [10] to efficiently retrieve candidate colors near the endpoints of the path.

▶ **Lemma 16** (Durocher et al. [10]). *There exists an $O(n)$-space data structure that supports k-nearest distinct ancestor queries on trees in $O(k)$ time. The query returns the k distinct colors closest to a node u on the path to the root, in order of increasing distance.*

Let $P(i, j)$ be the query path, and let $w = \text{LCA}(i, j)$. We define the length of the path as $N = |P(i, j)|$. We propose the following 4-phase query algorithm:

- **Phase 1 (Candidate Collection):** We retrieve the sets of distinct colors closest to the endpoints. Let $k = \lceil 2\alpha^{-1} \rceil$.

  $S_l \leftarrow$ the k distinct colors on $P(i, w)$ closest to i.
  $S_r \leftarrow$ the k distinct colors on $P(j, w)$ closest to j.

  Note that if the path segment $P(i, w)$ contains fewer than k distinct colors, S_l will simply contain all distinct colors on that segment (and similarly for S_r).
- **Phase 2 (Pruning Majorities):** We remove obvious majorities. For each $c \in S_l$, if the frequency of c on the sub-path $P(i, w)$ exceeds αN, we remove c from S_l. We perform the symmetric check for S_r on $P(j, w)$. This is valid because if a color is a majority on a sub-path with frequency exceeding the threshold for the *entire* path, it is certainly an α-majority for the whole path.
- **Phase 3 (Exact Verification of Overlap):** Identify colors $c \in S_l \cap S_r$. For these colors, we have identified both their first occurrence from i and their first occurrence from j. Since the sets S_l and S_r are built searching inwards from the endpoints, if c appears in both, we have bounded its occurrences. We can compute the exact frequency of c on $P(i, j)$ in $O(1)$ time (using the precomputed frequency data structures). If the frequency is $\leq \alpha N$, we return c immediately as the answer. We remove all colors $c \in S_l \cap S_r$ from S_l and S_r respectively. Thus, at the end of this phase, $S_l \cap S_r = \emptyset$.
- **Phase 4 (Random Sampling):** If no answer was returned in Phase 3, we construct the set of remaining candidates $D = S_l \cup S_r$. We select a color c uniformly at random from D and return it.

▶ **Lemma 17.** *The query algorithm returns an α-minority in $O(\alpha^{-1})$ time with probability $\geq \frac{1}{2}$.*

Proof. The time complexity is dominated by the retrieval of $O(\alpha^{-1})$ colors and the set operations, which take $O(\alpha^{-1})$ time. The frequency checks in Phase 2 and 3 take $O(1)$ using the standard Level Ancestor and precomputed depth/frequency arrays (as employed in the virtual tree structures).

We analyze the success probability in two cases:

1. **Case 1: The path has few distinct colors.** Suppose the number of distinct colors on $P(i,j)$ is at most $2\lceil 2\alpha^{-1}\rceil$. In this scenario, S_l and S_r effectively cover the entire path. Any color present on the path is in $S_l \cup S_r$. Specifically, any α-minority is in this set. Thus, the algorithm succeeds with probability 1.
2. **Case 2: The path has many distinct colors.** Suppose there are more than $2\lceil 2\alpha^{-1}\rceil$ distinct colors. The number of α-majorities (colors with frequency $> \alpha N$) is strictly less than α^{-1}. Our candidate set $D = S_l \cup S_r$ has size up to $2\lceil 2\alpha^{-1}\rceil \approx 4\alpha^{-1}$. Even in the worst case where every α-majority is present in D, there are at most α^{-1} "bad" candidates. The number of "good" candidates (α-minorities) in D is at least $|D| - \alpha^{-1} \approx 3\alpha^{-1}$. Therefore, the probability of selecting an α-minority at random is at least: $\frac{3\alpha^{-1}}{4\alpha^{-1}} = \frac{3}{4} \geq \frac{1}{2}$. ◀

7.2 Expected $O(\alpha^{-1} + \log\log n)$ Time Algorithm

To achieve a success probability of 1, we extend Phase 4 into a Las Vegas algorithm. Instead of returning the random candidate immediately, we verify it. We can determine the exact endpoints of c on $P(i,j)$ using an $O(\log\log n)$-time predecessor/successor query (using the structure from Lemma 10). Once the endpoints are known, calculating the frequency takes $O(1)$ time. We repeatedly pick a color c at random from $S_l \cup S_r$, and verify its frequency, until an α-minority is found.

Since the probability of success in one sampling step is at least $1/2$, the number of trials follows a geometric distribution with success parameter $p \geq 1/2$. The expected number of trials is at most 2. The total expected time complexity is: $O(\alpha^{-1}) + E[\text{trials}] \cdot O(\log\log n) = O(\alpha^{-1} + \log\log n)$.

References

1 Djamal Belazzougui, Fabiano C Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *European Symposium on Algorithms*, pages 682–693. Springer, 2009. doi:10.1007/978-3-642-04128-0_61.

2 Michael A Bender and Martin Farach-Colton. The lca problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

3 Michael A Bender and Martın Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004. doi:10.1016/J.TCS.2003.05.002.

4 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993. doi:10.1137/0222017.

5 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *Journal of computer and System Sciences*, 48(2):214–230, 1994. doi:10.1016/S0022-0000(05)80002-9.

6 Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014. doi:10.1007/S00224-013-9455-2.

7 Timothy M Chan, Stephane Durocher, Matthew Skala, and Bryan T Wilkinson. Linear-space data structures for range minority query in arrays. In *Scandinavian Workshop on Algorithm Theory*, pages 295–306. Springer, 2012. doi:10.1007/978-3-642-31155-0_26.

8 Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986. doi:10.1137/0215051.

9 Paul F Dietz. Finding level-ancestors in dynamic trees. In *Workshop on Algorithms and Data Structures*, pages 32–40. Springer, 1991. doi:10.1007/BFB0028247.

10 Stephane Durocher, Rahul Shah, Matthew Skala, and Sharma V Thankachan. Linear-space data structures for range frequency queries on arrays and trees. *Algorithmica*, 74:344–366, 2016. doi:10.1007/S00453-014-9947-8.

11 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984. doi:10.1145/828.1884.

12 Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *STACS 2001: 18th Annual Symposium on Theoretical Aspects of Computer Science Dresden, Germany, February 15–17, 2001 Proceedings 18*, pages 317–326. Springer, 2001. doi:10.1007/3-540-44693-1_28.

13 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.

14 Ovidiu Rata. Faster linear-space data structures for path frequency queries, 2026. arXiv:2604.18667.

15 Matthew Skala. *Array Range Queries*, pages 333–350. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_21.

New Algorithms for Girth and Cycle Detection

Liam Roditty
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Plia Trabelsi
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Abstract

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. Let g be the girth of G, that is, the length of a shortest cycle in G. We present a randomized algorithm with a running time of $\tilde{O}\left(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}}\right)$ that returns a cycle of length at most $2\ell\left\lceil\frac{g}{2}\right\rceil - 2\left\lfloor\varepsilon\left\lceil\frac{g}{2}\right\rceil\right\rfloor$, where $\ell \geq 2$ is an integer and $\varepsilon \in [0, 1]$, for every graph with $g = \text{polylog}(n)$.

Our algorithm generalizes an algorithm of Kadria et al. [SODA'22] that computes a cycle of length at most $4\left\lceil\frac{g}{2}\right\rceil - 2\left\lfloor\varepsilon\left\lceil\frac{g}{2}\right\rceil\right\rfloor$ in $\tilde{O}\left(n^{1+\frac{1}{2-\varepsilon}}\right)$ time. Kadria et al. presented also an algorithm that finds a cycle of length at most $2\ell\left\lceil\frac{g}{2}\right\rceil$ in $\tilde{O}\left(n^{1+\frac{1}{\ell}}\right)$ time, where ℓ must be an integer. Our algorithm generalizes this algorithm, as well, by replacing the integer parameter ℓ in the running time exponent with a real-valued parameter $\ell - \varepsilon$, thereby offering greater flexibility in parameter selection and enabling a broader spectrum of combinations between running times and cycle lengths.

We also show that for sparse graphs a better tradeoff is possible, by presenting an $\tilde{O}(\ell \cdot m^{1+\frac{1}{\ell-\varepsilon}})$ time randomized algorithm that returns a cycle of length at most $2\ell(\lfloor\frac{g-1}{2}\rfloor) - 2(\lfloor\varepsilon\lfloor\frac{g-1}{2}\rfloor\rfloor + 1)$, where $\ell \geq 3$ is an integer and $\varepsilon \in [0, 1)$, for every graph with $g = \text{polylog}(n)$.

To obtain our algorithms we develop several techniques and introduce a formal definition of *hybrid* cycle detection algorithms. Both may prove useful in broader contexts, including other cycle detection and approximation problems. Among our techniques is a new cycle searching technique, in which we search for a cycle from a given vertex and possibly all its neighbors in linear time. Using this technique together with more ideas we develop two hybrid algorithms. The first allows us to obtain an $\tilde{O}(m^{2-\frac{2}{\lceil g/2\rceil+1}})$-time, $(+1)$-approximation of g. The second is used to obtain our $\tilde{O}(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}})$-time and $\tilde{O}(\ell \cdot m^{1+\frac{1}{\ell-\varepsilon}})$-time approximation algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph algorithms, All pairs shortest path, Girth, Cycle approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.38

Related Version *Full Version*: https://arxiv.org/abs/2507.02061 [10]

Funding *Liam Roditty*: Supported in part by BSF grants 2016365 and 2020356.

1 Introduction

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. A set of vertices $C_\ell = \{v_1, v_2, \cdots, v_{\ell+1}\}$ in G, where $\ell \geq 2$, is a cycle of length ℓ if $v_1 = v_{\ell+1}$ and $(v_i, v_{i+1}) \in E$, for every $1 \leq i \leq \ell$. A $C_{\leq\ell}$ is a cycle of length at most ℓ. The *girth* g of G is the length of a shortest cycle in G. The girth of a graph has been studied extensively since the 1970s by researchers from both the graph theory and the algorithms communities.

Itai and Rodeh [6] showed that the girth can be computed in $O(mn)$ time or in $O(n^\omega)$ time, where $\omega < 2.371552$ [16], if Fast Matrix Multiplication (FMM) algorithms are used. They also proved that the problem of computing the girth is equivalent to the problem of deciding whether there is a C_3 (triangle) in a graph or not.

In practice, algorithms for FMM have very large constant factors in their running time. Combinatorial algorithms, informally, are algorithms which do not use algebraic methods that are being used by FMM algorithms, and consequently are often more practical. Vassilevska W.

© Liam Roditty and Plia Trabelsi;
licensed under Creative Commons License CC-BY 4.0

and Williams [15] showed that if there exists a truly subcubic time combinatorial algorithm which detects if a graph has a triangle (and therefore also a subcubic time algorithm that computes the exact girth), then there exists a truly subcubic time combinatorial algorithm for Boolean Matrix Multiplication (BMM) (and therefore also for unweighted All Pairs Shortest Path (APSP), see [5], [13], [14]). Such an algorithm would be a major breakthrough. As a result, to get a faster running time for computing the girth, it is natural to settle for an *approximation* algorithm for the girth instead of an exact computation. An (α, β)-approximation $\hat{g}$ of g (where $\alpha \geq 1$ and $\beta \geq 0$), satisfies $g \leq \hat{g} \leq \alpha \cdot g + \beta$. We denote an approximation as an α-approximation if $\beta = 0$ and as a $(+\beta)$-approximation if $\alpha = 1$.

Itai and Rodeh [6] presented a $(+1)$-approximation algorithm that runs in $O(n^2)$ time. Notice that in contrast to the BMM or APSP problems, where a running time of $\Omega(n^2)$ is inevitable since the output size is $\Omega(n^2)$, in the girth problem the output is a single number, thus, there is no natural barrier for subquadratic time algorithms. Indeed, Lingas and Lundell [8] presented a $\frac{8}{3}$-approximation algorithm that runs in $\tilde{O}(n^{3/2})$ time, and Roditty and V. Williams [12] presented a 2-approximation algorithm that runs in $\tilde{O}(n^{5/3})$ time. Dahlgaard, Knudsen and Stöckel [4] presented two tradeoffs between running time and approximation. One generalizes the algorithms of [8, 12] and computes a cycle of length at most $2\lceil \frac{g}{2} \rceil + 2\lceil \frac{g}{2(\ell-1)} \rceil$ in $\tilde{O}(n^{2-1/\ell})$ time. The other computes, whp, a $C_{\leq 2^\ell g}$, for any integer $\ell \geq 2$, in $\tilde{O}(n^{1+1/\ell})$ time.

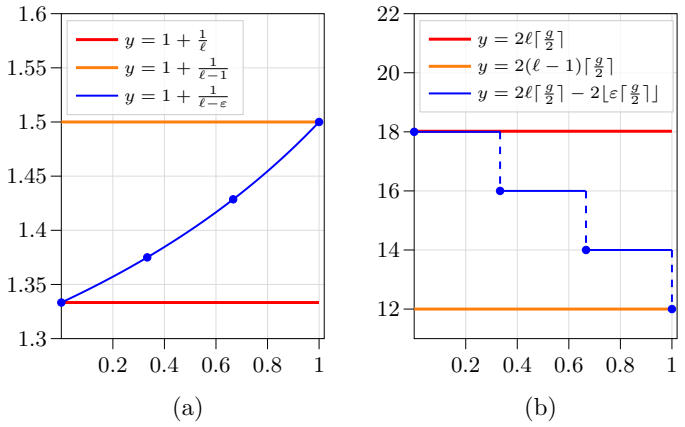
Kadria et al. [7] significantly improved upon the second algorithm of [4] and presented an algorithm, that for every integer $\ell \geq 1$, computes a $C_{\leq 2\ell\lceil g/2 \rceil}$ in $\tilde{O}(n^{1+1/\ell})$ time. They also presented an algorithm, that for every $\varepsilon \in (0, 1)$, computes a cycle of length at most $4\lceil \frac{g}{2} \rceil - 2\lfloor \varepsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\varepsilon)g + 4$, in $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time, for every graph with $g = \text{polylog}(n)$.

These two algorithms of Kadria et al., as well as few other approximation algorithms (see for example, [8], [2], [9]), were obtained using a general framework for girth approximation in which a search is performed over the range of possible values of g, using some algorithm $\mathcal{A}$ that gets as an input an integer $\tilde{g}$ which is a guess for the value of g. In each step of the search $\mathcal{A}$ either returns a cycle $C_{\leq f(\tilde{g})}$, where f is a non decreasing function, or determines that $g > \tilde{g}$. The goal of the search is to find the smallest $\tilde{g}$, for which $\mathcal{A}$ returns a cycle, because for this value we have $g > \tilde{g} - 1$ (and thus $g \geq \tilde{g}$), and algorithm $\mathcal{A}$ returns a $C_{\leq f(\tilde{g})}$. This cycle is of length at most $f(g)$ since $g \geq \tilde{g}$ and f is a non decreasing function (f can represent the approximation, for example $f(\tilde{g}) = 2\tilde{g}$ yields a 2-approximation). The two possible outcomes of $\mathcal{A}$ and its usage in the general girth approximation framework inspired us to formally define the notion of a (γ, δ)-*hybrid* algorithm as follows:

▶ **Definition 1.** *A (γ, δ)-hybrid algorithm is an algorithm that either outputs a $C_{\leq \gamma}$ or determines that $g > \delta$.*

When $\gamma = \delta$, the algorithm is referred to as a γ-hybrid algorithm. The girth approximation framework described above suggests that a possible approach for developing efficient girth approximation algorithms is by developing efficient (γ, δ)-hybrid algorithms.

Kadria et al. [7] designed several algorithms that satisfy the definition of (γ, δ)-hybrid algorithms. Their girth approximation algorithms mentioned above were obtained using two different $(f(\tilde{g}), \tilde{g})$-hybrid algorithms. Additionally, for every integer $k \geq 2$, they presented a $(2k, 3)$-hybrid and a $(2k, 4)$-hybrid algorithms that run in $O(\min\{m^{1+1/(k+1)}, n^{1+2/k}\})$ time, and a $(\max\{2k, g\}, 5)$-hybrid algorithm that runs in $O(\min\{m^{1+2/(k+1)}, n^{1+3/k}\})$ time. Therefore, for $k \geq 2$, $\tilde{g} \in \{3, 4, 5\}$ and $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$, there is a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs in $O(\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time. A natural question is whether these three algorithms are only part of a general tradeoff between the running time, γ and δ.



ε	Time exp	Cycle bound
0	$1+\frac{3}{9}$	18
$\frac{1}{3}$	$1+\frac{3}{8}$	16
$\frac{2}{3}$	$1+\frac{3}{7}$	14
1	$1+\frac{3}{6}$	12

Figure 1 Our $\tilde{O}(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}})$-time girth approximation algorithm compared to the $\tilde{O}(n^{1+\frac{1}{\ell}})$-time algorithm of [7], for every $\varepsilon \in [0,1]$, choosing $\ell = 3$ and $g = 5$ or 6. (a) The y-axis is the exponent of n in the running time, and the x-axis is ε. (b) The y-axis is the upper bound on the length of the returned cycle, and the x-axis is ε. The blue points correspond to our algorithm at four specific choices of ε: $\varepsilon = 0, \frac{1}{3}, \frac{2}{3}, 1$ (see the table on the right).

▶ **Problem 1.1.** *Let $\tilde{g} \geq 6$ and $k \geq 2$ be two integers and let $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$. Is it possible to obtain a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs in $O(\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time?*

In this paper we present a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs, whp, in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time, for every $\tilde{g} \geq 3$. This algorithm provides an affirmative answer to Problem 1.1, albeit the $(\frac{k+1}{\alpha-1} + \alpha)$ factor in the running time.

Kadria et al. [7] presented also a $(2k, 6)$-hybrid algorithm whose running time is $\tilde{O}(\min\{nm^{\frac{1}{2}+\frac{1}{2(k+1)}}, n^{\frac{3}{2}+\frac{1}{k}}\})$, for every integer $k \geq 4$. The running time of our algorithm when $\tilde{g} = 6$ is $\tilde{O}(k \cdot \min\{m^{1+\frac{2}{k+1}}, n^{1+\frac{3}{k}}\})$. We improve the running time for every constant $k \geq 5$, in the case of $m < O(n^{1+1/k})$ (when $m \geq O(n^{1+1/k})$ the two algorithms run a similar procedure and thus have the same asymptotic running time).

Using our $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm we obtain a generalization of the $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time algorithm of Kadria et al. [7] that computes a cycle of length at most $4\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (2-\varepsilon)g + 4$, for every graph with $g = \text{polylog}(n)$. Our generalized algorithm runs in $\tilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$ time, whp, and returns a cycle of length at most $2\ell\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (\ell-\varepsilon)g + \ell + 2$, where $\ell \geq 2$ is an integer and $\varepsilon \in [0,1]$, for every graph with $g = \text{polylog}(n)$. We also show that if the graph is sparse then the approximation can be improved. More specifically, we present an algorithm that runs in $\tilde{O}(\ell \cdot m^{1+1/(\ell-\varepsilon)})$ time, whp, and returns a cycle of length at most $(\ell-\varepsilon)g - \ell + 2\varepsilon$, where $\ell \geq 3$ is an integer and $\varepsilon \in [0,1)$, for every graph with $g = \text{polylog}(n)$.

Our $\tilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$-time algorithm also generalizes the $\tilde{O}(n^{1+1/\ell})$-time algorithm of Kadria et al. [7], which computes a $C_{\leq 2\ell\lceil g/2\rceil}$ for every **integer** $\ell \geq 1$. In our algorithm, the integer parameter ℓ that appears in the exponent of the running time is replaced by a real-valued parameter $\ell - \varepsilon$. Thus, we introduce many new points on the tradeoff curve between running time and approximation ratio. Specifically, for every integer $\ell \leq \text{polylog}(n)$, up to $\lceil g/2 \rceil - 1$ additional tradeoff points are added.[1] For example, consider $\ell = 3$ and a

[1] Up to $\lceil g/2 \rceil - 1$ tradeoff points are added since for every such ℓ, when ε is a multiple of $\frac{1}{\lceil g/2 \rceil}$, we get an $\tilde{O}(n^{1+1/(\ell-\varepsilon)})$-time algorithm which computes a $C_{\leq 2(\ell-\varepsilon)\lceil g/2 \rceil}$.

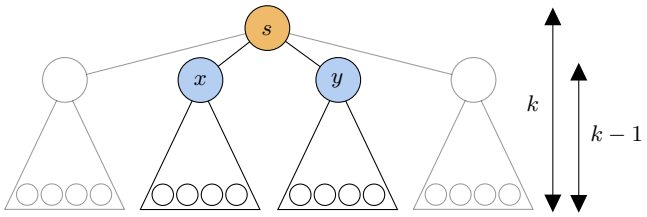


Figure 2 Disjoint sets of vertices at distance $k-1$ from x and y.

graph with girth $g = 5$ or $g = 6$. Our algorithm yields two additional points on the tradeoff curve, corresponding to $\varepsilon = \frac{1}{3}$ and $\varepsilon = \frac{2}{3}$. For $\varepsilon = \frac{1}{3}$, we compute a $C_{\leq 16}$ in $\tilde{O}(n^{1+\frac{3}{8}})$ time, and for $\varepsilon = \frac{2}{3}$, we compute a $C_{\leq 14}$ in $\tilde{O}(n^{1+\frac{3}{7}})$ time. These points lie between the two points on the tradeoff curve given by the algorithm of Kadria et al. [7], which computes either a $C_{\leq 12}$ in $\tilde{O}(n^{1+\frac{3}{6}})$ time or a $C_{\leq 18}$ in $\tilde{O}(n^{1+\frac{3}{9}})$ time. See Figure 1 for a comparison.

Together, the tradeoffs in the two algorithms of Kadria et al. [7] that we generalize encompass several other known results, including those of Itai and Rodeh [6], Lingas and Lundell [8], Roditty and V. Williams [12] and the first tradeoff of Dahlgaard et al. [4] (when $g = \text{polylog}(n)$). Notably, some of these algorithms have resisted improvement for many years. Therefore, the unification of these algorithms within a single tradeoff curve in our result, together with the addition of new valid points on this curve, reinforces the possibility that it captures a fundamental relationship between running time and approximation quality. This, in turn, motivates further investigation into whether a matching lower bound exists for this tradeoff.

The rest of this paper is organized as follows. In Section 2 we provide an overview. Preliminaries are in Section 3. In Section 4, we present a new cycle searching technique that is used by our algorithms. In Section 5 we present a $2k$-hybrid algorithm and then use it to obtain a $(+1)$-approximation algorithm for the girth. In Section 6 we generalize the $2k$-hybrid algorithm and present a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm. In Section 7 we use the hybrid algorithm from Section 6 to obtain two more approximation algorithms for the girth.

2 Overview

Among the techniques that we develop to obtain our new algorithms, is a new cycle searching technique that might be of independent interest. Our new technique exploits the property that if $s \in V$ is not on a $C_{\leq 2k}$, then for any two neighbors x and y of s, the set of vertices at distance exactly $k-1$ from x and y that are also at distance k from s are disjoint (see Figure 2). This allows us to check efficiently for all the neighbors of s if they are on a $C_{\leq 2k}$. Using this technique, together with more tools that we develop, we obtain *two* hybrid algorithms.

The first is a relatively simple $O(m^{1+\frac{k-1}{k+1}})$-time, $2k$-hybrid algorithm. We use this hybrid algorithm in the girth approximation framework described earlier, to obtain an $\tilde{O}(m^{1+\frac{\ell-1}{\ell+1}})$-time, $(+1)$-approximation of the girth, where $g = 2\ell$ or $g = 2\ell - 1$ (the running time can also be written as $\tilde{O}(m^{2-\frac{2}{\lceil g/2 \rceil + 1}})$). We remark that using an algorithm of [3] for C_{2k} detection, it is possible to obtain a $(+1)$-approximation in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time (see Section 5.2 for more details). However, the additional $\ell^{O(\ell)}$ factor might be significant even for small values of ℓ.

The second is the $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that solves Problem 1.1. Its main component is an $(2k, 2\alpha)$-hybrid algorithm that runs in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$-time, whp, and generalizes the first $O(m^{1+\frac{k-1}{k+1}})$-time $2k$-hybrid algorithm, by introducing an additional parameter $\alpha \leq k$. Using α we can tradeoff between the running time and the lower bound on g and obtain a faster running time at the price of a worse lower bound.

We compare our $(2k, 2\alpha)$-hybrid algorithm to algorithm `Cycle` of Kadria et al. [7], an $O(m + n^{1+\frac{1}{\ell}})$-time,[2] $(2\ell\alpha, \tilde{g})$-hybrid algorithm, where $\alpha = \lceil \tilde{g}/2 \rceil$, that they used to obtain the $\tilde{O}(n^{1+\frac{1}{\ell}})$-time, $2\ell\lceil g/2 \rceil$-approximation algorithm. As we show later, the running time of our $(2k, 2\alpha)$-hybrid algorithm can be bounded by $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot n^{1+\frac{\alpha}{k}})$. Since in our algorithm k is not necessarily a multiple of α (compared to the $\ell\alpha$ of `Cycle`), our algorithm allows more flexibility, and we achieve many more possible tradeoffs between the running time and the output cycle length. For example, if we consider a multiplicative approximation better than 3, when the value of g is a constant known in advance, our algorithm can return longer cycles that are still shorter than $3g$, in a faster running time.[3]

The flexibility of our algorithm is also demonstrated as follows. For a given constant value of k, if our $(2k, 2\alpha)$-hybrid algorithm returns a cycle then its length is at most $2k$. If we want algorithm `Cycle` to output a $C_{\leq 2k}$, then $\lfloor \frac{k}{\alpha} \rfloor$ is the largest ℓ that we can choose, since ℓ must be an integer. The running time is $O(n^{1+1/\ell}) = O(n^{1+1/\lfloor \frac{k}{\alpha} \rfloor})$. Our algorithm achieves a better running time if k is not divisible by α. (See the relevant figures in the full version of this paper [10] for a comparison).

Next, we overview our $(2k, 2\alpha)$-hybrid algorithm that either finds a $C_{\leq 2k}$ or determines that $g > 2\alpha$ in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time. In order to determine that $g \geq 2\alpha$, we can check for every $v \in V$ if v is on a $C_{\leq 2\alpha}$ (and either find a $C_{\leq 2k}$ since $\alpha \leq k$, or return that $g > 2\alpha$). If v is on a $C_{\leq 2\alpha}$, then all the vertices and edges of this $C_{\leq 2\alpha}$ are at distance at most α from v. If, for every $v \in V$, the number of edges at distance at most α is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$ then using standard techniques we can check for every $v \in V$ if v is on a $C_{\leq 2\alpha}$ in $O(m^{1+\frac{\alpha-1}{k+1}})$ time. However, this is not necessarily the case, and the region at distance at most α from some vertices might be dense. To deal with dense regions within the promised running time we develop an iterative sampling procedure (see `BfsSample` in Section 6), whose goal is to sparsify the graph, or to return a $C_{\leq 2k}$. One component of the iterative sampling procedure is a generalization of our new cycle searching technique mentioned above. In the generalization instead of checking whether a vertex s and its neighbors are on a $C_{\leq 2k}$, we check whether all the vertices up to a possibly further distance from s are on a $C_{\leq 2\alpha}$, for $\alpha \leq k$, and if not we mark them so that they can be removed later.

If the iterative sampling procedure ends without finding a $C_{\leq 2k}$ then there are two possibilities. Let $r = (k+1) \bmod (\alpha - 1)$. If $r = 0$ then it holds that the number of edges at distance at most α from every $v \in V$ is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$, whp, as required. If $r > 0$ then it holds that the number of edges at distance at most r from every $v \in V$ is $O(m^{\frac{r}{k+1}})$, whp. This does not necessarily imply that the graph is sparse enough for checking whether $g > 2\alpha$. In this case, we run another algorithm (see `HandleRemainder` in Section 6) that continues

[2] `Cycle` runs in $O(n^{1+\frac{1}{\ell}} + m)$ time, which can be reduced to $O(n^{1+\frac{1}{\ell}})$ time, as shown in [7].

[3] [7] presented also an $O((\alpha - c) \cdot n^{1+\frac{\alpha}{2\alpha-c}})$-time, $(4\alpha - 2c, 2\alpha)$-hybrid algorithm, where $0 < c \leq \alpha$ are integers. For $c = 2\alpha - k$, this is an $O((k - \alpha) \cdot n^{1+\frac{\alpha}{k}})$-time, $(2k, 2\alpha)$-hybrid algorithm, similar to our $(2k, 2\alpha)$-hybrid algorithm. However, since $0 < c \leq \alpha$, the possible values of k are restricted and must satisfy $\alpha \leq k < 2\alpha$. By choosing $\alpha = \lceil \frac{g}{2} \rceil$ and an appropriate k the two algorithms have a similar flexibility for a 2-approximation, but since in our algorithm also larger values of k are allowed, we can achieve a faster running time for a t-approximation where $t > 2$.

to sparsify the graph until the number of edges at distance at most α from every $v \in V$ is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$ and checking whether $g > 2\alpha$ is possible within the required running time of $O(m^{1+\frac{\alpha-1}{k+1}})$.

3 Preliminaries

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. Let $U \subseteq V$ be a set of vertices and let $G \setminus U$ be the graph obtained from G by deleting all the vertices of U together with their incident edges. For two graphs $G = (V, E)$ and $G' = (V', E')$, let $G \setminus G'$ be $G \setminus V'$. We say that $G \subseteq G'$ if $V \subseteq V'$ and $E \subseteq E'$. For convenience, we use both $u \in V$ and $u \in G$ to say that $u \in V$. For every $u, v \in V$, let $d_G(u, v)$ be the length of a shortest path between u and v in G. The *girth* g of G is the length of a shortest cycle in G. Let $wt(C)$ be the length of a cycle C. For an integer ℓ, we denote a cycle of length (at most) ℓ by $(C_{\leq \ell})$ C_ℓ.[4] Let $E(v)$ be the edges incident to v and $E(v, i)$ the ith edge in $E(v)$. Let $deg_G(v)$ be the degree of v in G. Let $N(v)$ be the set of neighbors of v, namely $N(v) = \{w \mid (v, w) \in E\}$. For an edge set S, let $V(S)$ be the endpoints of S's edges, that is, $V(S) = \{u \in V \mid \exists (u, v) \in S\}$. Let $e = (u, v) \in E$ and $w \in V$. The distance $d_G(w, e)$ between w and e is $\min\{d_G(w, u), d_G(w, v)\} + 1$. For every $u \in V$ and a real number k let $B(G, u, k) = (V_u^k(G), E_u^k(G))$ be the *ball graph* of u, where $V_u^k(G) = \{v \in V \mid d_G(u, v) \leq k\}$ and $E_u^k(G) = \{e \in E \mid d_G(u, e) \leq k\}$ [7]. For an integer $\ell \geq 0$ and a vertex $u \in V$, Let $L_u^\ell(G) = \{w \mid d_G(u, w) = \ell\}$.[5]

We now turn to present several essential tools that are required in order to obtain our new algorithms. We first restate an important property of the ball graph $B(v, R)$.

▶ **Lemma 2** ([7]). *Let $0 \leq t \leq R$ be two integers and let $v \in V$. If $B(v, R)$ is a tree then no vertex in V_v^{R-t} is part of a cycle of length at most $2t$ in G.*

We use procedure `BallOrCycle`(G, v, R) [7, 8] (pseudo-code in [10]) that searches for a $C_{\leq 2R}$ in the ball graph $B(v, R)$. We summarize `BallOrCycle`'s properties in the next lemma.

▶ **Lemma 3** ([7]). *Let $v \in V$. If the ball graph $B(v, R)$ is not a tree then* `BallOrCycle`(G, v, R) *returns a $C_{\leq 2R}$ from $B(v, R)$. If $B(v, R)$ is a tree then* `BallOrCycle`(G, v, R) *returns V_v^R.*[6] *The running time of* `BallOrCycle`(G, v, R) *is $O(|V_v^R|)$.*

Next, we obtain a simple t-hybrid algorithm, called `AllVtxBallOrCycle`, using `BallOrCycle`. `AllVtxBallOrCycle` (pseudo-code in [10]) gets a graph G and an integer $t \geq 2$, and runs `BallOrCycle`(v, t) from every $v \in G$ as long as no cycle is found by `BallOrCycle`. If `BallOrCycle` finds a cycle then `AllVtxBallOrCycle` stops and returns that cycle. If no cycle is found then `AllVtxBallOrCycle` returns `null`. We prove the next lemma in the full version of this paper [10].

▶ **Lemma 4.** `AllVtxBallOrCycle`(G, t) *either finds a $C_{\leq 2t}$ or determines that $g > 2t$, in $O(\sum_{v \in V} |V_v^t|)$ time.*

We now show that if the input graph satisfies a certain sparsity property then the running time of `AllVtxBallOrCycle`(G, t) can be bounded as follows.

---

[4] Both C_ℓ and $C_{\leq \ell}$ might not be simple cycles. However, the cycles that our algorithms return are simple.
[5] When the graph is clear from the context, we sometimes omit G from the notations.
[6] If V_v^R is returned then we assume that V_v^R is ordered by the distance from v, and for every $u \in V_v^R$ we store $d(u, v)$ with u. Thus, given the set V_v^R, we can find $V_v^{R'}$ for every $R' < R$ in $O(|V_v^{R'}|)$ time.

Algorithm 1 `SparseOrCycle`(G, D, x, y).

```
1 foreach w ∈ V do
2     if IsDense(G, w, D^x, x) = Yes then
3         (V_w^{x-1+y}, C) ← BallOrCycle(w, x − 1 + y);
4         if C ≠ null then return C;
5         G ← G \ V_w^{x-1};
6 return null;
```

▶ **Corollary 5.** *If $|E_u^{t-1}| < D^{t-1}$ for every $u \in V$ then* `AllVtxBallOrCycle`(G, t) *runs in $O(mD^{t-1})$ time.*

Proof. For every $v \in V$, we have $O(|V_v^t|) \leq O(|E_v^t|) \leq O(\sum_{u \in N(v)} |E_u^{t-1}|)$, which is at most $O(\sum_{u \in N(v)} D^{t-1}) = O(deg(v) \cdot D^{t-1})$. Thus, by Lemma 4, the running time of `AllVtxBallOrCycle`(G, t) is $O(\sum_{v \in V} |V_v^t|) \leq O(\sum_{v \in V} deg(v) \cdot D^{t-1}) = O(mD^{t-1})$. ◀

Next, we present procedure `IsDense`(G, w, T, r) from [7]. `IsDense` (pseudo-code in [10]) gets a graph G, a vertex w, a budget $T \geq 1$ (real) and a distance $r \geq 0$ (integer). In the procedure a BFS is executed from w. The BFS counts the edges that are scanned as long as their total number is less than T and the farthest vertex from w is at distance at most r.

▶ **Lemma 6** ([7])**.** *Procedure* `IsDense`(G, w, T, r) *runs in $O(\lceil T \rceil) = O(T)$ time. If* `IsDense`$(G, w, T, r) =$ `No` *then $|E_w^r| < T$. If* `IsDense`$(G, w, T, r) =$ `Yes` *then $|E_w^r| \geq T$.*

Given a vertex v and a distance R we sometimes want to bound $|E_v^R|$. Therefore, we adapt a lemma and a corollary of [7] from vertices to edges. The proofs are in the full version of this paper [10].

▶ **Lemma 7.** *Let x, y be positive integers, let $D \geq 1$ be a real number, and let $w \in V$. If $|E_w^x| < D^x$, and $|E_u^y| < D^y$ for every $u \in V$, then $|E_w^{x+y}| < D^{x+y}$.*

▶ **Corollary 8.** *Let x be a positive integer and let $D \geq 1$ be a real number. If $|E_w^x| < D^x$ for every $w \in V$, then $|E_w^{ix}| < D^{ix}$, for every $w \in V$ and $i \geq 1$.*

We also adapt procedure `SparseOrCycle`(G, D, x, y) of [7] to our needs. `SparseOrCycle` (see Algorithm 1) gets a graph G, a parameter $D \geq 1$, and two integers $x, y > 0$, and iterates over vertices using a for-each loop. Let w be the vertex currently considered and $G(w)$ the current graph. If `IsDense`$(G(w), w, D^x, x) =$ `Yes` then `BallOrCycle`$(G(w), w, x - 1 + y)$ is called. If `BallOrCycle` returns a cycle C then C is returned by `SparseOrCycle`. Otherwise, the vertex set V_w^{x-1} is removed from $G(w)$ along with the edge set E_w^x. After the loop ends, if no cycle was found, we return `null`. Let $W \subseteq V$ be the set of vertices for which `BallOrCycle` was called and no cycle was found, and $\hat{G} = (\hat{V}, \hat{E})$ the graph after `SparseOrCycle` ends. The following lemma is similar to the corresponding lemmas from [7]. For completeness, the proof is provided in the full version of this paper [10].

▶ **Lemma 9.** *`SparseOrCycle`(G, D, x, y) satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2(x - 1 + y)$*
- **(ii)** *If a cycle is not returned then $|E_u^x| < D^x$, for every $u \in \hat{G}$*
- **(iii)** *If $u \in G \setminus \hat{G}$ then u is not part of a $C_{\leq 2y}$ in G*
- **(iv)** *`SparseOrCycle`(G, D, x, y) runs in $O(nD^x + \sum_{w \in W}(|V_w^{x-1+y}|))$ time.*

Similarly to `AllVtxBallOrCycle`, we show for `SparseOrCycle` that if G satisfies a certain sparsity property, the running time can be bounded as follows. The proof is in the full version [10].

▶ **Corollary 10.** *If* $|E_u^{x-1+y}| < D^{x-1+y}$ *for every vertex* $u \in V$ *then* `SparseOrCycle`(G, D, x, y) *runs in* $O(nD^x + mD^{y-1})$ *time.*

Finally, we include a standard lemma about sampling a hitting set (see, e.g., [1], [8], [11]).

▶ **Lemma 11.** *It is possible to obtain in* $O(m)$ *time, using sampling, a set of edges* S *of size* $\tilde{\Theta}(\frac{m}{s})$*, that hits, whp, the* s *closest edges of every* $v \in V$*.*

We remark that some of our algorithms get a graph G that is being updated during their run. Within their scope, G denotes the current graph that includes all updates done so far.

4 A new cycle searching technique

In this section we introduce a new cycle searching technique implemented in algorithm `NbrBallOrCycle`. This technique exploits the property that a vertex $s \in V$ is not on a $C_{\leq 2k}$, to check efficiently whether any neighbor of s lies on a $C_{\leq 2k}$.

Consider a vertex $s \in V$. It is straightforward to check whether s is on a $C_{\leq 2k}$, for every integer k, using `BallOrCycle`(G, s, k) in $O(n)$ time. If `BallOrCycle`(G, s, k) does not return a $C_{\leq 2k}$ then for every $x, y \in N(s)$ it holds that $V_x^{k-1}(G') \cap V_y^{k-1}(G') = \emptyset$, where $G' = G \setminus \{s\}$, as otherwise there was a $C_{\leq 2k}$ passing through s and `BallOrCycle`(G, s, k) would have returned a $C_{\leq 2k}$ (see Figure 2). We show that it is possible to exploit this property to check for every $v \in N(s)$ whether v is on a $C_{\leq 2k}$, using `BallOrCycle`(G', v, k), in $O(n+m)$ time instead of $O(deg(s) \cdot n)$. More specifically, we present algorithm `NbrBallOrCycle`(G, s, k) (see Algorithm 2) that gets a graph G, a vertex s, and an integer $k \geq 2$. We first initialize $\hat{U}$ to $\emptyset$. Then, we run `BallOrCycle`(G, s, k). If a cycle C is found by `BallOrCycle`(G, s, k) then C is returned by `NbrBallOrCycle`. Otherwise, we add the vertex s to $\hat{U}$, keep the neighbors of s in N_s, and then remove s from G. Recall that G' equals $G \setminus \{s\}$. Next, for every $v \in N_s$ we run `BallOrCycle`(G', v, k), as long as a cycle is not found. If a cycle C is returned by `BallOrCycle`(G', v, k) then C is returned by `NbrBallOrCycle`.[7] Otherwise, v is added to $\hat{U}$. After the loop ends, the vertex s and its adjacent edges are added back to the graph, and the set $\hat{U}$ is returned by `NbrBallOrCycle`. We prove the following lemma in the full version of this paper [10].

▶ **Lemma 12.** *If algorithm* `NbrBallOrCycle`(G, s, k) *finds a cycle* C *then* $wt(C) \leq 2k$*. Otherwise, no vertex in* V_s^1 *is part of a* $C_{\leq 2k}$ *in* G*, and the set* $\hat{U} = V_s^1$ *is returned.*

To bound the running time of `NbrBallOrCycle`, we show how to use the fact that no $C_{\leq 2k}$ was found by `BallOrCycle`(G, s, k), to efficiently run `BallOrCycle`(G', v, k) for every $v \in N_s$.

▶ **Lemma 13.** *Let* $s \in V$*. If the ball graph* $B(G, s, k)$ *is a tree then the total cost of running* `BallOrCycle`(G', v, k) *for every* $v \in N_s$ *is* $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w))$*.*

[7] For our needs it suffices to stop and return a cycle passing through an s neighbor once we find one, though `BallOrCycle` can be run from all the neighbors of s in the same running time bound of $O(n+m)$.

Algorithm 2 `NbrBallOrCycle`(G, s, k).

```
1  Û ← ∅;
2  (V_s^k, C) ← BallOrCycle(s, k);
3  if C ≠ null then return (∅, C);
4  Û ← Û ∪ {s}, N_s ← N(s), G ← G \ {s};
5  foreach v ∈ N_s do
6      (V_v^k, C) ← BallOrCycle(v, k);
7      if C ≠ null then return (∅, C);
8      Û ← Û ∪ {v} ;
9  add s and the edge set {(s, v) | v ∈ N_s} back to G;
10 return (Û, null);
```

Proof. By the definitions of V_s^k, L_s^k and G', we know that $V_s^k(G) = \cup_{v \in N_s} V_v^{k-1}(G') \cup \{s\}$ and $L_s^k(G) = \cup_{v \in N_s} L_v^{k-1}(G')$. Since $B(G, s, k)$ contains no cycles it follows that $V_x^{k-1}(G') \cap V_y^{k-1}(G') = \emptyset$ and $L_x^{k-1}(G') \cap L_y^{k-1}(G') = \emptyset$, for any two distinct vertices $x, y \in N_s$. Therefore, **(i)** $|V_s^k(G)| = \sum_{v \in N_s} |V_v^{k-1}(G')| + 1$, and **(ii)** $L_s^k(G) = \cup_{v \in N_s} L_v^{k-1}(G')$. From Lemma 3 it follows that the total cost of the calls to `BallOrCycle`(G', v, k) for every $v \in N_s$ is $O(\sum_{v \in N_s} |V_v^k(G')|)$. It holds that $|V_v^k(G')| \leq |V_v^{k-1}(G')| + \sum_{w \in L_v^{k-1}(G')} deg(w)$, for every $v \in N_s$. Thus, we get that the total cost is $O(\sum_{v \in N_s} |V_v^k(G')|) = O(\sum_{v \in N_s} (|V_v^{k-1}(G')| + \sum_{w \in L_v^{k-1}(G')} deg(w)))$. This equals to $O(\sum_{v \in N_s} |V_v^{k-1}(G')| + \sum_{v \in N_s} \sum_{w \in L_v^{k-1}(G')} deg(w)))$, and it follows from **(i)** and **(ii)** that this is at most $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w))$. ◀

We use Lemma 13 to bound the running time of `NbrBallOrCycle`.

▶ **Lemma 14.** *Algorithm `NbrBallOrCycle` runs in $O(n + m)$ time.*

Proof. Running `BallOrCycle`(G, s, k) costs $O(n)$. If a cycle is found, it is returned and the running time is $O(n) = O(n+m)$. If no cycle is found, then removing (and later adding back) s and its edges costs $O(deg(s))$. By Lemma 13, the cost of running `BallOrCycle`(G', v, k) for every $v \in N_s$ is $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w)) = O(n + m)$. Adding s and N_s to $\hat{U}$ takes $O(V_s^1) = O(n)$ time. Thus, the total running time of `NbrBallOrCycle` is $O(n + m)$. ◀

5 A $2k$-hybrid algorithm and a $(+1)$-approximation of the girth

In this section we first show how to use algorithm `NbrBallOrCycle` from the previous section to obtain a $2k$-hybrid algorithm that in $O(m^{1+\frac{k-1}{k+1}})$ time, either returns a $C_{\leq 2k}$ or determines that $g > 2k$. Then, we use the $2k$-hybrid algorithm to compute a $(+1)$-approximation of g.

5.1 A $2k$-hybrid algorithm

We first present algorithm `2-SparseOrCycle`(G, k) that gets a graph G and an integer $k \geq 2$. Let G_0 (G_1) be G before (after) running `2-SparseOrCycle`. `2-SparseOrCycle`(G, k) either finds a $C_{\leq 2k}$ or removes vertices that are not on a $C_{\leq 2k}$, such that for every $u \in G_1$, the ball graph $B(G_1, u, 2)$ is relatively sparse, that is, $|E_u^2(G_1)| < m^{\frac{2}{k+1}}$. `2-SparseOrCycle` (pseudo-code in [10]) iterates over vertices using a for-each loop. Let s be the vertex currently considered. If $\sum_{v \in N(s)} deg(v) \geq m^{\frac{2}{k+1}}$ then `NbrBallOrCycle`(G, s, k) is called. If `NbrBallOrCycle` returns a cycle C then `2-SparseOrCycle` returns C. If `NbrBallOrCycle` returns a vertex set $\hat{U}$ then $\hat{U}$ is removed from G. After the loop ends, if no cycle was found, we return `null`.

▶ Remark. Notice that `SparseOrCycle`$(G, m^{\frac{2}{k+1}}, 2, k)$ either finds a $C_{\leq 2k+2}$ or removes vertices that are not on a $C_{\leq 2k}$, such that for every $u \in G_1$ it holds that $|E^2_u(G_1)| < m^{\frac{2}{k+1}}$. Using `NbrBallOrCycle` instead of `BallOrCycle` in 2-`SparseOrCycle` enables us in the case that a cycle is found to bound the cycle length with $2k$ rather than $2k+2$, while still maintaining the property that $|E^2_u(G_1)| < m^{\frac{2}{k+1}}$, for every $u \in G_1$, in the case that no cycle is found.

We prove the following lemma in the full version of this paper [10].

▶ **Lemma 15.** *2-`SparseOrCycle`(G, k) satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2k$*
- **(ii)** *If a cycle is not returned then $|E^2_u| < m^{\frac{2}{k+1}}$, for every $u \in G_1$*
- **(iii)** *If $u \in G_0 \setminus G_1$ then u is not part of a $C_{\leq 2k}$ in G_0*
- **(iv)** *2-`SparseOrCycle`(G, k) runs in $O(m^{1+\frac{k-1}{k+1}})$ time.*

Next, we use 2-`SparseOrCycle` to design a $2k$-hybrid algorithm called $2k$-`Hybrid`. Notice first that if $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$ for every $u \in V$, then it is straightforward to obtain an $O(m^{1+\frac{k-1}{k+1}})$-time $2k$-hybrid algorithm, by running `AllVtxBallOrCycle`(G, k). Thus, in $2k$-`Hybrid` we ensure that if we call `AllVtxBallOrCycle` then it holds for every $u \in V$ that $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$. To do so, we run 2-`SparseOrCycle` and possibly `SparseOrCycle`. If no cycle was returned then we have $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$ for every $u \in V$, and we can safely run `AllVtxBallOrCycle`.

$2k$-`Hybrid` (pseudo-code in [10]) gets a graph G and an integer $k \geq 2$. $2k$-`Hybrid` is composed of three stages. In the first stage we call 2-`SparseOrCycle`(G, k). If 2-`SparseOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise we proceed to the second stage. In the second stage, if $(k-1) \bmod 2 \neq 0$, we call `SparseOrCycle`$(G, m^{\frac{1}{k+1}}, 1, k)$. If `SparseOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise we proceed to the last stage. In the last stage, we call `AllVtxBallOrCycle`(G, k). If `AllVtxBallOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise $2k$-`Hybrid` returns `null`. We prove the next lemma in the full version of this paper [10].

▶ **Lemma 16.** *$2k$-`Hybrid`(G, k) either returns a $C_{\leq 2k}$ or determines that $g > 2k$, in $O(m^{1+\frac{k-1}{k+1}})$ time.*

5.2 A $(+1)$-approximation of the girth

Next, we describe algorithm `AdtvGirthApprox`, which uses $2k$-`Hybrid` and the framework described in Section 1, to obtain a $(+1)$-approximation of g, when $g \leq \log n$. `AdtvGirthApprox` (pseudo-code in [10]) gets a graph G. In `AdtvGirthApprox`, we set k to 2 and start a while loop. In each iteration, we create a copy G' of G and call $2k$-`Hybrid`(G', k). If $2k$-`Hybrid` finds a cycle C then `AdtvGirthApprox` stops and returns C, otherwise we increment k by 1 and continue to the next iteration. We prove the following theorem in the full version of this paper [10].

▶ **Theorem 17.** *Algorithm `AdtvGirthApprox`(G) returns either a C_g or a C_{g+1}, and runs in $\tilde{O}(m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$ and $2 \leq \ell < \log n$.*[8]

[8] We note that when $\ell \geq \log n$, the $O(n^2)$-time, $(+1)$-approximation of Itai and Rodeh [6] can be used.

▶ Remark. Dahlgaard, Knudsen and Stöckel [3] showed that a C_{2k}, if exists, can be found in $O(k^{O(k)} \cdot m^{\frac{2k}{k+1}})$ time. It is possible to use their C_{2k} detection algorithm to obtain a $(+1)$-approximation for the girth in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$, as mentioned in Section 2. This $(+1)$-approximation is obtained as follows. We run the detection algorithm with increasing values of k, starting with $k = 2$. If a C_{2k} is detected we stop and return the detected cycle. In such a case $2k$ is a $(+1)$-approximation for the girth. If there is no C_{2k} then it follows from the analysis of [3] that a C_{2k-1}, if exists, can be detected within the same running time using a slight modification of their algorithm. If a C_{2k-1} is detected we stop and return the detected cycle. In such a case $2k-1$ is the girth. If neither a C_{2k} nor a C_{2k-1} is found we increment k by 1. Since a cycle must be found when $k = \ell$, we obtain a $(+1)$-approximation for the girth in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$.

A disadvantage of the algorithm described above is the $\ell^{O(\ell)}$ factor which in our new algorithm does not exist. To better appreciate our contribution we compare our $2k$-`Hybrid` algorithm (for $C_{\leq 2k}$ detection) and the algorithm of [3] (for C_{2k} detection), that are used for obtaining the approximation.

Both algorithms first sparsify the graph (or find a cycle). The algorithm of [3] first handles high degree vertices, by running an $O(m)$-time C_{2k} detection algorithm from each high degree vertex (vertex of degree $> m^{2/(k+1)}$). If a C_{2k} is found then a C_{2k} is returned, otherwise the high degree vertices (and their adjacent edges) are removed. This is possible within the running time since there cannot be too many high degree vertices. In our algorithm, we handle first vertices with at least $m^{2/(k+1)}$ edges up to distance 2 rather than 1. This is possible using our `NbrBallOrCycle` algorithm, since we can search for a $C_{\leq 2k}$ from a vertex and also its neighbors in $O(m)$ time. If a C_{2k} is found then a C_{2k} is returned, otherwise the vertex and its neighbors are removed (therefore also the edges up to distance 2 from the vertex). Then, if required, we handle also high degree vertices and remove them.

After the sparsification stage, in [3] they prove that when the maximum degree is bounded, if there is no C_{2k} then the number of *capped k-walks* (walks of length k that visit only nodes according to a fixed ordering) in the graph is bounded by $O(k^{O(k)} m^{1+(k-1)/(k+1)})$, where the $k^{O(k)}$ factor follows from their analysis. They use this property to build a series of subgraphs in which the total number of edges is bounded, and therefore it is possible to search for a C_{2k} in these graphs efficiently. In our algorithm, after the sparsification stage it holds that the total number of edges up to distance k from all the vertices is bounded by $O(m^{1+(k-1)/(k+1)})$ (avoiding the $k^{O(k)}$ factor), and we can search efficiently for a $C_{\leq 2k}$.

6 A general hybrid algorithm

Algorithm $2k$-`Hybrid`(G, k), presented in the previous section, either returns a $C_{\leq 2k}$ or determines that $g > 2k$, in $O(m^{1+\frac{k-1}{k+1}})$ time. In this section we introduce an additional parameter $2 \leq \alpha \leq k$ and present a $(2k, 2\alpha)$-hybrid algorithm that either returns a $C_{\leq 2k}$ or determines that $g > 2\alpha$, in $O((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time. In Section 7 we use the $(2k, 2\alpha)$-hybrid algorithm to present two tradeoffs for girth approximation.

To obtain the $(2k, 2\alpha)$-hybrid algorithm we first extend algorithm `NbrBallOrCycle`. Then, we use the extended `NbrBallOrCycle` together with additional tools that we develop to either return a $C_{\leq 2k}$ or sparsify dense regions of the graph, so that we can check whether $g > 2\alpha$ (or return a $C_{\leq 2k}$) in $O(m^{1+\frac{\alpha-1}{k+1}})$ time, by running `AllVtxBallOrCycle`(G, α).

6.1 Extending NbrBallOrCycle

In algorithm NbrBallOrCycle we mark vertices that can be removed from the graph, by using the property that if BallOrCycle(v,k) did not return a $C_{\leq 2k}$ then v is not on a $C_{\leq 2k}$. In [7], they introduce an additional parameter α and used the following extended version of this property: If BallOrCycle(v,k) did not return a $C_{\leq 2k}$ then no vertex of $V_v^{k-\alpha}$ is on a $C_{\leq 2\alpha}$. We use the same approach and modify NbrBallOrCycle to get an additional integer parameter α such that $2 \leq \alpha \leq k$. After each call to BallOrCycle(G', v, k), where $v \in N_s$, if no cycle was found we add $V_v^{k-\alpha}$, instead of v, to $\hat{U}$. The modified pseudo-code appears in the full version of this paper [10]. We rephrase Lemma 12 to suit this modification. The proof is also given in [10].

▶ **Lemma 18.** *Let $2 \leq \alpha \leq k$. If NbrBallOrCycle(G, s, k, α) finds a cycle C then $wt(C) \leq 2k$. Otherwise, no vertex in $V_s^{k-\alpha+1}$ is part of a $C_{\leq 2\alpha}$ in G, and the set $\hat{U} = V_s^{k-\alpha+1}$ is returned.*

For the running time, we note that the sets V_v^k are computed during the execution of BallOrCycle(G', v, k), for every $(s, v) \in N_s$ for which no cycle was found. Their total size is also $O(n+m)$, and we can obtain from them the sets $V_v^{k-\alpha}$ and add these sets to $\hat{U}$ in $O(n+m)$ time. Therefore, the modified NbrBallOrCycle also runs in $O(n+m)$ time.

6.2 A $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm

In this section we present a $(\max\{2k, g\}, 2\alpha)$-hybrid algorithm called ShortCycle, where $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$. ShortCycle (pseudo-code in [10]) gets a graph G and two integers $\alpha, k \geq 2$. If $m \geq 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then we run algorithm ShortCycleDense(G, k) (pseudo-code in [10]), which is based on algorithm DegenerateOrCycle of [7].

If $m < 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then the main challenge is when $\alpha \leq k < \frac{n}{2}$. In this case we run algorithm ShortCycleSparse(G, k, α) (described later). The cases that $k \geq \frac{n}{2}$ or $k < \alpha$ are relatively simple and are treated by algorithm SpecialCases(G, k, α) (see the full version of this paper [10]). We summarize the properties of ShortCycle(G, k, α) in the next theorem.

▶ **Theorem 19.** *Let $\alpha, k \geq 2$ be integers. ShortCycle(G, k, α) runs whp in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time and either returns a $C_{\leq \max\{2k,g\}}$, or determines that $g > 2\alpha$.*

The next corollary follows from Theorem 19, when $\alpha \leq k$.

▶ **Corollary 20.** *Let $2 \leq \alpha \leq k$. Algorithm ShortCycle(G, k, α) runs whp in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time and either returns a $C_{\leq 2k}$, or determines that $g > 2\alpha$.*

In the rest of this section, we present the proof of Theorem 19. As follows from [7], if $m \geq 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then ShortCycleDense(G, k) returns in $O(\min\{m, n^{1+1/k}\})$ time a $C_{\leq 2k}$. We now consider the case in which $m < 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$. We prove in the full version of this paper [10] that SpecialCases(G, k, α) satisfies the claim of Theorem 19, when $k \geq \frac{n}{2}$ or $k < \alpha$.

Our main technical contribution is algorithm ShortCycleSparse that handles the case of $\alpha \leq k < \frac{n}{2}$. Notice that if $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$ for every $u \in V$, then AllVtxBallOrCycle(G, α) is a $(2k, 2\alpha)$-hybrid algorithm that either finds a $C_{\leq 2\alpha}$ (which is also a $C_{\leq 2k}$ as $\alpha \leq k$) or determines that $g > 2\alpha$, in $O(m^{1+\frac{\alpha-1}{k+1}})$ time. Thus, in ShortCycleSparse we ensure that if we call AllVtxBallOrCycle, the property that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$, for every $u \in V$ (whp), holds. To do so, we run BfsSample and possibly HandleRemainder. If no cycle was returned, the property holds, and we can safely run AllVtxBallOrCycle.

`ShortCycleSparse` (pseudo-code in [10]) gets a graph G and two integers $\alpha, k \geq 2$ such that $\alpha \leq k < \frac{n}{2}$, and is composed of three stages. In the first stage we call `BfsSample`(G, k, α) (described later). If `BfsSample` returns a cycle C then `ShortCycleSparse` stops and returns C, otherwise we proceed to the second stage. In the second stage, if $(k+1) \bmod (\alpha - 1) \neq 0$, we call `HandleRemainder`$(G, k, \alpha)$ (also described later). If `HandleRemainder` returns a cycle C then `ShortCycleSparse` stops and returns C, otherwise we proceed to the last stage. In the last stage, we call `AllVtxBallOrCycle`(G, α). If `AllVtxBallOrCycle` returns a cycle C then $2k$`-Hybrid` stops and returns C, otherwise `ShortCycleSparse` returns `null`.

Next, we give a high level description of `BfsSample`. The goal of `BfsSample` is to either sparsify the graph without removing any $C_{\leq 2\alpha}$, or to report a $C_{\leq 2k}$. For simplicity assume that $(k+1) \bmod (\alpha - 1) = 0$. In such a case, if `BfsSample` does not report a $C_{\leq 2k}$, then the graph after `BfsSample` ends contains every $C_{\leq 2\alpha}$ that was in the original graph, and satisfies, whp, the following sparsity property: For every $u \in V$ it holds that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$.

This implies that in `BfsSample` we need to find every $u \in V$ which is in a dense region with $|E_u^{\alpha-1}| \geq m^{\frac{\alpha-1}{k+1}}$, and to check if u is in a $C_{\leq 2\alpha}$, so that if not we can remove u. Finding every such u is possible within the time limit by running `IsDense`$(G, u, m^{\frac{\alpha-1}{k+1}}, \alpha - 1)$ for every $u \in V$. The problem is that checking whether u is on a $C_{\leq 2\alpha}$ for every such u is too costly since there might be n such vertices, and this check costs $O(n)$ using `BallOrCycle`(G, u, α).

One way to overcome this problem is to sample an edge set S of size $\tilde{\Theta}(m^{1-\frac{\alpha-1}{k+1}})$ that hits the $m^{\frac{\alpha-1}{k+1}}$ closest edges of each vertex, and then use S to detect the vertices in the dense regions that are not on a $C_{\leq 2\alpha}$. In `BfsSample` we use a *detection process* in which we call `BallOrCycle` or `NbrBallOrCycle` from the endpoints of S's edges, and then, if no cycle was found, we use the information obtained from this call to identify vertices that are not on a $C_{\leq 2\alpha}$. The detection process either detects vertices that are not on a $C_{\leq 2\alpha}$ and can be removed, or reports a $C_{\leq 2k}$. However, it is not clear how to implement this detection process efficiently, since just running `BallOrCycle` from the endpoints of S's edges takes $O(nm^{1-\frac{\alpha-1}{k+1}})$ time which might be too much. Our solution is an *iterative* sampling procedure that starts with a smaller hitting set of edges, of size $\tilde{\Theta}(m^{\frac{\alpha-1}{k+1}})$. For such a hitting set we can run our detection process. If a $C_{\leq 2k}$ was not reported, then we remove the appropriate vertices and sparsify the graph without removing any $C_{\leq 2\alpha}$. When the graph is sparser, the running time of our detection process becomes faster. Thus, in the following iteration we can sample a larger hitting set for which we run this process, and either return a $C_{\leq 2k}$ or sparsify the graph further for the next iteration. We continue the iterative sampling procedure until we get to the required sparsity property in which $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$ for every $u \in V$ (whp).

We remark that in the first iteration of `BfsSample`, the detection process calls `NbrBallOrCycle`, while in the rest of the iterations `BallOrCycle` is called. The use of `NbrBallOrCycle` allows us, in the case that no $C_{\leq 2k}$ is reported, to bound $|E_v^d|$ with $m^{\frac{d}{k+1}}$ for every $v \in V$, rather than $|E_v^{d-1}|$. This is used to achieve the required sparsity property. Since `NbrBallOrCycle` runs in $O(m)$ time we can only use it in the first iteration when the sampled set is small enough. In the rest of the iterations we use `BallOrCycle` instead.

We now formally describe `BfsSample`. `BfsSample` (see Algorithm 3) gets a graph G and two integers $\alpha, k \geq 2$ such that $\alpha \leq k < \frac{n}{2}$. We first set y to $\lceil \frac{k+1}{\alpha-1} \rceil - 1$. Then, we start the main for loop that has at most y iterations. In the ith iteration, we initialize $\hat{V}$ to $\emptyset$ and sample a set $S_i \subseteq E$ of size $\tilde{\Theta}(m^{\frac{i\cdot(\alpha-1)}{k+1}})$. Next, we scan the endpoints in $V(S_i)$ using an inner for-each loop.

If $i = 1$, we call `NbrBallOrCycle`(G, s, k, α) from every endpoint $s \in V(S_1)$. `NbrBallOrCycle` returns either a cycle or a set of vertices $\hat{U}$. If `NbrBallOrCycle` returns a cycle then the cycle is returned by `BfsSample`. Otherwise, we add $\hat{U}$ to $\hat{V}$.

Algorithm 3 `BfsSample`(G, k, α).

```
1  y ← ⌈(k+1)/(α−1)⌉ − 1;
2  for i ← 1 to y do
3      sample a set S_i of Θ̃(m^{i·(α−1)/(k+1)}) edges;
4      V̂ ← ∅;
5      foreach s ∈ V(S_i) do
6          if i = 1 then
7              (Û, C) ← NbrBallOrCycle(G, s, k, α);
8              if C ≠ null then return C;
9              V̂ ← V̂ ∪ Û;
10         else
11             k_i ← (k + 1) − (i − 1) · (α − 1);
12             (V_s^{k_i}, C) ← BallOrCycle(s, k_i);
13             if C ≠ null then return C;
14             V̂ ← V̂ ∪ V_s^{k_i−α};
15     G ← G \ V̂;
16 return null;
```

If $i > 1$ then we call `BallOrCycle`(s, k_i), where $k_i = (k+1) - (i-1)\cdot(\alpha-1)$, from every endpoint $s \in V(S_i)$. If a cycle is found by `BallOrCycle` then the cycle is returned by `BfsSample`. If `BallOrCycle` does not return a cycle then we add $V_{u_j}^{k_i-\alpha}$ to $\hat{V}$.

Right after the inner for-each loop ends, we remove $\hat{V}$ from G, and continue to the next iteration of the main for loop. If no cycle was found after y iterations, we return `null`. Let $\ell \leq y$ be the last iteration in which vertices were removed. Let $\hat{V}_i$ be the set of vertices that were removed during the ith iteration, where $\hat{V}_i = \emptyset$ for $i > \ell$. Let G_0 (G_1) be G before (after) running `BfsSample`. The relevant figures in the full version of this paper [10] illustrate the key steps of the first and the following iterations of `BfsSample`. We summarize the properties of `BfsSample` in the next lemma. The proof is also given in the full version [10].

▶ **Lemma 21.** *`BfsSample`(G, k, α) satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2k$*
- **(ii)** *If a cycle is not returned then $|E_u^{(k+1)-y\cdot(\alpha-1)}| < m^{1-\frac{y\cdot(\alpha-1)}{k+1}}$, for every $u \in G_1$, whp*
- **(iii)** *If $u \in G_0 \setminus G_1$ then u is not part of a $C_{\leq 2\alpha}$ in G_0*
- **(iv)** *`BfsSample`(G, k, α) runs in $\tilde{O}(\lfloor \frac{k+1}{\alpha-1} \rfloor \cdot m^{1+\frac{\alpha-1}{k+1}})$ time, whp.*

Recall that our goal is to obtain the sparsity property that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$, for every $u \in V$, so that we can run `AllVtxBallOrCycle`(G, α). However, after running `BfsSample` the required sparsity property is guaranteed to hold (whp) only if $(k+1) \bmod (\alpha-1) = 0$. In the case that $(k+1) \bmod (\alpha-1) \neq 0$ we need an additional step which is implemented in `HandleRemainder`, to guarantee that the required sparsity property holds.

Next, we formally describe `HandleRemainder`. `HandleRemainder` (see Algorithm 4) gets a graph G and two integers $\alpha, k \geq 2$ such that $k+1 = q(\alpha-1)+r$ where $q \geq 1$ and $r > 0$. We set D to $m^{\frac{1}{k+1}}$ and r to $(k+1) \bmod (\alpha-1)$. Then, a while loop runs as long as $(\alpha-1) \bmod r \neq 0$. Let r_i be the value of r when the ith iteration begins, so that r_1 is $(k+1) \bmod (\alpha-1)$. Let ℓ be the total number of iterations and $r_{\ell+1}$ the value of r after the ℓth iteration. During the

Algorithm 4 `HandleRemainder`(G, k, α).

$D \leftarrow m^{\frac{1}{k+1}}$;
$r \leftarrow (k+1) \bmod (\alpha - 1)$;
while $(\alpha - 1) \bmod r \neq 0$ **do**
  $r \leftarrow (\lceil \frac{\alpha-1}{r} \rceil \cdot r) - (\alpha - 1)$;
  $C \leftarrow$ `SparseOrCycle`(G, D, r, α);
  if $C \neq$ *null* **then return** C;
return `null`;

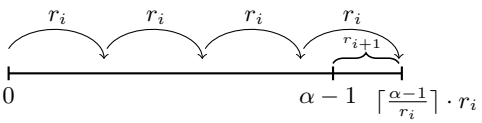


Figure 3 The relation between r_i, r_{i+1} and $\alpha - 1$.

ith iteration, we set r_{i+1} to $(\lceil \frac{\alpha-1}{r_i} \rceil \cdot r_i) - (\alpha - 1)$, where $\lceil \frac{\alpha-1}{r_i} \rceil \cdot r_i$ is the smallest multiple of r_i that is at least $\alpha - 1$ (see Figure 3). Then, we call `SparseOrCycle`(G, D, r_{i+1}, α). If `SparseOrCycle` returns a cycle C then `HandleRemainder` returns C. If `SparseOrCycle` does not return a cycle then it might be that some vertices were removed from G, and we continue to the next iteration. If the while loop ends without returning a cycle then we return `null`. Let G_0 (G_1) be G before (after) running `HandleRemainder`. Next, we prove two properties on the value of r during the run of `HandleRemainder` (see the full version of this paper [10] for the proof).

▷ Claim 22. Let $k+1 = q(\alpha-1)+r_1$ and assume $r_1 > 0$. **(i)** $0 < r_{\ell+1} < r_\ell < \cdots < r_1 < \alpha-1$. **(ii)** $(r_{i+1} + \alpha - 1) \bmod r_i = 0$, for every $1 \leq i \leq \ell$.

We also prove in the full version [10] the main lemma regarding `HandleRemainder`.

▶ **Lemma 23.** *Let $k+1 = q(\alpha-1)+r_1$ and assume that $r_1 > 0$, $q \geq 1$, and that $|E_u^{r_1}| < D^{r_1}$, for every $u \in V$. `HandleRemainder`(G, k, α) satisfies the following:*

(i) *If a cycle C is returned then $wt(C) \leq 2k$*
(ii) *If a cycle is not returned then $|E_u^{\alpha-1}| < D^{\alpha-1}$, for every vertex $u \in G_2$*
(iii) *If $u \in G_1 \setminus G_2$ then u is not on a $C_{\leq 2\alpha}$ in G_1*
(iv) *`HandleRemainder`(G, k, α) runs in $O(r_1 \cdot m^{1+\frac{\alpha-1}{k+1}})$ time.*

Now we are ready to prove the correctness and running time of `ShortCycleSparse`. The proof is in the full version [10].

▶ **Lemma 24.** *Let $2 \leq \alpha \leq k < \frac{n}{2}$ such that $k+1 = q(\alpha-1)+r$, where $q \geq 1$ and $0 \leq r < \alpha-1$ are integers. Algorithm `ShortCycleSparse`(G, k, α) runs whp in $\tilde{O}((q+r) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time and either returns a $C_{\leq 2k}$, or determines that $g > 2\alpha$.*

Since `ShortCycleSparse` is run by `ShortCycle` when $m \leq O(n^{1+\frac{1}{k}})$ and when $\alpha \leq k$ and so $1 + \frac{\alpha-1}{k+1} \leq 2$, we have $m^{1+\frac{\alpha-1}{k+1}} \leq O(n^{1+\frac{\alpha}{k}})$. Thus, the running time of `ShortCycleSparse` is whp $\tilde{O}((q+r) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$, which is at most $\tilde{O}((\frac{k+1}{\alpha-1}+\alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$. Together with the relatively simple special cases handled in the full version of this paper [10], this completes the proof of Theorem 19.

7 Approximation of the girth

In this section we present two new tradeoffs for girth approximation that follow from Corollary 20. In these tradeoffs we use `ShortCycle` with $2 \leq \alpha \leq k$, so by Corollary 20, `ShortCycle` is an $\tilde{O}((\frac{k+1}{\alpha-1}+\alpha)\cdot\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$-time, $(2k, 2\alpha)$-hybrid algorithm.

7.1 Dense graphs

Kadria et al. [7] presented an $O((\alpha-c)\cdot n^{1+\frac{\alpha}{2\alpha-c}})$-time, $(4\alpha-2c, 2\alpha)$-hybrid algorithm, where $0 < c \leq \alpha$ are two integers (see footnote 3 in Section 2). This algorithm, combined with a binary search, was used by [7] to compute for every $\varepsilon \in (0,1]$ a cycle C such that $wt(C) \leq 4\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (2-\varepsilon)g+4$, in $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time, if $g \leq \log^2 n$. We use `ShortCycle` in a similar way and prove:

▶ **Theorem 25.** *Let $\ell \geq 2$ be an integer, $\varepsilon \in [0,1]$ and $g \leq \log^2 n$. It is possible to compute, whp, in $\widetilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$ time, a cycle C such that $wt(C) \leq 2\ell\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (\ell-\varepsilon)g+\ell+2$.*[9]

The proof is in the full version of this paper [10]. Our algorithm can be viewed as a natural generalization of a couple of algorithms from [7]. By setting $\ell = 2$ in Theorem 25 we get the $(2-\varepsilon)g+4$ approximation of [7]. By setting $\varepsilon = 0$ we get an $\widetilde{O}(\ell \cdot n^{1+1/\ell})$ time algorithm that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$, which is similar to the $\widetilde{O}(n^{1+1/\ell})$ time algorithm of [7] that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$.

7.2 Sparse graphs

We use a similar approach to obtain a tradeoff for girth approximation in sparse graphs. We prove the following theorem in the full version of this paper [10].

▶ **Theorem 26.** *Let $\ell \geq 3$ be an integer, $\varepsilon \in [0,1)$ and $g \leq \log^2 n$. It is possible to compute, whp, in $\widetilde{O}(\ell \cdot m^{1+1/(\ell-\varepsilon)})$ time, a cycle C such that $wt(C) \leq 2\ell(\lceil\frac{g}{2}\rceil-1) - 2\lfloor\varepsilon(\lceil\frac{g}{2}\rceil-1)\rfloor - 2 \leq (\ell-\varepsilon)g - \ell + 2\varepsilon$.*

By setting $\varepsilon = 0$ we get an $\widetilde{O}(\ell \cdot m^{1+1/\ell})$-time algorithm that computes a $C_{\leq 2\ell(\lceil\frac{g}{2}\rceil-1)-2}$, as opposed to the $\widetilde{O}(\ell \cdot n^{1+1/\ell})$ time algorithm that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1010–1023. ACM, 2020. doi:10.1145/3357713.3384330.

3 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 112–120. ACM, 2017. doi:10.1145/3055399.3055459.

[9] We note that when $\ell > \log n$, we can run the $\tilde{O}(n^{1+1/\ell'})$-time algorithm of [7], which computes a $C_{\leq 2\ell'\lceil g/2\rceil}$, with $\ell' = \ell - 1$, to get the required approximation. Since $\ell > \log n$, its running time is $\tilde{O}(n)$. Thus, our running time becomes $\tilde{O}(n^{1+1/(\ell-\varepsilon)})$, where the ℓ factor is absorbed into the $\tilde{O}$ notation.

4 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. New subquadratic approximation algorithms for the girth. *arXiv preprint arXiv:1704.02178*, 2017. arXiv:1704.02178.

5 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997. doi:10.1006/INCO.1997.2620.

6 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.

7 Avi Kadria, Liam Roditty, Aaron Sidford, Virginia Vassilevska Williams, and Uri Zwick. Algorithmic trade-offs for girth approximation in undirected graphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1471–1492. SIAM, 2022. doi:10.1137/1.9781611977073.62.

8 Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Information Processing Letters*, 109(10):493–498, 2009. doi:10.1016/J.IPL.2009.01.008.

9 Liam Roditty and Roei Tov. Approximating the girth. *ACM Transactions on Algorithms (TALG)*, 9(2):1–13, 2013. doi:10.1145/2438645.2438647.

10 Liam Roditty and Plia Trabelsi. New algorithms for girth and cycle detection. *arXiv preprint arXiv:2507.02061*, 2025. doi:10.48550/arXiv.2507.02061.

11 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013. doi:10.1145/2488608.2488673.

12 Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 833–845. SIAM, 2012. doi:10.1137/1.9781611973099.67.

13 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995. doi:10.1006/JCSS.1995.1078.

14 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 605–614. IEEE, 1999. doi:10.1109/SFFCS.1999.814635.

15 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

16 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

Faster Approximate Linear Matroid Intersection

Tatsuya Terao
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

We consider a fast approximation algorithm for the linear matroid intersection problem. In this problem, we are given two $r \times n$ matrices M_1 and M_2, and the objective is to find a largest set of columns that are linearly independent in both M_1 and M_2. We design a $(1-\varepsilon)$-approximation algorithm with time complexity $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$, where $\mathrm{nnz}(M_i)$ denotes the number of nonzero entries in M_i for $i = 1, 2$, r_* denotes the maximum size of a common independent set, and $\omega < 2.372$ denotes the matrix multiplication exponent. Our approximation algorithm is faster than the exact algorithm by Harvey [FOCS'06 & SICOMP'09] and Cheung–Kwok–Lau [STOC'12 & JACM'13], which runs in $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$ time.

We also develop a fast $(1-\varepsilon)$-approximation algorithm for the weighted version of the linear matroid intersection problem. In fact, we design a $(1-\varepsilon)$-approximation algorithm for weighted linear matroid intersection with time complexity $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$. Our algorithm improves upon the $(1-\varepsilon)$-approximation algorithm by Huang–Kakimura–Kamiyama [SODA'16 & Math. Program.'19], which runs in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$ time.

To obtain these results, we combine Quanrud's adaptive sparsification framework [ICALP'24] with a simple yet effective method for efficiently checking whether a given vector lies in the linear span of a subset of vectors, which is of independent interest.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques; Mathematics of computing → Matroids and greedoids

Keywords and phrases Linear matroid intersection, fast approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.39

Related Version *Full Version*: `https://arxiv.org/abs/2604.11725` [54]

Funding This work was partially supported by the joint project of Kyoto University and Toyota Motor Corporation, titled "Advanced Mathematical Science for Mobility Society" and by JSPS KAKENHI Grant Number JP24KJ1494.

Acknowledgements We thank Yusuke Kobayashi for his insightful comments, generous support, and valuable feedback on the manuscript. In particular, his input greatly helped us establish Lemma 11. We also thank Kou Hamada for helpful conversations that contributed to the initial motivation for this work. We are grateful to Takashi Noguchi for his helpful comments, which simplified the proof of Lemma 11. Finally, we thank the anonymous reviewers for their careful reading and valuable suggestions.

1 Introduction

Fast Algorithms for Linear Matroid Problems. Matroids are combinatorial structures that abstract linear independence and provide a unifying framework capturing fundamental properties of matrices and graphs. We focus on fast algorithms for problems on linear matroids that are given explicitly as the columns of an $r \times n$ matrix M over a field $\mathbb{F}$.

Motivated by the view that running time should reflect not only the matrix dimensions $r \times n$ but also its sparsity, we design faster algorithms whose complexity depends on $\mathrm{nnz}(M)$ (the number of nonzero entries of M) in addition to r and n. In fact, many commonly studied matroids admit sparse linear representations. For example, it is well known that a graphic

© Tatsuya Terao;
licensed under Creative Commons License CC-BY 4.0

matroid can be represented by a vertex-edge incidence matrix with exactly two nonzero entries per edge, resulting in a total of $2m$ nonzero entries, where m is the number of edges (see e.g., [47, Section 5.1]).

From this perspective, an important contribution is due to Cheung–Kwok–Lau [14], who developed a remarkable sketching technique and, as a result, presented a surprising algorithm for computing the rank of a matrix M in $\tilde{O}(\mathrm{nnz}(M) + (\mathrm{rank}(M))^\omega)$ time.[1] Note that even checking whether a given subset of vectors is independent already requires $O((\mathrm{rank}(M))^\omega)$ time, and reading the input takes $O(\mathrm{nnz}(M))$ time.

Nguyễn [44] presented an elegant algorithm for computing a maximum-weight base of a linear matroid represented by a matrix M. While a direct application of the sketching technique of Cheung–Kwok–Lau would only yield a running time of $\tilde{O}\big(\mathrm{nnz}(M) + n \cdot (\mathrm{rank}(M))^{\omega-1}\big)$, his refined use of this technique leads to $\big(\mathrm{nnz}(M) + (\mathrm{rank}(M))^\omega\big)^{1+o(1)}$. This problem has also been studied, under the name column rank profile problem [19, 20, 36, 52], which asks for the lexicographically smallest sequence of $\mathrm{rank}(M)$ column indices such that the corresponding columns of the matrix are linearly independent. Dumas–Pernet–Sultan [20] achieved the same time complexity for the column rank profile problem.

Linear Matroid Intersection. In this work, we consider fast approximation algorithms for linear matroid intersection. This linear matroid intersection problem can be described without relying on matroid terminology as follows: Given two $r \times n$ matrices M_1 and M_2 over a field $\mathbb{F}$, where the columns in M_1 and M_2 are indexed by $\{1, \ldots, n\}$, the objective is to find a set $I \subseteq \{1, \ldots, n\}$ of maximum size such that the columns indexed by I are linearly independent in both M_1 and M_2. Many important problems in combinatorial optimization, such as bipartite matching, rainbow spanning tree, arborescence, and spanning tree packing, can be viewed as special cases of linear matroid intersection.

For the linear matroid intersection problem, Cunningham [16] gave a combinatorial algorithm with time complexity $\tilde{O}(nr^2)$. Gabow–Xu [28, 29] proposed a faster combinatorial algorithm with time complexity $\tilde{O}(nrr_*^{1/(4-\omega)}) = O(nrr_*^{0.62})$, where ω is the exponent of matrix multiplication, known to satisfy $\omega < 2.371339$ [3]. Here, r_* denotes the maximum size of a common independent set. Harvey [33] developed an algebraic algorithm with time complexity $O(nr^{\omega-1})$. Cheung–Kwok–Lau [14] combined their elegant sketching technique with Harvey's algorithm [33], obtaining an algorithm with running time $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$. In fact, simply applying their sketching technique reduces an $r \times n$ input matrix to an $O(r_*) \times n$ matrix while preserving linear dependence among the columns.

Given the progress on the case of a single matroid [14, 20, 44], it is natural to ask whether linear matroid intersection can be solved in $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ time. Note that even checking whether a given subset is independent in a single matroid already requires $O(r_*^\omega)$ time, and reading the input takes $O(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2))$ time. Avoiding the dependence on n is most meaningful when $r_* \ll n$, a regime that arises naturally, for example, for graphic matroids. However, achieving a running time of $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ seems quite challenging. To achieve such a running time, one must avoid explicitly computing the product of an $O(r_*) \times n$ matrix and an $n \times O(r_*)$ matrix, which would already take $O(nr_*^{\omega-1})$ time. Here, it is well known that M_1 and M_2 have a common base if and only if $\det(M_1 D M_2^\top) \neq 0$, where D is a diagonal matrix with variables on the diagonal $D_{i,i} = x_i$ (see for e.g., [31, Lemma 2.5]). Using this matrix formulation, a standard approach to the

[1] The $\tilde{O}$ notation omits polylog factors.

decision version of linear matroid intersection, i.e., deciding whether a common base exists, requires computing the product of an $O(r_*) \times n$ matrix and an $n \times O(r_*)$ matrix. Hence, to our knowledge, no prior result has shown that the time complexity for merely estimating the size of a maximum common independent set can be faster than that of Cheung–Kwok–Lau.

Our Result for Maximum Cardinality Linear Matroid Intersection. Our main result is to develop a $(1-\varepsilon)$-approximation algorithm for linear matroid intersection running in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ time.[2] We note that our algorithm is applicable over any field.

▶ **Theorem 1.** *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximate solution to the maximum cardinality linear matroid intersection problem can be computed with high probability*[3] *in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ time.*

Our approximation algorithm in Theorem 1 is faster than the exact algorithm obtained by combining Harvey's algorithm [33] with the sketching technique of Cheung–Kwok–Lau [14], which runs in $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$ time. Even when M_1 and M_2 are dense and $r_* = \Theta(r)$, our approximation algorithm achieves a running time of $\tilde{O}_\varepsilon(nr + r^\omega)$, which is faster than the $O(nr^{\omega-1})$ time required by Harvey's exact algorithm [33]. Surprisingly, our complexity is almost the same as that of computing the rank of a matrix [14], and also that of finding a maximum-weight base of a single linear matroid [20, 44].

Weighted Linear Matroid Intersection. We also consider the weighted version of the linear matroid intersection problem. In this problem, we are given two $r \times n$ matrices M_1 and M_2 over a field $\mathbb{F}$, whose columns are indexed by $\{1, \dots, n\}$, and a weight function $c : \{1, \dots, n\} \to \mathbb{R}_{\geq 0}$. The objective is to find a set $I \subseteq \{1, \dots, n\}$ such that the columns indexed by I are linearly independent in both M_1 and M_2, and the total weight $\sum_{e \in I} c(e)$ is maximized. Gabow–Xu [28, 29] gave a combinatorial algorithm with time complexity $\tilde{O}(nr^{\frac{7-\omega}{5-\omega}} \log(nC)) = \tilde{O}(nr^{1.77} \log(nC))$, where C is the largest given weight. Harvey [32] proposed an algebraic algorithm with time complexity $\tilde{O}(C^{1+o(1)} nr^{\omega-1})$. Huang–Kakimura–Kamiyama [35] presented a $(1-\varepsilon)$-approximation algorithm with time complexity $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$, where r_* is the maximum size of a common independent set.

Our Result for Maximum Weight Linear Matroid Intersection. Using a similar idea to that used to obtain Theorem 1, we also develop a $(1-\varepsilon)$-approximation algorithm for weighted linear matroid intersection running in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ time. We note that our algorithm is applicable over any field.

▶ **Theorem 2.** *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximate solution to the weighted linear matroid intersection problem can be computed with high probability in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$ time.*

Our algorithm in Theorem 2 improves upon the $(1-\varepsilon)$-approximation weighted linear matroid intersection algorithm by Huang–Kakimura–Kamiyama [35], which runs in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + nr_*^{\omega-1})$ time.

[2] The $\tilde{O}_\varepsilon$ notation omits polylog factors and polynomial factors in ε^{-1}.
[3] We say "with high probability" if the failure probability is at most $n^{-\Theta(1)}$.

1.1 Technical Overview

Adaptive Sparsification Framework of Quanrud [48]. Our linear matroid intersection algorithm builds upon the adaptive sparsification framework of Quanrud [48], which was developed to obtain a fast $(1-\varepsilon)$-approximation algorithm for matroid intersection in the oracle model. However, we emphasize that a direct application of Quanrud's framework does not yield the running time achieved by our algorithm.

In Quanrud's framework, the original problem over n elements is reduced to $O(\log(n)/\varepsilon)$ primal-dual instances of matroid intersection, each over $\tilde{O}_\varepsilon(r_*)$ elements, where r_* is the maximum size of a common independent set. For linear matroid intersection, computing primal and dual solutions to sparsified instances can be done straightforwardly using Harvey's exact algorithm for linear matroid intersection [33]. Applied to the sparsified instance of size $\tilde{O}_\varepsilon(r_*)$, Harvey's algorithm runs in $\tilde{O}_\varepsilon(r_*^\omega)$ time, which is sufficiently fast. To adaptively construct sparsified instances, it is necessary to compute $\mathrm{span}_{\mathcal{M}_1}(\tilde{S})$ and $\mathrm{span}_{\mathcal{M}_2}(\tilde{T})$, where $(\tilde{S},\tilde{T})$ denotes a dual solution to the sparsified instance. For linear matroids, a straightforward method can compute the spans in $O(nr_*^{\omega-1})$ time, but this is still too slow for our purpose.

Challenge in Computing a Span. Without using matroid terminology, our task is as follows: Given an $r \times n$ matrix M and a subset $S \subseteq [n]$, determine whether each column vector $\boldsymbol{m}_i$ lies in the linear span of the set $\{\boldsymbol{m}_j \mid j \in S\}$, where $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n$ denote the vectors corresponding to the columns of M. We present a simple randomized algorithm that solves this task in $\tilde{O}(\mathrm{nnz}(M) + r^\omega)$ time, which is of independent interest.

In the oracle model, where an algorithm accesses a matroid $\mathcal{M}$ through an independence oracle, the span

$$\mathrm{span}_{\mathcal{M}}(S) = \{v \in V \mid \mathrm{rank}_{\mathcal{M}}(S \cup \{v\}) = \mathrm{rank}_{\mathcal{M}}(S)\}$$

of a set S can be computed naively using a linear number of queries. However, when the matroid is given via a matrix representation, the standard approaches are too inefficient for our purpose. Naively, each independence query takes $O(r^\omega)$ time, and thus the most straightforward implementation requires $O(nr^\omega)$ time. The standard Gaussian elimination approach takes $O(nr^2)$ time to compute the span. Using the fact that the linear representation of $\mathcal{M}/S$, the matroid obtained by contracting $\mathcal{M}$ by S, can be computed in $O(nr^{\omega-1})$ time (see [33, Fact 4.1]), we can reduce the time complexity to $O(nr^{\omega-1})$. However, this complexity is still inefficient for obtaining a linear matroid intersection algorithm that outperforms the best known exact algorithm. To compute the span faster than $O(nr^{\omega-1})$ time, one must avoid computing the product of an $r \times n$ matrix and an $n \times r$ matrix, which is a challenging task.

To overcome this difficulty, our fast span computation algorithm shares a similar spirit with the famous Freivalds' algorithm [24] (see e.g., [42, Section 7.1]) for verifying matrix multiplication without actually performing the multiplication. In our approach, we sample a vector $\boldsymbol{v}$ from the orthogonal complement of $\mathrm{span}\{\boldsymbol{m}_j \mid j \in S\}$, which can be computed in $\tilde{O}(\mathrm{nnz}(M) + r^\omega)$ time. Then, for each column vector $\boldsymbol{m}_i$, we check whether $\boldsymbol{v}^\top \boldsymbol{m}_i = 0$. It is obvious that if $\boldsymbol{m}_i \in \mathrm{span}\{\boldsymbol{m}_j \mid j \in S\}$, then $\boldsymbol{v}^\top \boldsymbol{m}_i = 0$ always holds. Furthermore, we can show that if $\boldsymbol{m}_i \notin \mathrm{span}\{\boldsymbol{m}_j \mid j \in S\}$, then $\boldsymbol{v}^\top \boldsymbol{m}_i \neq 0$ with high probability. The total time required to compute all inner products $\boldsymbol{v}^\top \boldsymbol{m}_i$ for all $i \in [n]$ is $O(\mathrm{nnz}(M))$. Therefore, the total running time of our span computation algorithm is $\tilde{O}(\mathrm{nnz}(M) + r^\omega)$, which is efficient enough to enable the design of a linear matroid intersection algorithm that outperforms the best known exact algorithm.

Challenge for Weighted Linear Matroid Intersection. For weighted linear matroid intersection, a naive combination of the adaptive sparsification framework of Quanrud [48] and our fast span computation algorithm yields a $(1-\varepsilon)$-approximation algorithm with time complexity $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1)+\mathrm{nnz}(M_2)+r_*^{\omega+1})$. This additional factor arises because, in the weighted setting, we need to test membership in $\mathrm{span}_{\mathcal{M}}(S)$ not for a single candidate set S, but for $O(r_*)$ different candidates. To improve the running time to $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1)+\mathrm{nnz}(M_2)+r_*^{\omega})$, we carefully employ the fast Gram–Schmidt orthonormalization algorithm of van den Brand [57]. Concretely, we use van den Brand's algorithm to compute, for each set in the support of the dual solution in the weighted setting, a basis for the corresponding subspace as well as a basis for its orthogonal complement.

When considering matrices over the real field $\mathbb{R}$, the idea of employing the fast Gram–Schmidt orthonormalization algorithm of van den Brand works correctly in conjunction with the techniques described above. However, over an arbitrary field $\mathbb{F}$, the argument may fail. This is because, when considering matrices over a field other than $\mathbb{R}$, it may happen that a linear subspace W satisfies $W\cap W^\perp\neq\{\mathbf{0}\}$ and $W+W^\perp\neq\mathbb{F}^r$,[4] where $W^\perp$ is the orthogonal complement of W (i.e., $W^\perp=\{\boldsymbol{v}\in\mathbb{F}^r\mid\forall\boldsymbol{w}\in W,\boldsymbol{v}^\top\boldsymbol{w}=0\}$). This issue matters in our setting because we need to maintain bases for subspaces corresponding to many different sets, and for this purpose, our algorithm relies on the property that $W\cap W^\perp=\{\mathbf{0}\}$ and $W\oplus W^\perp=\mathbb{F}^r$.

To design an algorithm applicable to any field, we replace the Euclidean inner product $\boldsymbol{v}^\top\boldsymbol{w}$ with the bilinear form $\langle\boldsymbol{v},\boldsymbol{w}\rangle=\boldsymbol{v}^\top B\boldsymbol{w}$, where we choose $w_1,\dots,w_r$ independently and uniformly at random from $\mathbb{F}$, and define a diagonal $r\times r$ matrix B such that $B_{i,i}=w_i$. Here, we define the orthogonal complement with respect to the bilinear form $\langle\boldsymbol{v},\boldsymbol{w}\rangle=\boldsymbol{v}^\top B\boldsymbol{w}$ as $W_B^\perp=\{\boldsymbol{v}\in\mathbb{F}^r\mid\forall\boldsymbol{w}\in W,\langle\boldsymbol{v},\boldsymbol{w}\rangle=\boldsymbol{v}^\top B\boldsymbol{w}=0\}$. For a fixed linear subspace W, we can show that both $W\cap W_B^\perp=\{\mathbf{0}\}$ and $W\oplus W_B^\perp=\mathbb{F}^r$ hold with high probability.

To determine whether $\boldsymbol{m}_i\in\mathrm{span}\{\boldsymbol{m}_j\mid j\in S\}$, it suffices to check whether $\langle\boldsymbol{v},\boldsymbol{m}_i\rangle=\boldsymbol{v}^\top B\boldsymbol{m}_i=0$. Here, $\boldsymbol{v}$ is a vector sampled from the orthogonal complement of $\mathrm{span}\{\boldsymbol{m}_j\mid j\in S\}$ with respect to the bilinear form $\langle\boldsymbol{v},\boldsymbol{w}\rangle=\boldsymbol{v}^\top B\boldsymbol{w}$. We make use of a fast orthogonalization algorithm with respect to the bilinear form $\langle\boldsymbol{v},\boldsymbol{w}\rangle=\boldsymbol{v}^\top B\boldsymbol{w}$, which is a modified version of the fast Gram–Schmidt orthonormalization algorithm of van den Brand [57].

1.2 Related Work

Faster Matroid Intersection under Oracle Models. In addition to the linear matroid model, another standard model for handling matroids is the oracle model. In this model, an algorithm accesses a matroid through an oracle that answers queries about its structure. In recent years, the development of faster algorithms for matroid intersection under oracle models has attracted significant attention. Starting the work of Edmonds [21,22], many fast algorithms in the oracle model have been studied [2,6–9,12,13,16,18,25,35,38,39,45,48,51,53,55,56]. There have also been many studies on fast matroid intersection algorithms for several important specific classes of matroids [16, 26–29, 32, 33, 35, 59]. Recently, Blikstad–Mukhopadhyay–Nanongkai–Tu [7] proposed a remarkable new oracle model, called dynamic oracle, which unifies various results on fast matroid intersection algorithms for specific classes of matroids; see [7, Table 2].

[4] For example, consider $W=\mathrm{span}\{(1,1,0,0)^\top,(0,0,1,1)^\top\}$ over the binary field GF(2). Then, $W^\perp=\mathrm{span}\{(1,1,0,0)^\top,(0,0,1,1)^\top\}$. In this case, neither $W\cap W^\perp=\{\mathbf{0}\}$ nor $W+W^\perp=(\mathrm{GF}(2))^4$ holds.

Note that the recent dynamic oracle framework of Blikstad–Mukhopadhyay–Nanongkai–Tu [7] yields a linear matroid intersection algorithm with time complexity $O(n^{2.5925}\sqrt{r_*})$, using the dynamic matrix rank maintenance algorithm of [58]. However, this complexity is slower than that of Harvey [33].

Additional Related Work. There are several studies on fast algebraic algorithms for problems that generalize linear matroid intersection, such as linear matroid parity [15], fractional linear matroid parity [46], and linear delta-matroid parity [37]. Matoya–Oki [41] derandomized Harvey's [33] linear matroid intersection algorithm for a special case known as a Pfaffian pair. Linear matroid intersection has also been well studied in parallel and other computational models [1,30,31,43]. The intersection of multiple linear matroids has also attracted interest in the parameterized algorithm community; see, for example, [17, Section 12.3] and [5,10,23,40].

Concurrent Work. Independent of our work, Dudeja–Grilnberger [18] presented a remarkable reduction in the independence oracle model that converts any $(1-\varepsilon)$-approximation algorithm for unweighted matroid intersection into a $(1-\varepsilon)$-approximation algorithm for weighted matroid intersection, while increasing the running time by only a factor of $\log C$, where C denotes the aspect ratio. Thus, one might hope that, when combined with our unweighted result in Theorem 1, their reduction could yield a weighted result comparable to Theorem 2. However, their reduction relies on several subroutines, including a procedure for greedily computing a common independent set, and it is unclear whether all of these subroutines can be implemented efficiently for linear matroids. Accordingly, it is unclear whether their reduction, combined with Theorem 1, can yield a result comparable to Theorem 2.

2 Preliminaries

2.1 Basic Notation

The set $\{1,\dots,n\}$ is denoted by $[n]$. If U is a set, then $\binom{U}{k}$ denotes the set of all its subsets of size k.

Consider a matrix M over a field $\mathbb{F}$. Let $\mathrm{nnz}(M)$ denote the number of nonzero entries in M. Let $M[S]$ denote the submatrix consisting of columns indexed by S. Let $M[R,C]$ denote the submatrix consisting of the rows indexed by R and the columns indexed by C. Let $[A \mid B]$ denote the matrix whose columns consist of those of A, followed by those of B. We write I to denote the identity matrix. Let $\mathrm{span}\{\boldsymbol{v}_1,\dots,\boldsymbol{v}_k\}$ denote the linear span of the set of vectors $\{\boldsymbol{v}_1,\dots,\boldsymbol{v}_k\}$.

2.2 Assumptions and Conventions

When $\mathbb{F}$ is a finite field, we can assume that $|\mathbb{F}| = \Omega(\mathrm{poly}(n))$ by the following well-known lemma. As a result of this assumption, the running time of our algorithm increases by a multiplicative factor of $\mathrm{polylog}(n)$.

▶ **Lemma 3** (from [14, Lemma 2.1]). *Let A be an $r \times n$ matrix over a field $\mathbb{F}$ with p^c elements. We can construct a finite field $\mathbb{F}'$ with $p^{ck} = \Omega(\mathrm{poly}(n))$ elements and an injective mapping $f\colon \mathbb{F} \to \mathbb{F}'$ so that the image of $\mathbb{F}$ is a subfield of $\mathbb{F}'$. Then, the $r \times n$ matrix A' where $A'_{ij} = f(A_{ij})$ satisfies the following property: a set of columns of A is independent if and only if the corresponding set of columns of A' is independent. This preprocessing step takes $O(\mathrm{nnz}(A))$ time. Each field operation in $\mathbb{F}'$ can be performed using $\mathrm{polylog}(n)$ field operations in $\mathbb{F}$.*

When $\mathbb{F}$ is an infinite field, we assume the exact arithmetic model where each field operation is performed in one unit of time. In the algorithm, we also require the ability to sample a random element from an arbitrary subset of size $\Omega(\mathrm{poly}(n))$ in $\mathbb{F}$. This assumption is needed for the application of the Schwartz–Zippel Lemma.[5]

▶ **Lemma 4** (Schwartz–Zippel Lemma [50]). *Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero polynomial of total degree d over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, \ldots, r_n$ be chosen independently and uniformly at random from S. Then, the probability that $P(r_1, \ldots, r_n) = 0$ is at most $d/|S|$.*

We assume that two $n \times n$ matrices can be multiplied in $O(n^\omega)$ time. The current best upper bound on ω is $\omega < 2.371339$ [3]. For an $n \times n$ matrix, it is known that computing the determinant, rank, and inverse can all be done in the same time complexity as one matrix multiplication [11, 34].

The assumptions in this subsection were also made in [14, 44].

2.3 Matroid Preliminaries

For the basics of matroid theory, see, for example, the excellent textbooks of Oxley [47] or Schrijiver [49].

A pair $\mathcal{M} = (V, \mathcal{I})$ of a finite set V and a non-empty set family $\mathcal{I} \subseteq 2^V$ is called a *matroid* if the following properties are satisfied.

(Downward closure property) If $I \in \mathcal{I}$ and $J \subseteq I$, then $J \in \mathcal{I}$.

(Augmentation property) If $I, J \in \mathcal{I}$ and $|J| < |I|$, then there exists $v \in I \setminus J$ such that $J \cup \{v\} \in \mathcal{I}$.

A set $I \subseteq V$ is called *independent* if $I \in \mathcal{I}$ and *dependent* otherwise.

Matroid Rank and Span. Motivated by linear algebra, matroids have associated notions of a *rank function* and a *span function*, defined as follows. For a matroid $\mathcal{M} = (V, \mathcal{I})$, the *rank* of $\mathcal{M}$ is $\mathrm{rank}(\mathcal{M}) = \max\{|I| \mid I \in \mathcal{I}\}$. In addition, for any $S \subseteq V$, the *rank* of S is $\mathrm{rank}_{\mathcal{M}}(S) = \max\{|I| \mid I \subseteq S, I \in \mathcal{I}\}$. Moreover, for any $S \subseteq V$, the *span* of S is $\mathrm{span}_{\mathcal{M}}(S) = \{v \in V \mid \mathrm{rank}_{\mathcal{M}}(S \cup \{v\}) = \mathrm{rank}_{\mathcal{M}}(S)\}$.

Matroid Intersection. In the *matroid intersection* problem, we are given two matroids $\mathcal{M}_1 = (V, \mathcal{I}_1)$ and $\mathcal{M}_2 = (V, \mathcal{I}_2)$, and the objective is to find a common independent set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum size.

Edmonds [21] proved the following min-max theorem, which is now well known.

▶ **Theorem 5** (Matroid intersection theorem [21]; see also [49, Theorem 41.1]). *Let $\mathcal{M}_1 = (V, \mathcal{I}_1)$, $\mathcal{M}_2 = (V, \mathcal{I}_2)$ be matroids, with rank function $\mathrm{rank}_{\mathcal{M}_1}$ and $\mathrm{rank}_{\mathcal{M}_2}$, respectively. Then,*

$$\max_{I \in \mathcal{I}_1 \cap \mathcal{I}_2} |I| = \min_{S,T \subseteq V, S \cup T = V} (\mathrm{rank}_{\mathcal{M}_1}(S) + \mathrm{rank}_{\mathcal{M}_2}(T)).$$

[5] The Schwartz–Zippel Lemma is used in Harvey's exact algorithm for linear matroid intersection [33] and in the sketching technique of Cheung–Kwok–Lau [14], both of which are components of our algorithm.

Here, (S, T) is called a dual solution to matroid intersection, where S and T minimize the right-hand side of the equation in Theorem 5.

In our linear matroid intersection algorithm, we use Harvey's exact algorithm for linear matroid intersection [33] to compute primal and dual solutions to sparsified instances.

▶ **Theorem 6** (Harvey's exact linear matroid intersection algorithm; see [33, Section 4.6])**.** *An exact solution to the maximum cardinality linear matroid intersection problem can be computed with high probability in $\tilde{O}(nr^{\omega-1})$ time. Furthermore, an exact dual solution can also be computed with high probability in $\tilde{O}(nr^{\omega-1})$ time.*

2.4 Sketching Technique of Cheung–Kwok–Lau

In our algorithm, we use the sketching technique of Cheung–Kwok–Lau [14]. Using this technique, we can reduce the dimension of the input matrix from $r \times n$ to $O(r_*) \times n$.

▶ **Lemma 7** (Sketching technique of Cheung–Kwok–Lau [14]; from [14, Lemma 2.9])**.** *There is an algorithm that, given an $r \times n$ matrix A over a field $\mathbb{F}$, returns an $O(k) \times n$ matrix A' over a field $\mathbb{F}$ in $O(\mathrm{nnz}(A))$ time, with $\mathrm{nnz}(A') = O(\mathrm{nnz}(A))$, such that for any set $S \subseteq [n]$ of size at most k, if the columns indexed by S are linearly independent in A, then the columns of A' indexed by S are also linearly independent with probability at least $1 - O(1/n^{1/3})$.*

In our fast span computation algorithm, we use the following lemma.

▶ **Lemma 8** (Fast computation of a basis of the orthogonal complement of the column space)**.** *There is a randomized algorithm that, given an $r \times n$ matrix M over a field $\mathbb{F}$, finds a basis of the orthogonal complement of the column space of M with high probability in $O((\mathrm{nnz}(M) + r^{\omega}) \log n)$ time.*

Lemma 8 can be easily derived from the following lemma, which was obtained using the sketching technique of Cheung–Kwok–Lau. Here, the null space of a matrix A is the subspace of vectors $\boldsymbol{x}$ such that $A\boldsymbol{x} = \mathbf{0}$.

▶ **Lemma 9** (from [14, Theorem 1.3(2)]; see also [14, Section 4.1.2])**.** *There is a randomized algorithm that, given an $m \times n$ matrix A over a field $\mathbb{F}$, finds a basis of the null space of A with high probability in $O((\mathrm{nnz}(A) + n(\mathrm{rank}(A))^{\omega-1}) \log n)$ time.*

Let W denote the column space of an $r \times n$ matrix M, i.e., $W = \mathrm{span}\{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n\}$, where $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n$ are the column vectors of M. The orthogonal complement of W, denoted by $W^{\perp}$, is defined as $W^{\perp} = \{\boldsymbol{x} \in \mathbb{F}^r \mid \forall \boldsymbol{w} \in W, \boldsymbol{w}^{\top}\boldsymbol{x} = 0\}$. Now, $\boldsymbol{x} \in W^{\perp}$ if and only if $\boldsymbol{m}_i^{\top}\boldsymbol{x} = 0$ for any $i \in [n]$. We have $W^{\perp} = \{\boldsymbol{x} \in \mathbb{F}^r \mid M^{\top}\boldsymbol{x} = \mathbf{0}\}$. Therefore, to find a basis of the orthogonal complement of the column space of M, we simply compute a basis of the null space of $M^{\top}$ using Lemma 9. Note that $M^{\top}$ is an $n \times r$ matrix. Hence, Lemma 8 follows from Lemma 9.

2.5 Fast Gram–Schmidt Orthonormalization

For our maximum weight linear matroid intersection algorithm for matrices over $\mathbb{R}$, we use the fast Gram–Schmidt orthonormalization algorithm of van den Brand [57].

▶ **Lemma 10** (Fast Gram–Schmidt orthonormalization; from van den Brand [57, Section 4.5])**.** *Given an $r \times n$ matrix M with $r \leq n$ over $\mathbb{R}$, where $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n$ are column vectors of M, there is an algorithm that repeatedly computes $\boldsymbol{m}'_i = \boldsymbol{m}_i - P_{i-1}\boldsymbol{m}_i$, where $P_{i-1}\boldsymbol{m}_i$ denotes the orthogonal projection of $\boldsymbol{m}_i$ onto $\mathrm{span}\{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{i-1}\}$, and then normalizes $\boldsymbol{m}'_i$ if $\boldsymbol{m}'_i \neq \mathbf{0}$. The total running time of this process is $O(n^{\omega})$.*

Algorithm 1 Span computation algorithm.

1 By applying Lemma 8 to $M[S]$, obtain a basis $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_k\}$ of the orthogonal complement of $\operatorname{span}\{\boldsymbol{m}_j \mid j \in S\}$.
2 Let $\boldsymbol{v} = \sum_{j=1}^{k} r_j \boldsymbol{b}_j$, where $r_1, \dots, r_k$ are chosen independently and uniformly at random from a finite subset $F \subseteq \mathbb{F}$ with $|F| = \Theta(n^2)$.
3 Let $T \leftarrow \emptyset$
4 **for** $i \in [n]$ **do**
5 **if** $\boldsymbol{v}^\top \boldsymbol{m}_i = 0$ **then**
6 $T \leftarrow T \cup \{i\}$
7 **return** T

To extend our algorithm to arbitrary fields, we modify a fast Gram–Schmidt orthonormalization algorithm of van den Brand into a fast orthogonalization algorithm with respect to the bilinear form $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \boldsymbol{v}^\top B \boldsymbol{w}$; see the full version [54] for details.

3 Fast Span Computation Algorithm

As part of our technical contribution, we develop a fast span computation algorithm, which is used as a subroutine in our maximum cardinality linear matroid intersection algorithm. The following lemma states its running time guarantee.

▶ **Lemma 11.** *There is a randomized algorithm that, given an $r \times n$ matrix M over a field $\mathbb{F}$ whose columns are the representation of elements of a matroid $\mathcal{M}$, and a set $S \subseteq [n]$,*[6]*, computes $\operatorname{span}_{\mathcal{M}}(S)$ with high probability in $\tilde{O}(\operatorname{nnz}(M) + r^\omega)$ time.*

We note that our algorithm is applicable over any field.

The core idea of our algorithm is very simple. In our algorithm, we first sample a vector $\boldsymbol{v}$ from the orthogonal complement of $\operatorname{span}\{\boldsymbol{m}_j \mid j \in S\}$, where $\boldsymbol{m}_1, \dots, \boldsymbol{m}_n$ denote the vectors corresponding to the columns of the input matrix M. Then, we output the set $\{i \in [n] \mid \boldsymbol{v}^\top \boldsymbol{m}_i = 0\}$ as $\operatorname{span}_{\mathcal{M}}(S)$. See Algorithm 1 for the pseudocode of our algorithm.

Let $W = \operatorname{span}\{\boldsymbol{m}_j \mid j \in S\}$. The orthogonal complement of W, denoted by $W^\perp$, is defined as $W^\perp = \{\boldsymbol{v} \in \mathbb{F}^r \mid \forall \boldsymbol{w} \in W, \boldsymbol{v}^\top \boldsymbol{w} = 0\}$. When considering matrices over a field other than $\mathbb{R}$, it may occur that W does not satisfy $W \cap W^\perp = \{\boldsymbol{0}\}$ or $W + W^\perp = \mathbb{F}^r$. Nevertheless, for our purpose it suffices that $(W^\perp)^\perp = W$, which in fact holds over any field.[7]

Note that we assume $|\mathbb{F}| = \Omega(\operatorname{poly}(n))$. Recall that, by Lemma 3, this assumption can be made without loss of generality.

Proof. In our algorithm, we first compute a basis $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_k\}$ of the orthogonal complement of $\operatorname{span}\{\boldsymbol{m}_j \mid j \in S\}$ by applying Lemma 8 to $M[S]$. This step takes $O((\operatorname{nnz}(M) + r^\omega) \log n)$ time.

[6] Although in our application the size of S is always $\tilde{O}_\varepsilon(r)$, we state the lemma in a general form for potential broader applicability.

[7] This can be verified by a simple argument in linear algebra. Now, fix any $\boldsymbol{x} \in W$. For any $\boldsymbol{y} \in W^\perp$, we have $\boldsymbol{y}^\top \boldsymbol{x} = 0$. Thus, $\boldsymbol{x} \in (W^\perp)^\perp$, and hence $W \subseteq (W^\perp)^\perp$. Moreover, the Rank–nullity theorem implies that $\dim(W^\perp) = r - \dim(W)$ and $\dim((W^\perp)^\perp) = r - \dim(W^\perp)$. Therefore, $\dim((W^\perp)^\perp) = \dim(W)$, which, together with $W \subseteq (W^\perp)^\perp$, yields $(W^\perp)^\perp = W$.

Next, we choose $r_1, \ldots, r_k$ independently and uniformly at random from a finite subset $F \subseteq \mathbb{F}$ with $|F| = \Theta(n^2)$, and compute

$$\boldsymbol{v} = \sum_{j=1}^{k} r_j \boldsymbol{b}_j,$$

which takes $O(r^2)$ time, since $k \leq r$ and each $\boldsymbol{b}_j$ is an r-dimensional vector.

We claim that the value of $\boldsymbol{v}^\top \boldsymbol{m}_i$ satisfies the following two properties. Here, $\boldsymbol{m_1}, \ldots, \boldsymbol{m_n}$ denote the vectors corresponding to the columns of the matrix M.

- If $i \in \mathrm{span}_{\mathcal{M}}(S)$, then we have $\boldsymbol{v}^\top \boldsymbol{m}_i = 0$ for every possible choice of $r_1, \ldots, r_k$, since $\boldsymbol{v}$ lies in the orthogonal complement of $\mathrm{span}\{\boldsymbol{m}_j \mid j \in S\}$.
- If $i \notin \mathrm{span}_{\mathcal{M}}(S)$, then we have $\boldsymbol{v}^\top \boldsymbol{m}_i \neq 0$ with probability at least $1 - 1/|F|$. This can be shown as follows. Consider the multivariate polynomial

  $$P(x_1, \ldots, x_k) = \sum_{j=1}^{k} x_j \boldsymbol{b}_j^\top \boldsymbol{m}_i.$$

  Here, to show that $P(x_1, \ldots, x_k)$ is a nonzero polynomial, we prove the following claim.[8] Recall that we are assuming $i \notin \mathrm{span}_{\mathcal{M}}(S)$.

  ▷ Claim 12. There exists some $j \in [k]$ such that $\boldsymbol{b}_j^\top \boldsymbol{m}_i \neq 0$.

  Proof of Claim 12. Let $W = \mathrm{span}\{\boldsymbol{m}_j \mid j \in S\}$. Here, we have $\boldsymbol{m}_i \notin W$. Suppose, for the sake of contradiction, that $\boldsymbol{b}_j^\top \boldsymbol{m}_i = 0$ for all $j \in [k]$, where $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ form a basis of $W^\perp$. Then, $\boldsymbol{m}_i$ is orthogonal to every vector in $W^\perp$, which implies that $\boldsymbol{m}_i \in (W^\perp)^\perp = W$, contradicting the assumption $\boldsymbol{m}_i \notin W$. Therefore, there exists some $j \in [k]$ such that $\boldsymbol{b}_j^\top \boldsymbol{m}_i \neq 0$, which completes the proof. ◁

  By Claim 12, $P(x_1, \ldots, x_k)$ is a nonzero polynomial of total degree 1. Since $r_1, \ldots, r_k$ are chosen independently and uniformly at random from a finite subset $F \subseteq \mathbb{F}$, by the Schwartz–Zippel Lemma (Lemma 4), the probability that

  $$P(r_1, \ldots, r_k) = \sum_{j=1}^{k} r_j \boldsymbol{b}_j^\top \boldsymbol{m}_i = \boldsymbol{v}^\top \boldsymbol{m}_i = 0$$

  is at most $1/|F|$.

Then, we only need to check, for each $i \in [n]$, whether $\boldsymbol{v}^\top \boldsymbol{m}_i = 0$. The total time required to compute all inner products $\boldsymbol{v}^\top \boldsymbol{m}_i$ for all $i \in [n]$ is $O(\mathrm{nnz}(M))$. Finally, we output the set $\{i \in [n] \mid \boldsymbol{v}^\top \boldsymbol{m}_i = 0\}$ as $\mathrm{span}_{\mathcal{M}}(S)$.

There are at most n operations that check whether $i \in \mathrm{span}_{\mathcal{M}}(S)$, and each fails with probability at most $1/|F|$. Therefore, the probability that at least one of them fails is at most $n/|F| = O(1/n)$, where we recall that $|F| = \Theta(n^2)$.

In summary, we can compute $\mathrm{span}_{\mathcal{M}}(S)$ with high probability in $O((\mathrm{nnz}(M) + r^\omega) \log n)$ time. ◀

[8] Over the real number field $\mathbb{R}$, the claim is immediate. However, over an arbitrary field $\mathbb{F}$, one cannot in general rely on $W \cap W^\perp = \{\boldsymbol{0}\}$ or $W + W^\perp = \mathbb{F}^r$, so some care is required. For completeness, we give a careful proof, which uses only the identity $(W^\perp)^\perp = W$.

Algorithm 2 Quanrud's maximum cardinality matroid intersection algorithm in the independence oracle model (from [48, Figure 1]).

1 Let $w(e) \leftarrow 1$ for all $e \in V$.
2 **for** $\ell = 1$ *to* L **do** // $L = O(\log(n)/\varepsilon)$
3 Sample a set $V' \subseteq V$ of $\Theta(\min\{k \log(n)/\varepsilon, n\})$ elements, where each element $e \in V$ is chosen with probability proportional to $w(e)$.
4 Compute a maximum common independent set $I^{(\ell)}$ and a dual solution $(\tilde{S}^{\ell}, \tilde{T}^{\ell})$ for the subproblem over V'.
5 Let $S^{(\ell)} = \mathrm{span}_{\mathcal{M}_1}(\tilde{S}^{(\ell)})$ and $T^{(\ell)} = \mathrm{span}_{\mathcal{M}_2}(\tilde{T}^{(\ell)})$.
6 For each element $e \in S^{(\ell)} \cup T^{(\ell)}$, set $w(e) \leftarrow w(e)/2$.
7 Output $I^{(\ell)}$ for some iteration $\ell \in [L]$ for which $|I^{(\ell)}|$ is maximized over all iterations.

4 Adaptive Sparsification Framework of Quanrud

In this section, we describe the adaptive sparsification framework of Quanrud [48], which was developed to obtain a fast $(1-\varepsilon)$-approximation algorithm for matroid intersection in the independence oracle model. Our linear matroid intersection algorithm builds upon this framework.

Quanrud's framework employs a multiplicative weight update method, inspired by Assadi's semi-streaming maximum matching algorithm [4]. In this framework, we maintain a weight for each element and repeat the following $O(\log(n)/\varepsilon)$ iterations: In each iteration, we sample $\Theta(\min\{k \log(n)/\varepsilon, n\})$ elements, in proportion to their weight, compute primal and dual solutions over the sampled elements, and then update the weights of elements. In Quanrud's paper, k is set to r_*, where r_* denote the maximum size of a common independent set.[9]

4.1 Maximum Cardinality Matroid Intersection

In this subsection, we describe Quanrud's framework for maximum cardinality matroid intersection. The pseudocode is shown in Algorithm 2.

We maintain a weight function $w : V \to \mathbb{R}_{\geq 0}$ over the elements, where all elements initially have equal weights. We then repeat the following process for $L = O(\log(n)/\varepsilon)$ iterations: In each iteration, we first sample $\tilde{O}(k/\varepsilon)$ elements $V' \subseteq V$ in proportion to w. Then, we compute a maximum common independent set I and a dual solution $(\tilde{S}, \tilde{T})$ for the subproblem over V'.[10] Note that $\tilde{S}, \tilde{T} \subseteq V'$. We extend these sets to their spans, setting $S = \mathrm{span}_{\mathcal{M}_1}(\tilde{S})$ and $T = \mathrm{span}_{\mathcal{M}_2}(\tilde{T})$. Finally, we update the weights w by setting $w(e) \leftarrow w(e)/2$ for all elements $e \in S \cup T$. After L iterations, we output I with the largest cardinality among all iterations.

When applying Quanrud's framework to linear matroid intersection, we use Harvey's linear matroid intersection algorithm (Theorem 6) to implement Line 4 of Algorithm 2, and our fast span computation algorithm presented in Section 3 to implement Line 5. See Section 5.1 for details.

[9] When we apply Quanrud's framework to linear matroid intersection, we set $k = r$ instead of $k = r_*$.
[10] In Quanrud's paper, a $(1-\varepsilon)$-approximate solution is computed in Line 4 of Algorithm 2. However, when applying Quanrud's framework to linear matroid intersection, we compute an exact solution. It is clear that this does not affect the correctness.

Algorithm 3 Quanrud's maximum weight matroid intersection algorithm in the independence oracle model (from [48, Figure 2]).

Let $w(e) \leftarrow 1$ for all $e \in V$.
for $\ell = 1$ *to* L **do** // $L = O(\log(n)/\varepsilon)$
  Sample a set $V' \subseteq V$ of $\Theta(\min\{k\log(n)/\varepsilon, n\})$ elements, where each element $e \in V$ is chosen with probability proportional to $w(e)$.
  Compute a $(1-\varepsilon)$-approximate maximum-weight common independent set $I^{(\ell)}$ and a compact $(1-\varepsilon)$-approximate dual solution $\tilde{y}^{(\ell)}, \tilde{z}^{(\ell)} : 2^{V'} \to \mathbb{R}_{\geq 0}$ for the subproblem over V'.
  Define $y^{(\ell)}, z^{(\ell)} : 2^V \to \mathbb{R}_{\geq 0}$ by

$$y^{(\ell)}(\mathrm{span}_{\mathcal{M}_1}(S')) = \lceil \tilde{y}^{(\ell)}(S') \rceil_{1+\varepsilon} \qquad \text{for all } S' \in \mathrm{support}(\tilde{y}^{(\ell)}),$$
$$z^{(\ell)}(\mathrm{span}_{\mathcal{M}_2}(T')) = \lceil \tilde{z}^{(\ell)}(T') \rceil_{1+\varepsilon} \qquad \text{for all } T' \in \mathrm{support}(\tilde{z}^{(\ell)}),$$

  where $\lceil x \rceil_{1+\varepsilon} = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} x \rceil}$ rounds x up to the nearest power of $1+\varepsilon$.
  For each element $e \in V$, if $\sum_{S \ni e} (y(S) + z(S)) \geq c(e)$ holds, then set $w(e) \leftarrow w(e)/2$.
Output $I^{(\ell)}$ for some iteration $\ell \in [L]$ for which $c(I^{(\ell)})$ is maximized over all iterations.

4.2 Maximum Weight Matroid Intersection

In this subsection, we describe Quanrud's framework for maximum weight matroid intersection. The high-level approach in the weighted case is similar to that in the unweighted case, but requires additional technical sophistication. The pseudocode is shown in Algorithm 3.

Here, we refer to a solution (y, z), where $y, z : 2^V \to \mathbb{R}_{\geq 0}$, of the following LP as a dual solution. This LP is the dual of the LP relaxation of the weighted matroid intersection problem.

$$\begin{aligned} \text{minimize} \quad & \sum_{S \subseteq V} (\mathrm{rank}_{\mathcal{M}_1}(S) y(S) + \mathrm{rank}_{\mathcal{M}_2}(S) z(S)) \quad \text{over } y, z : 2^V \to \mathbb{R}_{\geq 0} \\ \text{s.t.} \quad & \sum_{S \ni e} (y(S) + z(S)) \geq c(e) \quad \text{for all } e \in V \end{aligned} \tag{1}$$

Quanrud defines (y, z) to be *compact* if the supports of y and z satisfy the following conditions:

$$\mathrm{support}(y) \subseteq \{\mathrm{span}_{\mathcal{M}_1}(\{e_1\}), \mathrm{span}_{\mathcal{M}_1}(\{e_1, e_2\}), \ldots, \mathrm{span}_{\mathcal{M}_1}(\{e_1, \ldots, e_k\})\},$$
$$\mathrm{support}(z) \subseteq \{\mathrm{span}_{\mathcal{M}_2}(\{f_1\}), \mathrm{span}_{\mathcal{M}_2}(\{f_1, f_2\}), \ldots, \mathrm{span}_{\mathcal{M}_2}(\{f_1, \ldots, f_k\})\}.$$

Here, $e_1, \ldots, e_k \in V$ and $f_1, \ldots, f_k \in V$ are two sequences of k elements.

As in the unweighted case, we maintain a weight function $w : V \to \mathbb{R}_{\geq 0}$ with initially equal weights, and repeat the following process for $L = O(\log(n)/\varepsilon)$ iterations: In each iteration, we first sample $\tilde{O}(k/\varepsilon)$ elements $V' \subseteq V$ in proportion to w. Then, we compute a $(1-\varepsilon)$-approximate maximum-weight common independent set I and a compact $(1-\varepsilon)$-approximate dual solution $\tilde{y}, \tilde{z} : 2^{V'} \to \mathbb{R}_{\geq 0}$ for the subproblem over V'. We then construct $y, z : 2^V \to \mathbb{R}_{\geq 0}$ from $\tilde{y}, \tilde{z}$ by setting

$$y(\mathrm{span}_{\mathcal{M}_1}(S')) = \lceil \tilde{y}(S') \rceil_{1+\varepsilon} \qquad \text{for all } S' \in \mathrm{support}(\tilde{y}),$$
$$z(\mathrm{span}_{\mathcal{M}_2}(T')) = \lceil \tilde{z}(T') \rceil_{1+\varepsilon} \qquad \text{for all } T' \in \mathrm{support}(\tilde{z}).$$

We set $y(F) = 0$ (resp. $z(F) = 0$) for any $F \in 2^V$ not of the form $\mathrm{span}_{\mathcal{M}_1}(S')$ with $S' \in \mathrm{support}(\tilde{y})$ (resp. $\mathrm{span}_{\mathcal{M}_2}(T')$ with $T' \in \mathrm{support}(\tilde{z})$). Here, $\lceil x \rceil_{1+\varepsilon} = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} x \rceil}$ denote rounding x up to the nearest power of $1+\varepsilon$. Since (y', z') is compact, there exists a sequence $e_1, \ldots, e_k$ such that every set in the support of y' has the form $S'_i = (\mathrm{span}_{\mathcal{M}_1}(\{e_1, \ldots, e_i\})) \cap V'$. Then, every set in the support of y has the form $S_i = \mathrm{span}_{\mathcal{M}_1}(S'_i) = \mathrm{span}_{\mathcal{M}_1}(\{e_1, \ldots, e_i\})$. Symmetrically, the same property holds for z, with a different sequence of k elements determined by z'. Thus, (y, z) is also compact. Finally, we update the weights w by setting $w(e) \leftarrow w(e)/2$ for all elements $e \in V$ such that $\sum_{S \ni e}(y(S) + z(S)) \geq c(e)$ holds. After L iterations, we output I that maximizes $c(I)$ over all iterations.

When applying Quanrud's framework to maximum weight linear matroid intersection, we use the $(1-\varepsilon)$-approximate weighted linear matroid intersection algorithm by Huang–Kakimura–Kamiyama [35] to implement Line 4 of Algorithm 3. To implement Line 6, we combine our fast span computation algorithm presented in Section 3 with the fast Gram–Schmidt orthonormalization algorithm of van den Brand [57] (Lemma 10). See Section 5.2 for details.

5 Linear Matroid Intersection Algorithms

In this section, we prove Theorems 1 and 2 by combining the sparsification framework of Quanrud [48] (see Section 4 for details) with our fast span computation algorithm presented in Section 3.

When we apply Quanrud's framework to linear matroid intersection, we set $k = r$ instead of $k = r_*$. In the proofs of Theorems 1 and 2, we apply Quanrud's framework to matrices that have been suitably preprocessed using the sketching technique of Cheung–Kwok–Lau [14] (Lemma 7). For the preprocessed matrices, the maximum size of a common independent set is within a constant factor of the number of rows, which justifies setting $k = r$.[11]

5.1 Maximum Cardinality Linear Matroid Intersection

We first show the following theorem, and then prove Theorem 1 using this theorem.

▶ **Theorem 13.** *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximate solution to the maximum cardinality linear matroid intersection problem can be computed with high probability in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$ time.*

Proof. We show that, when the matroids are given by matrix representations, Quanrud's maximum cardinality matroid intersection algorithm (Algorithm 2) can be implemented in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$ time. By using Harvey's exact algorithm for linear matroid intersection (Theorem 6), Line 4 can be implemented in $\tilde{O}_\varepsilon(r^\omega)$ time, since the number of elements in the sparsified set is $\tilde{O}_\varepsilon(r)$. By applying Lemma 11, Line 5 can be implemented in $\tilde{O}(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$ time. Furthermore, Lines 3 and 6 can be implemented in $\tilde{O}_\varepsilon(n)$ time. Since the number of iterations of the for loop is $O(\log(n)/\varepsilon)$, the total running time is the desired bound of $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$, which completes the proof. ◀

Following the approach of [14, Section 4.3.2], we can improve the running time to $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$, which proves Theorem 1.

[11] Actually, sampling $\Theta(\min\{r \log(n)/\varepsilon, n\})$ elements, instead of $\Theta(\min\{r_* \log(n)/\varepsilon, n\})$, in each iteration does not affect the correctness of Quanrud's framework. This is because the approximation guarantee relies on Lemma 9 (and Lemma 13 in the weighted case) in Quanrud's paper [48], which states that sampling $\Omega(\min\{r_* \log(n)/\varepsilon, n\})$ elements is sufficient.

Proof of Theorem 1. We assume without loss of generality that $\varepsilon < 1/2$.

Given an integer ℓ, we consider the task of finding a set $I' \subseteq [n]$ of size at least $\min\{(1-\varepsilon)\ell, (1-\varepsilon)\cdot r_*\}$ such that the columns indexed by I' are linearly independent in both M_1 and M_2. Using Lemma 7 with $k = \ell$, we compress the input matrices M_1 and M_2 into $O(\ell) \times n$ matrices M_1' and M_2', in $O(\mathrm{nnz}(M_1))$ and $O(\mathrm{nnz}(M_2))$ time, respectively. We have $\mathrm{nnz}(M_1') = O(\mathrm{nnz}(M_1))$ and $\mathrm{nnz}(M_2') = O(\mathrm{nnz}(M_2))$. Furthermore, for any set $S \subseteq [n]$ of size at most ℓ, if the columns indexed by S are linearly independent in both M_1 and M_2, then the columns indexed by S are also linearly independent in both M_1' and M_2' with probability at least $1 - O(1/n^{1/3})$. Consequently, the maximum size of a common independent set for M_1' and M_2' is $\min\{\ell, r_*\}$ with high probability. Then, we apply the algorithm in Theorem 13 to find a $(1-\varepsilon)$-approximate solution to the linear matroid intersection problem on M_1' and M_2', which runs in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1') + \mathrm{nnz}(M_2') + \ell^\omega) = \tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + \ell^\omega)$ time.

To find a solution of size at least $(1-\varepsilon)\cdot r_*$, we set $\ell = 2, 4, 8, \ldots$ and apply the above algorithm until there is no solution of size $\ell/2$. The total running time of this algorithm is $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r_*^\omega)$, which completes the proof. ◀

5.2 Maximum Weight Linear Matroid Intersection

We first show the following theorem, and then prove Theorem 2 using this theorem.

▶ **Theorem 14.** *For any $\varepsilon > 0$, a $(1-\varepsilon)$-approximate solution to the maximum weight linear matroid intersection problem can be computed with high probability in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$ time.*

In Quanrud's framework, we need to compute a $(1-\varepsilon)$-approximate maximum-weight common independent set and a compact $(1+\varepsilon)$-approximate dual solution for the sparsified instances. When implementing Quanrud's framework for linear matroids, we use the following lemma to compute compact primal-dual solutions. This lemma is obtained by combining the $(1-\varepsilon)$-approximation algorithm for the weighted matroid intersection by Huang–Kakimura–Kamiyama [35] with the argument of Quanrud's paper [48]. See the full version [54] for the proof of Lemma 15.

▶ **Lemma 15.** *Given two $r \times n$ matrix M_1 and M_2 over a field $\mathbb{F}$, and a weight function $c : [n] \to \mathbb{R}_{\geq 0}$, there is an algorithm that computes a $(1-\varepsilon)$-approximate maximum-weight common independent set I, along with a compact $(1+\varepsilon)$-approximate dual solution $y, z : 2^{[n]} \to \mathbb{R}_{\geq 0}$, with high probability in $\tilde{O}_\varepsilon(nr^{\omega-1})$ time.*

To efficiently implement Line 6 of Algorithm 3, we present the following lemma. This lemma is obtained by combining our fast span computation algorithm described in Section 3 with the fast Gram–Schmidt orthonormalization algorithm of van den Brand (Lemma 10).

▶ **Lemma 16.** *Let M be an $r \times n$ matrix over a field $\mathbb{F}$, and let $\{i_1, \ldots, i_r\} \subseteq [n]$ be a set of indices. Here, $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n$ denote the vectors corresponding to the columns of M. Then, there is an algorithm that answers the following query: given $j \in [n]$ and $\alpha \in [r]$, determine whether $\boldsymbol{m}_j \in \mathrm{span}\{\boldsymbol{m}_{i_\beta} \mid \beta \leq \alpha\}$ with high probability in $O(\mathrm{nnz}(\boldsymbol{m}_j))$ time. This algorithm requires $O(r^\omega)$ time for preprocessing.*

Here, we provide a proof only for matrices over $\mathbb{R}$. The extension to matrices over an arbitrary field is deferred to the full version [54]. To make the algorithm applicable over arbitrary fields, the key modification is to replace the Euclidean inner product $\boldsymbol{v}^\top \boldsymbol{w}$ with the bilinear form $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \boldsymbol{v}^\top B \boldsymbol{w}$ (see the full version [54] for details).

Proof of Lemma 16 for matrices over $\mathbb{R}$. We first describe the preprocessing algorithm. First, we apply the fast Gram–Schmidt orthonormalization algorithm of van den Brand [57] (Lemma 10) to the sequence of $2r$ vectors $\boldsymbol{m}_{i_1}, \dots, \boldsymbol{m}_{i_r}, \boldsymbol{e}_1, \dots, \boldsymbol{e}_r$, which takes $O(r^\omega)$ time. Here, $\boldsymbol{e}_i \in \mathbb{R}^r$ is the standard basis vector (i.e., the vector with 1 in the i-th coordinate and 0 elsewhere). Then, we obtain a vector sequence $\boldsymbol{m}'_{i_1}, \dots, \boldsymbol{m}'_{i_r}, \boldsymbol{e}'_1, \dots, \boldsymbol{e}'_r$. We note that this vector sequence includes the zero vector $\boldsymbol{0}$. We then choose $q_1, \dots, q_{2r}$ independently and uniformly at random from a finite subset $R \subset \mathbb{R}$ of size $\mathrm{poly}(n)$ (e.g., $R = \{1, 2, \dots, \mathrm{poly}(n)\}$). Next, for each $\alpha \in [r]$, we compute a vector $\boldsymbol{v}_\alpha = \sum_{j=\alpha+1}^{r} q_j \boldsymbol{m}'_{i_j} + \sum_{j=1}^{r} q_{r+j} \boldsymbol{e}'_j$. This can be implemented in $O(r^2)$ time by computing the sums in reverse order. We note that $\boldsymbol{v}_\alpha$ lies in the orthogonal complement of $\mathrm{span}\{\boldsymbol{m}_{i_1}, \dots, \boldsymbol{m}_{i_\alpha}\}$.

Next, we describe how to answer the query. It suffices to check whether $\boldsymbol{v}_\alpha^\top \boldsymbol{m}_j = 0$, which can be done in $O(\mathrm{nnz}(\boldsymbol{m}_j))$ time. As in the proof of Lemma 11, if $\boldsymbol{m}_j \in \mathrm{span}\{\boldsymbol{m}_{i_\beta} \mid \beta \le \alpha\}$ holds, then $\boldsymbol{v}_\alpha^\top \boldsymbol{m}_j = 0$ always holds, otherwise $\boldsymbol{v}_\alpha^\top \boldsymbol{m}_j \neq 0$ holds with probability at least $1 - 1/|R| = 1 - 1/\mathrm{poly}(n)$. Therefore, checking whether $\boldsymbol{v}_\alpha^\top \boldsymbol{m}_j = 0$ suffices to decide whether $\boldsymbol{m}_j \in \mathrm{span}\{\boldsymbol{m}_{i_\beta} \mid \beta \le \alpha\}$, which completes the proof. ◀

We now present the proof of Theorem 14.

Proof of Theorem 14. We show that, when the matroids are given by matrix representations, Quanrud's maximum weight matroid intersection algorithm (Algorithm 3) can be implemented in $\tilde{O}_\varepsilon(\mathrm{nnz}(M_1) + \mathrm{nnz}(M_2) + r^\omega)$ time. By Lemma 15, Line 4 can be implemented in $\tilde{O}_\varepsilon(r^\omega)$ time, since the number of elements in the sparsified set is $\tilde{O}_\varepsilon(r)$.

In our algorithm, we do not explicitly compute (y, z) computed in Line 5. Here, (y, z) computed in Line 5 is compact, and thus we can write

$$\begin{aligned} \mathrm{support}(y) \subseteq &\{\mathrm{span}_{\mathcal{M}_1}(\{i_1\}), \mathrm{span}_{\mathcal{M}_1}(\{i_1, i_2\}), \dots, \mathrm{span}_{\mathcal{M}_1}(\{i_1, \dots, i_r\})\}, \\ \mathrm{support}(z) \subseteq &\{\mathrm{span}_{\mathcal{M}_2}(\{j_1\}), \mathrm{span}_{\mathcal{M}_2}(\{j_1, j_2\}), \dots, \mathrm{span}_{\mathcal{M}_2}(\{j_1, \dots, j_r\})\}, \end{aligned} \tag{2}$$

for two sequences of r elements $i_1, \dots, i_r \in [n]$ and $j_1, \dots, j_r \in [n]$. Here, let $\mathcal{M}_1$ (resp. $\mathcal{M}_2$) denote the matroid represented by the matrix M_1 (resp. M_2). The elements $i_1, \dots, i_r$ and $j_1, \dots, j_r$ belong to the sparsified set, and are computed in Line 4. To implement Line 5, we simply compute the value of $y(\mathrm{span}_{\mathcal{M}_1}(\{i_1, \dots, i_\alpha\}))$ and $z(\mathrm{span}_{\mathcal{M}_2}(\{j_1, \dots, j_\alpha\}))$ for each $\alpha \in [r]$, which takes $O(r)$ time. Note that we do not explicitly compute $\mathrm{span}_{\mathcal{M}_1}(\{i_1, \dots, i_\alpha\})$ or $\mathrm{span}_{\mathcal{M}_2}(\{j_1, \dots, j_\alpha\})$.

It remains to bound the running time of Line 6. In this line, for each $i \in [n]$, we need to check whether $\sum_{S \ni i} (y(S) + z(S)) \ge c(i)$. Here, $\boldsymbol{m}_1^{(1)}, \dots, \boldsymbol{m}_n^{(1)}$ (resp. $\boldsymbol{m}_1^{(2)}, \dots, \boldsymbol{m}_n^{(2)}$) denote the vectors corresponding to the columns of M_1 (resp. M_2). As preprocessing, for all $\alpha \in [r]$, we compute

$$y_\alpha = \sum_{\gamma \in \{\alpha, \alpha+1, \dots, r\}} y(\mathrm{span}_{\mathcal{M}_1}(\{i_1, \dots, i_\gamma\})), \quad z_\alpha = \sum_{\gamma \in \{\alpha, \alpha+1, \dots, r\}} z(\mathrm{span}_{\mathcal{M}_2}(\{j_1, \dots, j_\gamma\})),$$

which takes $O(r)$ time. Furthermore, we apply the preprocessing algorithm of Lemma 16 to M_1 and $\{i_1, \dots, i_r\}$, which takes $O(r^\omega)$ time; we do the same for M_2 and $\{j_1, \dots, j_r\}$.

We now describe how to check whether $\sum_{S \ni i} (y(S) + z(S)) \ge c(i)$ holds for each $i \in [n]$ in $\tilde{O}(\mathrm{nnz}(\boldsymbol{m}_i^{(1)}) + \mathrm{nnz}(\boldsymbol{m}_i^{(2)}))$ time. We can binary search for the first index $\alpha^{(1)} \in [r]$ (resp. $\alpha^{(2)} \in [r]$) such that $\boldsymbol{m}_i^{(1)} \in \mathrm{span}\{\boldsymbol{m}_{i_\beta}^{(1)} \mid \beta \le \alpha^{(1)}\}$ (resp. $\boldsymbol{m}_i^{(2)} \in \mathrm{span}\{\boldsymbol{m}_{j_\beta}^{(2)} \mid \beta \le \alpha^{(2)}\}$).

Note that, for all $\gamma \geq \alpha^{(1)}$, we have $\boldsymbol{m}_i^{(1)} \in \operatorname{span}\{\boldsymbol{m}_{i_\beta}^{(1)} \mid \beta \leq \gamma\}$. By Lemma 16, this binary search process takes $\tilde{O}(\operatorname{nnz}(\boldsymbol{m}_i^{(1)}) + \operatorname{nnz}(\boldsymbol{m}_i^{(2)}))$ time. Here, we have $\sum_{S \ni i}(y(S) + z(S)) = y_{\alpha^{(1)}} + z_{\alpha^{(2)}}$, so this value can be computed in $O(1)$ time. Thus, for each $i \in [n]$, we can check whether $\sum_{S \ni i}(y(S) + z(S)) \geq c(i)$ in $\tilde{O}(\operatorname{nnz}(\boldsymbol{m}_i^{(1)}) + \operatorname{nnz}(\boldsymbol{m}_i^{(2)}))$ time. Therefore, Line 6 can be implemented in $\tilde{O}(\operatorname{nnz}(M_1) + \operatorname{nnz}(M_2) + r^\omega)$ time, which completes the proof. ◀

Using the sketching technique of Cheung–Kwok–Lau [14] (Lemma 7), we can improve the running time to $\tilde{O}_\varepsilon(\operatorname{nnz}(M_1) + \operatorname{nnz}(M_2) + r_*^\omega)$, which proves Theorem 2.

Proof of Theorem 2. As a first step, we compute a 2/3-approximate solution $\bar{S}_*$ to the maximum cardinality linear matroid intersection for the input matrices M_1 and M_2, using Theorem 1, in $\tilde{O}_\varepsilon(\operatorname{nnz}(M_1) + \operatorname{nnz}(M_2) + r_*^\omega)$ time. We then apply Lemma 7 with $k = \frac{3}{2}|\bar{S}_*|$. Note that $k \geq r_*$ and $k = \Theta(r_*)$. The lemma compresses the input matrices M_1 and M_2 into $O(r_*) \times n$ matrices M_1' and M_2', in $O(\operatorname{nnz}(M_1))$ and $O(\operatorname{nnz}(M_2))$ time, respectively. We have $\operatorname{nnz}(M_1') = O(\operatorname{nnz}(M_1))$ and $\operatorname{nnz}(M_2') = O(\operatorname{nnz}(M_2))$. Furthermore, for any set $S \subseteq [n]$ of size at most r_*, if the columns indexed by S are linearly independent in both M_1 and M_2, then the columns indexed by S are also linearly independent in both M_1' and M_2' with probability at least $1 - O(1/n^{1/3})$. We then apply the algorithm in Theorem 14 to the compressed matrices M_1' and M_2' to find a $(1-\varepsilon)$-approximate solution to the maximum weight linear matroid intersection problem. This algorithm runs in $\tilde{O}_\varepsilon(\operatorname{nnz}(M_1') + \operatorname{nnz}(M_2') + r_*^\omega) = \tilde{O}_\varepsilon(\operatorname{nnz}(M_1) + \operatorname{nnz}(M_2) + r_*^\omega)$ time. ◀

References

1 Aryan Agarwala, Yaroslav Alekseev, and Antoine Vinciguerra. Linear matroid intersection is in catalytic logspace. In *Proceedings of the 17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*, volume 362, pages 3:1–3:23, 2026. doi:10.4230/LIPIcs.ITCS.2026.3.

2 Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971. doi:10.1090/S0002-9947-1971-0286689-5.

3 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

4 Sepehr Assadi. A simple $(1-\varepsilon)$-approximation semi-streaming algorithm for maximum (weighted) matching. *TheoretiCS*, 4, 2025. Preliminary version in SOSA 2024. doi:10.46298/theoretics.25.16.

5 Alexander I Barvinok. New algorithms for linear k-matroid intersection and matroid k-parity problems. *Mathematical Programming*, 69(1):449–470, 1995. doi:10.1007/BF01585571.

6 Joakim Blikstad. Breaking $O(nr)$ for matroid intersection. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 198, pages 31:1–31:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.31.

7 Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. Fast algorithms via dynamic-oracle matroids. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 1229–1242, 2023. doi:10.1145/3564246.3585219.

8 Joakim Blikstad and Ta-Wei Tu. Efficient matroid intersection via a batch-update auction algorithm. In *Proceedings of the 8th Symposium on Simplicity in Algorithms (SOSA 2025)*, pages 226–237. SIAM, 2025. doi:10.1137/1.9781611978315.18.

9 Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. Breaking the quadratic barrier for matroid intersection. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 421–432, 2021. doi:10.1145/3406325.3451092.

10 Cornelius Brand and Kevin Pratt. Parameterized applications of symbolic differentiation of (totally) multilinear polynomials. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 38:1–38:19, 2021. doi:10.4230/LIPIcs.ICALP.2021.38.

11 James R Bunch and John E Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

12 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 1146–1168. IEEE, 2019. doi:10.1109/FOCS.2019.00072.

13 Chandra Chekuri and Kent Quanrud. A fast approximation for maximum weight matroid intersection. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 445–457. SIAM, 2016. doi:10.1137/1.9781611974331.ch33.

14 Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *Journal of the ACM (JACM)*, 60(5):1–25, 2013. Preliminary version in STOC 2012. doi:10.1145/2528404.

15 Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Algebraic algorithms for linear matroid parity problems. *ACM Transactions on Algorithms (TALG)*, 10(3):1–26, 2014. Preliminary version in SODA 2011. doi:10.1145/2601066.

16 William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986. doi:10.1137/0215066.

17 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

18 Aditi Dudeja and Mara Grilnberger. A weighted-to-unweighted reduction for matroid intersection. *arXiv preprint arXiv:2602.15702*, 2026. doi:10.48550/arXiv.2602.15702.

19 Jean-Guillaume Dumas, Clément Pernet, and Ziad Sultan. Simultaneous computation of the row and column rank profiles. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation (ISSAC 2013)*, pages 181–188, 2013. doi:10.1145/2465506.2465517.

20 Jean-Guillaume Dumas, Clément Pernet, and Ziad Sultan. Fast computation of the rank profile matrix and the generalized bruhat decomposition. *Journal of Symbolic Computation*, 83:187–210, 2017. doi:10.1016/j.jsc.2016.11.011.

21 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

22 Jack Edmonds. Matroid intersection. In *Annals of Discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979. doi:10.1016/S0167-5060(08)70817-3.

23 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving. *TheoretiCS*, 4, 2025. Preliminary version in SODA 2024. doi:10.46298/theoretics.25.21.

24 Rūsiņš Freivalds. Fast probabilistic algorithms. In *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science (MFCS 1979)*, pages 57–69. Springer, 1979.

25 Satoru Fujishige and Zhang Xiaodong. An efficient cost scaling algorithm for the independent assignment problem. *Journal of the Operations Research Society of Japan*, 38(1):124–136, 1995.

26 Harold N Gabow and Matthias Stallmann. Efficient algorithms for graphic matroid intersection and parity. In *Proceedings of the 12th Colloquium on Automata, Languages and Programming (ICALP 1985)*, pages 210–220. Springer, 1985. doi:10.1007/BFb0015746.

27 Harold N Gabow and Robert E Tarjan. Efficient algorithms for simple matroid intersection problems. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS 1979)*, pages 196–204. IEEE, 1979. doi:10.1109/SFCS.1979.14.

28 Harold N Gabow and Ying Xu. Efficient algorithms for independent assignment on graphic and linear matroids. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 106–111. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63463.

29 Harold N Gabow and Ying Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *Journal of Computer and System Sciences*, 53(1):129–147, 1996. doi:10.1006/jcss.1996.0054.

30 Rohit Gurjar, Taihei Oki, and Roshan Raj. Fractional linear matroid matching is in Quasi-NC. In *Proceedings of the 32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308, pages 63:1–63:14, 2024. doi:10.4230/LIPIcs.ESA.2024.63.

31 Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in Quasi-NC. *computational complexity*, 29(2):9, 2020. Preliminary version in STOC 2017. doi:10.1007/s00037-020-00200-z.

32 Nicholas JA Harvey. An algebraic algorithm for weighted linear matroid intersection. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 444–453, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283430.

33 Nicholas JA Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 39(2):679–702, 2009. Preliminary version in FOCS 2006. doi:10.1137/070684008.

34 Nicholas James Alexander Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, 2008.

35 Chien-Chung Huang, Naonori Kakimura, and Naoyuki Kamiyama. Exact and approximation algorithms for weighted matroid intersection. *Mathematical Programming*, 177(1-2):85–112, 2019. Preliminary version in SODA 2016. doi:10.1007/s10107-018-1260-x.

36 Claude-Pierre Jeannerod, Clément Pernet, and Arne Storjohann. Rank-profile revealing gaussian elimination and the cup matrix decomposition. *Journal of Symbolic Computation*, 56:46–68, 2013. doi:10.1016/j.jsc.2013.04.004.

37 Tomohiro Koana and Magnus Wahlström. Faster algorithms on linear delta-matroids. In *Proceedings of 42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)*, volume 327, pages 62:1–62:19, 2025. doi:10.4230/LIPIcs.STACS.2025.62.

38 Eugene L Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, 1975. doi:10.1007/BF01681329.

39 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 1049–1065. IEEE, 2015. doi:10.1109/FOCS.2015.68.

40 Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009. Preliminary version in ICALP 2006. doi:10.1016/j.tcs.2009.07.027.

41 Kazuki Matoya and Taihei Oki. Pfaffian pairs and parities: Counting on linear matroid intersection and parity problems. *SIAM Journal on Discrete Mathematics*, 36(3):2121–2158, 2022. Preliminary version in IPCO 2021. doi:10.1137/21m1421751.

42 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. doi:10.1017/cbo9780511814075.

43 Harihar Narayanan, Huzur Saran, and Vijay V Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arborescences and edge-disjoint spanning trees. *SIAM Journal on Computing*, 23(2):387–397, 1994. Preliminary version in SODA 1992. doi:10.1137/S0097539791195245.

44 Huy L Nguyen. Fast greedy for linear matroids. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 516–524. SIAM, 2019. doi:10.1137/1.9781611975482.32.

45 Huy L Nguyen. A note on Cunningham's algorithm for matroid intersection. arXiv preprint arXiv:1904.04129, 2019. doi:10.48550/arXiv.1904.04129.

46 Taihei Oki and Tasuku Soma. Algebraic algorithms for fractional linear matroid parity via noncommutative rank. *SIAM Journal on Computing*, 54(1):134–162, 2025. Preliminary version in SODA 2023. doi:10.1137/22m1537096.

47 James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

48 Kent Quanrud. Adaptive sparsification for matroid intersection. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297, pages 118:1–118:20, 2024. doi:10.4230/LIPIcs.ICALP.2024.118.

49 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

50 Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

51 Maiko Shigeno and Satoru Iwata. A dual approximation approach to weighted matroid intersection. *Operations Research Letters*, 18(3):153–156, 1995. doi:10.1016/0167-6377(95)00047-X.

52 Arne Storjohann and Shiyun Yang. Linear independence oracles and applications to rectangular and low rank linear systems. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*, pages 381–388, 2014. doi:10.1145/2608628.2608673.

53 Tatsuya Terao. Deterministic $(2/3-\varepsilon)$-approximation of matroid intersection using nearly-linear independence-oracle queries. In *Proceedings of the 19th International Symposium on Algorithms and Data Structures (WADS 2025)*, volume 349, pages 50:1–50:18, 2025. doi:10.4230/LIPIcs.WADS.2025.50.

54 Tatsuya Terao. Faster approximate linear matroid intersection. arXiv preprint arXiv:2604.11725, 2026. doi:10.48550/arXiv.2604.11725.

55 Nobuaki Tomizawa and Masao Iri. An algorithm for determining the rank of a triple matrix product AXB with application to the problem of discerning the existence of the unique solution in a network. *Electronics and Communications in Japan (Scripta Electronica Japonica II)*, 57(11):50–57, 1974.

56 Ta-Wei Tu. Subquadratic weighted matroid intersection under rank oracles. In *Proceedings of the 33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248, pages 63:1–63:14, 2022. doi:10.4230/LIPIcs.ISAAC.2022.63.

57 Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Proceedings of Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 1–13. SIAM, 2021. doi:10.1137/1.9781611976496.1.

58 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 456–480. IEEE, 2019. doi:10.1109/FOCS.2019.00036.

59 Ying Xu and Harold N Gabow. Fast algorithms for transversal matroid intersection problems. In *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC 1994)*, pages 625–633. Springer, 1994. doi:10.1007/3-540-58325-4_231.

Maximum Independent Sets in Disk Graphs with Disks in Convex Position

Anastasiia Tkachenko
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

For a set $\mathcal{D}$ of disks in the plane, its disk graph $G(\mathcal{D})$ is the graph with vertex set $\mathcal{D}$, where two vertices are adjacent if and only if the corresponding disks intersect. Given a set $\mathcal{D}$ of n weighted disks, computing a maximum independent set of $G(\mathcal{D})$ is NP-hard. In this paper, we present an $O(n^3 \log n)$-time algorithm for this problem in a special setting in which the disks are in *convex position*, meaning that every disk appears on the convex hull of $\mathcal{D}$. This setting has been studied previously for disks of equal radius, for which an $O(n^{37/11})$-time algorithm was known. Our algorithm also works in the weighted case where disks have weights and the goal is to compute a maximum-weight independent set. As an application of our result, we obtain an $O(n^3 \log^2 n)$-time algorithm for the *dispersion problem* on a set of n disks in convex position: given an integer k, compute a subset of k disks that maximizes the minimum pairwise distance among all disks in the subset.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases disk graphs, independent sets, convex position, dispersion

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.40

Related Version *Full Version*: `https://arxiv.org/abs/2604.10828`

1 Introduction

An *independent set* in a graph is a subset of vertices with no edges between any two of them. The *maximum independent set problem* asks for an independent set of maximum cardinality. In weighted graphs, where each vertex is assigned a weight, the *maximum-weight independent set problem* seeks an independent set of maximum total weight.

In general graphs, these problems are computationally intractable [20]. This hardness has motivated research along two complementary directions. One direction focuses on designing approximation algorithms, which trade exactness for efficiency. The other aims to identify restricted settings, such as specific graph classes or additional structural constraints, under which exact polynomial-time algorithms become possible. Exact polynomial-time algorithms are known for a variety of restricted graph classes, including circular [5, 17], chordal [16], trapezoid [15], outerstring [7], and Burling graphs [28], where the additional structure can be exploited algorithmically.

In light of the above, geometric intersection graphs are of particular interest [1]. In these graphs, vertices represent geometric objects and edges encode their intersections. In this work, we focus on *disk graphs*, the intersection graphs of disks, which are among the most extensively studied geometric graph families. Beyond their intrinsic theoretical interest, disk graphs provide a natural abstraction for wireless and sensor networks, where connectivity is governed by transmission ranges that are commonly modeled as disks [6, 12, 24, 25].

© Anastasiia Tkachenko and Haitao Wang;
licensed under Creative Commons License CC-BY 4.0

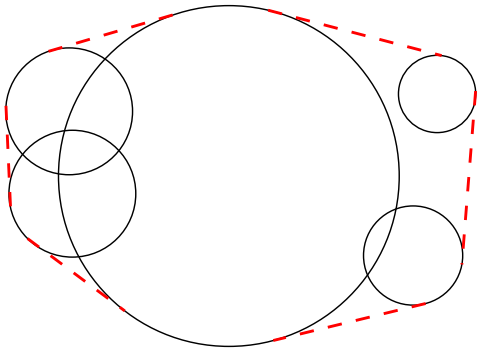


Figure 1 Illustrating a set of disks in convex position. The red dashed segments are on the boundary of the convex hull of all disks.

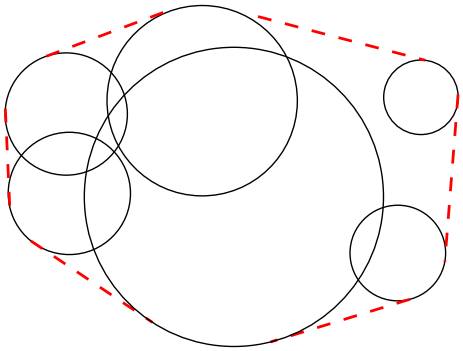


Figure 2 Illustrating a set of disks in strongly convex position. The red dashed segments are on the boundary of the convex hull of all disks.

Formally, let $\mathcal{D}$ be a set of disks in the plane. The *disk graph* $G(\mathcal{D})$ is the graph with vertex set $\mathcal{D}$, where two vertices are adjacent if and only if the corresponding disks intersect. Thus, a subset $\mathcal{D}' \subseteq \mathcal{D}$ is an independent set in $G(\mathcal{D})$ if and only if the disks in $\mathcal{D}'$ are pairwise disjoint. In addition, each disk is assigned a weight.

The maximum independent set problem in disk graphs remains NP-hard even when all disks have the same radius, in which case the graph is a *unit-disk graph* [12]. Many approximation algorithms for this problem have been developed; see, for example, [1, 9, 13, 14, 22, 23]. Beyond their algorithmic interest, independent sets in intersection graphs arise in several applications, such as map labeling in computational cartography [3]. For instance, in map labeling one is given a collection of candidate label regions and seeks a largest subset that can be placed without overlap; selecting such a subset is precisely a maximum independent set problem in the corresponding intersection graph.

These results naturally raise the question of whether, and to what extent, geometric structure can impose sufficient constraints to render otherwise intractable combinatorial problems tractable.

1.1 Our result

In this paper, we consider a special setting in which the disks in $\mathcal{D}$ are in *convex position*, meaning that every disk of $\mathcal{D}$ contributes to the boundary of the convex hull of $\mathcal{D}$ (see Figure 1). Under this assumption, we present an exact algorithm that computes a maximum-weight independent set in $G(\mathcal{D})$ in $O(n^3 \log n)$ time. This setting bridges the gap between the general two-dimensional problem and one-dimensional variants such as circular graphs, which admit near-linear-time algorithms [5].

The maximum-weight independent set problem under this convex position assumption has been studied previously, but only for the unit-disk case in which all disks have the same radius [30, 33]; in that setting, the previously best algorithm runs in $O(n^{37/11})$ time [33]. Our algorithm applies to disks of arbitrary radii and, moreover, improves the running time even in the unit-disk case.

We say that a set of disks is in *strongly convex position* if every disk has a single maximal disk arc appearing on the boundary of the convex hull of all disks (see Figure 2; but the disks in Figure 1 are not in strongly convex position). We solve the problem gradually as follows.

1. We first consider a very special case in which the disks of $\mathcal{D}$ are in strongly convex position and $\mathcal{D}$ has a maximum-weight independent set that is itself strongly convex; we refer to this as the *strongly-convex-input-and-output* case. Our algorithm for this case is based on dynamic programming and can be viewed as an extension of the unit-disk algorithm of [33] to disks of arbitrary radii. Moreover, we identify a new monotonicity property that allows our algorithm to run faster than the previous unit-disk algorithm [33]. Our algorithm for this case is presented in Section 3.
2. We then handle the *strongly-convex-input* case, in which $\mathcal{D}$ is strongly convex but does not necessarily admit a strongly convex maximum-weight independent set. We reduce this case to the first one by adding a set B of $O(n)$ auxiliary points, treated as special disks, such that (i) these special disks appear in any maximum-weight independent set of $\mathcal{D} \cup B$, and (ii) for any subset $\mathcal{D}' \subseteq \mathcal{D}$, the set $\mathcal{D}' \cup B$ is strongly convex. This case is discussed in Section 4.
3. Finally, we address the most general case in Section 5, where $\mathcal{D}$ is not necessarily strongly convex and does not necessarily admit a strongly convex maximum-weight independent set. We reduce this case to the second one by again adding a set of auxiliary points as special disks.

As an application of our independent set algorithm, we also solve a *dispersion problem* for a set of n disks in convex position: given an integer k, the goal is to find a subset of k disks that maximizes the minimum pairwise distance among all disks in the subset, where the distance between two disks is defined as the minimum distance between any point in one disk and any point in the other. Our algorithm for this problem runs in $O(n^3 \log^2 n)$ time and is presented in Section 6.

Due to the space limit, many proofs are omitted but can be found in the full paper.

1.2 Related work

Recent developments in geometric algorithms have demonstrated that the convex position constraint can serve as a powerful structural assumption. For a variety of classical problems, it enables exact polynomial-time algorithms, often by exploiting the inherent cyclic structure induced by convexity.

For example, the k-center problem is NP-hard for arbitrary point sets, but becomes polynomial-time solvable when the points are in convex position [11, 33]. Another recent result that exploits convex position concerns dominating sets in disk graphs [31, 33]. Note that for unit-disk graphs, the disk set is in convex position if and only if the centers of the disks are in convex position; this equivalence no longer holds when disks have different radii.

In the literature, related convexity assumptions have also been studied for disk graphs, including settings in which the disk centers are in convex position and settings in which the disks themselves are in convex position. For instance, Herrera and Pérez-Lantero [18] and Huemer and Pérez-Lantero [19] investigate combinatorial properties arising when the centers follow a prescribed convex pattern: each disk uses a distinct side of a convex polygon as its diameter, and its center is the midpoint of that side. Under the convex position assumption on the disks, it has also been reported that a maximum clique in the corresponding disk graph can be computed in polynomial time [8].

The convex position assumption has also been explored for classical problems that are already polynomial-time solvable in general, as it can simplify the structure and lead to faster algorithms. A well-known example is the linear-time algorithm for computing the Voronoi diagram of points in convex position [4]. Additional results for points in convex position can be found in [10, 18, 21, 27, 32].

A closely related problem is *dispersion* (also known as the *maximally separated set* problem [2]). Given a set of points in the plane and an integer k, the goal is to select k points that maximize their minimum pairwise distance; the problem is NP-hard in general [34]. When the points are in convex position, however, the problem becomes polynomial-time solvable [30, 33], and the previously best algorithm runs in $O(n^{37/11} \log n)$ time [33]. Our dispersion problem generalizes this setting from points to disks and our algorithm is even faster.

2 Preliminaries

In this section, we introduce notation and basic concepts used throughout the paper.

Let $\mathcal{D}$ denote a set of n disks D_i in the plane $\mathbb{R}^2$, for $1 \leq i \leq n$. For each disk D_i, let p_i be its center and let r_i be its radius. For any subset $\mathcal{D}' \subseteq \mathcal{D}$, we use $\mathcal{H}(\mathcal{D}')$ to denote the convex hull of $\mathcal{D}'$.

For any region R in the plane, let ∂R denote its boundary (e.g., $\partial\mathcal{H}(\mathcal{D})$ is the boundary of $\mathcal{H}(\mathcal{D})$ and ∂D is the boundary of a disk D). We say that the disks of $\mathcal{D}$ are in *convex position* if the boundary of every disk appears on $\partial\mathcal{H}(\mathcal{D})$, and they are in *strongly convex position* if the boundary of every disk has a single (maximal) arc appearing on $\partial\mathcal{H}(\mathcal{D})$. For simplicity, for any subset $\mathcal{D}' \subseteq \mathcal{D}$, if the disks of $\mathcal{D}'$ are in (strongly) convex position, we also say that $\mathcal{D}'$ is (strongly) convex.

For any two points p and q in the plane, let $\overline{pq}$ denote the line segment connecting them, and let $|pq|$ denote the Euclidean distance between p and q. For a point p with a *weight* $w(p)$, define

$$d_p(q) = |pq| + w(p)$$

for any point q in the plane. We call $d_p(q)$ the *weighted distance* between p and q.

For each disk $D_i \in \mathcal{D}$, we define the weight of its center p_i to be $-r_i$, i.e., the negative of the radius of D_i. Hence, for any point $q \in \mathbb{R}^2$, we have $d_{p_i}(q) = |p_iq| - r_i$. If q lies outside D_i, then $d_{p_i}(q)$ equals the minimum Euclidean distance between q and D_i. For this reason, we also refer to $d_{p_i}(q)$ as the *distance* between q and D_i (noting that this distance is negative when q lies in the interior of D_i).

Two disks $D_i, D_j \in \mathcal{D}$ are said to be *disjoint* if $|p_ip_j| > r_i + r_j$. A subset $\mathcal{D}' \subseteq \mathcal{D}$ is an *independent set* if the disks in $\mathcal{D}'$ are pairwise disjoint.

Throughout the paper, we assume that the disks of $\mathcal{D}$ are in convex position (but not necessarily in strongly convex position), and that each disk D_i is assigned a weight W_i. Our goal is to compute a maximum-weight independent set of $\mathcal{D}$, that is, an independent set of disks whose total weight is maximized. Since disks with non-positive weight can be ignored, we assume $W_i > 0$ for all D_i. For any subset $\mathcal{D}' \subseteq \mathcal{D}$, we use $W(\mathcal{D}')$ to denote the sum of the weights of the disks in $\mathcal{D}'$.

For any disk D and any two points $a, b \in \partial D$, we use $\partial_{[a,b]}D$ to denote the portion of ∂D from a to b in counterclockwise order. Similarly, for any subset $\mathcal{D}' \subseteq \mathcal{D}$ and any two points $a, b \in \partial\mathcal{H}(\mathcal{D}')$, we use $\partial_{[a,b]}\mathcal{H}(\mathcal{D}')$ to denote the portion of $\partial\mathcal{H}(\mathcal{D}')$ from a to b in counterclockwise order.

Finally, for ease of exposition, we assume general position: no point in the plane is equidistant from four disks of $\mathcal{D}$, and no line is tangent to three disks.

3 The strongly-convex-input-and-output case

We consider the *strongly-convex-input-and-output* case, in which $\mathcal{D}$ is strongly convex and admits a maximum-weight independent set that is itself strongly convex. To simplify the discussion, we assume that each disk in $\mathcal{D}$ has positive radius (i.e., no disk degenerates to a point). The same algorithm applies when some disks are points, but handling such degeneracies would require more technical discussion. We present an algorithm that computes a maximum-weight independent set in $O(n^3 \log n)$ time.

We begin with the following lemma (see the full paper for the proof).

▶ **Lemma 1.** *For any subset of three disks $\{D_i, D_j, D_k\} \subseteq \mathcal{D}$ that forms a strongly convex independent set, there exists a unique point $v_{ijk} \in \mathbb{R}^2$ that is equidistant from the three disks. Furthermore, v_{ijk} lies outside each of the three disks.*

In light of the above lemma, for any subset of three disks $\{D_i, D_j, D_k\} \subseteq \mathcal{D}$ that forms a strongly convex independent set, there exists a disk centered at v_{ijk} that is tangent to all three disks. We use D_{ijk} to denote this disk, and let r_{ijk} denote its radius. Note that $r_{ijk} = d_{p_i}(v_{ijk}) = d_{p_j}(v_{ijk}) = d_{p_k}(v_{ijk})$.

Since $\mathcal{D}$ is strongly convex, each disk in $\mathcal{D}$ contributes a single arc to the convex hull boundary $\partial\mathcal{H}(\mathcal{D})$, and $\partial\mathcal{H}(\mathcal{D})$ consists of an alternating sequence of disk arcs and line segments, where each line segment is an outer common tangent of two adjacent disks along $\partial\mathcal{H}(\mathcal{D})$. Let

$$\mathcal{D} = \langle D_1, D_2, \ldots, D_n \rangle$$

be a cyclic list of the disks whose arcs appear on $\partial\mathcal{H}(\mathcal{D})$, ordered counterclockwise. For any $i \neq j$, let $\mathcal{D}(i, j)$ denote the sublist of $\mathcal{D}$ from D_i to D_j in counterclockwise order along $\partial\mathcal{H}(\mathcal{D})$ excluding D_i and D_j. That is,

$$\mathcal{D}(i,j) = \begin{cases} \langle D_{i+1}, D_{i+2}, \ldots, D_{j-1} \rangle, & \text{if } i < j, \\ \langle D_{i+1}, D_{i+2}, \ldots, D_n, D_1, \ldots, D_{j-1} \rangle, & \text{if } i > j. \end{cases}$$

In addition, we have the following lemma, which will be used later (see the full paper for the proof).

▶ **Lemma 2.** *For any subset $\mathcal{D}' \subseteq \mathcal{D}$ that is strongly convex, the counterclockwise order of the disks of $\mathcal{D}'$ along $\partial\mathcal{H}(\mathcal{D}')$ is consistent with the index order of $\mathcal{D}$.*

Our algorithm is based on dynamic programming. In Section 3.1, we describe the algorithm, define the subproblems of the dynamic program, and present the dependency relations. We prove the correctness of the algorithm in Section 3.2, and discuss how to implement it efficiently in Section 3.3.

3.1 Algorithm description

For three disk indices i, j, k, we call an ordered triple (i, j, k) a *canonical triple* if D_i, D_j, and D_k appear on $\partial\mathcal{H}(\mathcal{D})$ in counterclockwise order and $\{D_i, D_j, D_k\}$ is a strongly convex independent set.

For a canonical triple (i, j, k), since $\{D_i, D_j, D_k\}$ is a strongly convex independent set, the disk D_{ijk} exists by Lemma 1. We define $\mathcal{D}_k(i, j)$ as the set of disks $D \in \mathcal{D}(i, j)$ such that D is disjoint from D_i, D_j, and D_{ijk}. For convenience, for any two disk indices i and j such that D_i is disjoint from D_j, we also treat $(i, j, 0)$ as a canonical triple, and define $\mathcal{D}_0(i, j)$ to be the set of disks $D \in \mathcal{D}(i, j)$ that are disjoint from both D_i and D_j.

For a canonical triple (i, j, k), including the case $k = 0$, let $\mathcal{D}'_k(i, j)$ denote the subset of disks $D \in \mathcal{D}_k(i, j)$ such that $\{D, D_i, D_j\}$ is strongly convex.

Subproblems. For a canonical triple (i, j, k), including the case $k = 0$, we define $f(i, j, k)$ to be the maximum total weight of any subset $\mathcal{D}' \subseteq \mathcal{D}_k(i, j)$ such that $\mathcal{D}' \cup \{D_i, D_j\}$ forms a strongly convex independent set. If no such subset $\mathcal{D}'$ exists, we set $f(i, j, k) = 0$.

The following lemma gives the dependency relation among the subproblems. Since the proof is lengthy and technical, we defer it to Section 3.2.

▶ **Lemma 3.** *For any canonical triple (i, j, k), including the case $k = 0$, the following holds:*

$$f(i,j,k) = \begin{cases} \max\limits_{D_l \in \mathcal{D}'_k(i,j)} \big(f(i,l,j) + f(l,j,i) + W_l\big), & \text{if } \mathcal{D}'_k(i,j) \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

We define W^* to be the optimal objective value of our problem, that is, W^* equals the total weight of a maximum-weight independent set of $\mathcal{D}$. For convenience, if disks D_i and D_j intersect, we define $f(i, j, 0) = -W_i - W_j$. The next lemma shows that the optimal value W^* can be computed using the values $f(i, j, 0)$ for all pairs (i, j) (see the full paper for the proof). This explains our choice of $f(i, j, k)$ as the subproblems in the dynamic program.

▶ **Lemma 4.** $W^* = \max_{1 \le i,j \le n}(f(i, j, 0) + W_i + W_j)$.

3.2 Algorithm correctness: Proving Lemma 3

We now prove Lemma 3. Consider a canonical triple (i, j, k). Without loss of generality, we assume that the centers p_i and p_j of disks D_i and D_j lie on the same horizontal line, with p_i to the left of p_j. This assumption allows us to clearly distinguish the upper and lower (outer) common tangents of D_i and D_j.

If $k \neq 0$, then by definition $\{D_i, D_j, D_k\}$ is a strongly convex independent set, and the disk D_{ijk} exists by Lemma 1. In the special case $k = 0$, we let D_{ijk} denote the upper halfplane bounded by the upper common tangent line of D_i and D_j. Accordingly, its center v_{ijk} is interpreted as a point with infinitely large y-coordinate and its radius r_{ijk} is taken to be infinite. With this convention, the arguments below apply uniformly to both cases $k = 0$ and $k \neq 0$.

If $\mathcal{D}'_k(i, j) = \emptyset$, then $f(i, j, k) = 0$ by Corollary 6, which is proved using Lemma 5.

▶ **Lemma 5.** *For any subset $\mathcal{D}' \subseteq \mathcal{D}_k(i, j)$ such that $\mathcal{D}' \cup \{D_i, D_j\}$ is a strongly convex independent set, the set $\mathcal{D}'$ must contain a disk D for which $\{D, D_i, D_j\}$ is a strongly convex independent set.*

Proof. Let $S = \mathcal{D}' \cup \{D_i, D_j\}$. Since S is strongly convex, by Lemma 2, the counterclockwise order of the disks of S along $\partial\mathcal{H}(S)$ is consistent with the index order of $\mathcal{D}$. Because $\mathcal{D}' \subseteq \mathcal{D}_k(i, j) \subseteq \mathcal{D}(i, j)$, the arcs of D_i and D_j on $\partial\mathcal{H}(S)$ must be adjacent; that is, they are connected by a line segment on $\partial\mathcal{H}(S)$, and this segment is a common tangent of D_i and D_j. Since we have assumed that the centers p_i and p_j lie on a horizontal line, this segment must be the upper common tangent of D_i and D_j.

The above argument shows that the upper common tangent of D_i and D_j must be an edge of $\mathcal{H}(S)$. Since S is strongly convex, the lower common tangent of D_i and D_j cannot be an edge of $\mathcal{H}(S)$ (since otherwise it would not be possible that each disk of S has a single arc appearing on $\partial\mathcal{H}(S)$, contradicting that S is strongly convex). This implies that at least one disk $D \in \mathcal{D}'$ must have a point lying strictly below the supporting line of the lower common tangent of D_i and D_j (see Figure 3). Consequently, the set $\{D, D_i, D_j\}$ is strongly convex. Since S is an independent set, $\{D, D_i, D_j\}$ is also an independent set, completing the proof. ◀

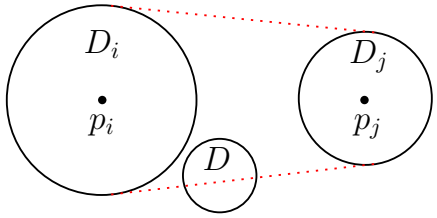


Figure 3 Illustrating the proof of Lemma 5.

▶ **Corollary 6.** *If $\mathcal{D}'_k(i,j) = \emptyset$, then $\mathcal{D}_k(i,j)$ does not contain any subset $\mathcal{D}'$ such that $\mathcal{D}' \cup \{D_i, D_j\}$ is a strongly convex independent set. Consequently, $f(i,j,k) = 0$.*

Proof. Suppose, for contradiction, that $\mathcal{D}_k(i,j)$ contains a subset $\mathcal{D}'$ such that $\mathcal{D}' \cup \{D_i, D_j\}$ is a strongly convex independent set. By Lemma 5, the set $\mathcal{D}'$ must contain a disk D such that $\{D, D_i, D_j\}$ is a strongly convex independent set. Since $\mathcal{D}' \subseteq \mathcal{D}_k(i,j)$, it follows that $\mathcal{D}_k(i,j)$ contains such a disk D, contradicting the assumption that $\mathcal{D}'_k(i,j) = \emptyset$. The corollary follows. ◀

In the following, we assume that $\mathcal{D}'_k(i,j) \neq \emptyset$. We will prove Lemma 3 in two parts: a *forward* direction and a *backward* direction.

Let S be an optimal solution defining $f(i,j,k)$; that is, $S \subseteq \mathcal{D}_k(i,j)$, the set $S \cup \{D_i, D_j\}$ is a strongly convex independent set, and $f(i,j,k) = W(S)$. In the forward direction, we show that S contains a disk $D_l \in \mathcal{D}'_k(i,j)$ such that S can be partitioned as $S = S_1 \cup S_2 \cup \{D_l\}$, where $S_1 \subseteq \mathcal{D}_j(i,l)$ and $S_2 \subseteq \mathcal{D}_i(l,j)$, and both $S_1 \cup \{D_i, D_l\}$ and $S_2 \cup \{D_l, D_j\}$ are strongly convex independent sets. This implies that $f(i,j,k) \leq \max_{D_l \in \mathcal{D}'_k(i,j)} \big(f(i,l,j) + f(l,j,i) + W_l\big)$.

In the backward direction, we show that for any disk $D_l \in \mathcal{D}'_k(i,j)$ and any sets $S_1 \subseteq \mathcal{D}_j(i,l)$ and $S_2 \subseteq \mathcal{D}_i(l,j)$ such that both $S_1 \cup \{D_i, D_l\}$ and $S_2 \cup \{D_l, D_j\}$ are strongly convex independent sets, the set $S_1 \cup S_2 \cup \{D_l\}$ is contained in $\mathcal{D}_k(i,j)$ and $S_1 \cup S_2 \cup \{D_l, D_i, D_j\}$ is a strongly convex independent set. This implies that $f(i,j,k) \geq \max_{D_l \in \mathcal{D}'_k(i,j)} \big(f(i,l,j) + f(l,j,i) + W_l\big)$.

Together, the two directions establish Lemma 3.

3.2.1 The forward direction

Let S be an optimal solution defining $f(i,j,k)$; that is, $S \subseteq \mathcal{D}_k(i,j)$, the set $S \cup \{D_i, D_j\}$ is a strongly convex independent set, and $f(i,j,k) = W(S)$.

Let γ_{ij} denote the *bisector* of D_i and D_j, which consists of all points q that are equidistant from D_i and D_j. It is well known that γ_{ij} is a branch of a hyperbola (degenerating to a line when $r_i = r_j$) [29]. Moreover, since we assume that the centers of D_i and D_j lie on a horizontal line, γ_{ij} is y-monotone.

A total order $\prec$ on γ_{ij}. For any two points $a, b \in \gamma_{ij}$, we define a total order $a \prec b$ if a has a smaller y-coordinate than b. The following lemma will be frequently used.

▶ **Lemma 7.** *Suppose $D_h \in \mathcal{D}$ is a disk such that $\{D_h, D_i, D_j\}$ is a strongly convex independent set, and let v be a point on γ_{ij}. If $D_h \in \mathcal{D}(i,j)$, then $v_{ijh} \prec v$ if and only if $d_{p_i}(v) < d_{p_h}(v)$; see Figure 4. If $D_h \in \mathcal{D}(j,i)$, then $v \prec v_{ijh}$ if and only if $d_{p_i}(v) < d_{p_h}(v)$.*

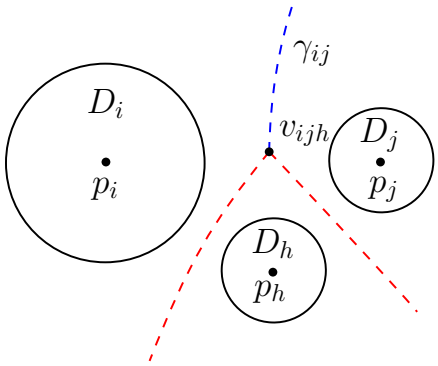


Figure 4 Illustrating Lemma 7. The three dashed curves (two red and one blue) are edges of Voronoi diagram of the three disks. The blue curve belongs to γ_{ij}.

Proof. We prove only the case where $D_h \in \mathcal{D}(i,j)$; the other case can be handled analogously.

Let $\mathcal{D}' = \{D_h, D_i, D_j\}$. Consider the additively-weighted Voronoi diagram $\mathrm{Vor}(\mathcal{D}')$ of the three weighted centers of $\mathcal{D}'$. By Lemma 1, the diagram has a unique Voronoi vertex, namely v_{ijh}. Note that $v_{ijh} \in \gamma_{ij}$. Since $\mathcal{D}'$ is strongly convex, Lemma 2 implies that D_i, D_h, and D_j appear on $\partial\mathcal{H}(\mathcal{D}')$ in counterclockwise order.

Consequently, the portion of the bisector γ_{ij} above v_{ijh} belongs to the common boundary between the Voronoi cells of p_i and p_j, while the portion of γ_{ij} below v_{ijh} lies entirely within the Voronoi cell of p_h. Therefore, for any point $v \in \gamma_{ij}$, we have $v_{ijh} \prec v$ if and only if $d_{p_i}(v) < d_{p_h}(v)$. ◀

By Lemma 5, the set S contains at least one disk D such that $\{D, D_i, D_j\}$ is strongly convex. For each disk $D_h \in S$ with this property, the point v_{ijh} exists by Lemma 1. Among all such disks $D_h \in S$, let D_l be the one for which v_{ijl} is largest under the order $\prec$. Due to the general position assumption, D_l is unique. Define $S_1 = S \cap \mathcal{D}(i,l)$ and $S_2 = S \cap \mathcal{D}(l,j)$. Since $S \subseteq \mathcal{D}_k(i,j)$, the sets S_1, S_2, and $\{D_l\}$ form a partition of S. By definition, $D_l \in \mathcal{D}'_k(i,j)$.

To complete the proof of the forward direction, it remains to show that $S_1 \subseteq \mathcal{D}_j(i,l)$, $S_2 \subseteq \mathcal{D}_i(l,j)$, and that both $S_1 \cup \{D_i, D_l\}$ and $S_2 \cup \{D_l, D_j\}$ are strongly convex independent sets. These statements are proved in the next two lemmas.

▶ **Lemma 8.** *$S_1 \subseteq \mathcal{D}_j(i,l)$ and $S_2 \subseteq \mathcal{D}_i(l,j)$.*

Proof. We only prove $S_1 \subseteq \mathcal{D}_j(i,l)$; the other case can be proved analogously.

Consider a disk $D_h \in S_1$. Our goal is to prove $D_h \in \mathcal{D}_j(i,l)$. To this end, we need to show that D_h is disjoint from the three disks D_i, D_l, and D_{ijl}. Since $S_1 \subseteq S$, $D_l \in S$, and $S \cup \{D_i, D_j\}$ is an independent set, we know that D_h is disjoint from both D_i and D_l. Hence, it remains to prove that D_h is disjoint from D_{ijl}, or equivalently, $d_{p_h}(v_{ijl}) > r_{ijl}$; recall that r_{ijl} denotes the radius of D_{ijl} and $r_{ijl} = d_{p_i}(v_{ijl})$. Hence, it suffices to show that $d_{p_h}(v_{ijl}) > d_{p_i}(v_{ijl})$. Let $S' = \{D_h, D_i, D_j\}$. Depending on whether S' is strongly convex, we distinguish two cases.

If S' is strongly convex, then by Lemma 1, v_{ijh} exists. By the definition of D_l, we have $v_{ijh} \prec v_{ijl}$. Consequently, applying Lemma 7 (with $v = v_{ijl}$) obtains $d_{p_i}(v_{ijl}) < d_{p_h}(v_{ijl})$.

We now consider the case in which S' is not strongly convex. First of all, since $\mathcal{D}$ is strongly convex and $S' \subseteq \mathcal{D}$, S' must be convex. Since $S \cup \{D_i, D_j\}$ is strongly convex and $S \subseteq \mathcal{D}(i,j)$, as argued in the proof of Lemma 5, the upper common tangent of D_i and D_j

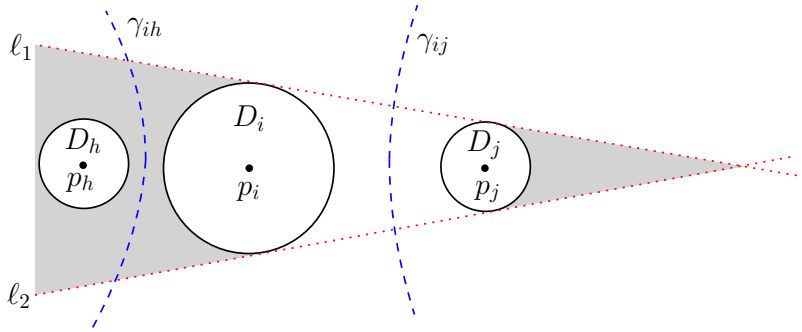


Figure 5 The disk D_h can only be in one of the two shaded regions. The curve γ_{ij} is the bisector of D_i and D_j, and γ_{ih} is the bisector of D_i and D_h. The two bisectors are the only two edges of the Voronoi diagram of the three disks.

must be on $\partial\mathcal{H}(S \cup \{D_i, D_j\})$. Since $S' \subseteq S \cup \{D_i, D_j\}$, the upper common tangent of D_i and D_j must be on $\partial\mathcal{H}(S')$. Let ℓ_1 and ℓ_2 be the upper and lower common tangent lines of D_i and D_j, respectively.

Since S' is independent and convex but not strongly convex, D_h must be either in the region to the left of D_i, below ℓ_1, and above ℓ_2, or in the region to the right of D_j, below ℓ_1, and above ℓ_2 (see Figure 5). In either case, if we consider the Voronoi diagram $\mathrm{Vor}(S')$ of the disks of S', then the bisector γ_{ij} is a common edge of the Voronoi cells of D_i and D_j. This implies that for any point $v \in \gamma_{ij}$, it holds that $d_{p_i}(v) < d_{p_h}(v)$. Since $v_{ijl} \in \gamma_{ij}$, we obtain $d_{p_i}(v_{ijl}) < d_{p_h}(v_{ijl})$. ◀

▶ **Lemma 9.** *Both $S_1 \cup \{D_i, D_l\}$ and $S_2 \cup \{D_l, D_j\}$ are strongly convex independent sets.*

Proof. See the full paper for the proof. ◀

3.2.2 The backward direction

Consider a disk $D_l \in \mathcal{D}'_k(i,j)$ together with $S_1 \subseteq \mathcal{D}_j(i,l)$ and $S_2 \subseteq \mathcal{D}_i(l,j)$ such that both $S_1 \cup \{D_i, D_l\}$ and $S_2 \cup \{D_l, D_j\}$ are strongly convex independent sets. Our goal is to show that $S_1 \cup S_2 \cup \{D_l\} \subseteq \mathcal{D}_k(i,j)$ and $S_1 \cup S_2 \cup \{D_l, D_i, D_j\}$ is a strongly convex independent set. Let $S = S_1 \cup S_2 \cup \{D_l\}$.

We begin with arguing $S \subseteq \mathcal{D}_k(i,j)$. Let D_h be a disk of S. If D_h is D_l, then since $D_l \in \mathcal{D}'_k(i,j)$ and $\mathcal{D}'_k(i,j) \subseteq \mathcal{D}_k(i,j)$, we have $D_l \in \mathcal{D}_k(i,j)$. Otherwise, D_h is either in S_1 or in S_2. We assume that $D_h \in S_1$ (the other case can be handled analogously). In the following, we argue that $D_h \in \mathcal{D}_k(i,j)$. To this end, since $D_h \in S_1 \subseteq \mathcal{D}_j(i,l) \subseteq \mathcal{D}(i,l) \subseteq \mathcal{D}(i,j)$, we need to show that D_h is disjoint from D_i, D_j, and D_{ijk}.

First of all, since $D_h \in S_1 \subseteq \mathcal{D}_j(i,l)$, D_h must be disjoint from D_i. In the next two lemmas, we prove that D_h is disjoint from D_{ijk} and D_j, respectively.

▶ **Lemma 10.** *D_h is disjoint from D_{ijk}.*

Proof. It suffices to show that $d_{p_h}(v_{ijk}) > r_{ijk}$, or equivalently, $d_{p_h}(v_{ijk}) > d_{p_i}(v_{ijk})$ as $r_{ijk} = d_{p_i}(v_{ijk})$.

Since $D_l \in \mathcal{D}'_k(i,j)$, $\{D_l, D_i, D_j\}$ is a strongly convex independent set and thus v_{ijl} exists by Lemma 1. Clearly, both v_{ijk} and v_{ijl} are on the bisector γ_{ij} of D_i and D_j. Depending on whether $\{D_i, D_j, D_h\}$ is strongly convex, we distinguish two cases.

- If $\{D_i, D_j, D_h\}$ is not strongly convex, then similarly to the proof of Lemma 8, we can obtain that for any point $v \in \gamma_{ij}$, $d_{p_i}(v) < d_{p_h}(v)$. As $v_{ijk} \in \gamma_{ij}$, it follows that $d_{p_i}(v_{ijk}) < d_{p_h}(v_{ijk})$.
- If $\{D_i, D_j, D_h\}$ is strongly convex, then v_{ijh} exists by Lemma 1. We claim that $v_{ijh} \prec v_{ijl}$. To see this, since $D_h \in S_1 \subseteq \mathcal{D}_j(i,l)$, D_h is disjoint from D_{ijl}, or equivalently, $d_{p_i}(v_{ijl}) < d_{p_h}(v_{ijl})$. By Lemma 7 (with $v = v_{ijl}$), we obtain $v_{ijh} \prec v_{ijl}$.
  Since $D_l \in \mathcal{D}_k(i,j)$, following the similar argument as above, we can obtain $v_{ijl} \prec v_{ijk}$. Hence, we have $v_{ijh} \prec v_{ijk}$. Consequently, applying Lemma 7 again (with $v = v_{ijk}$) leads to $d_{p_i}(v_{ijk}) < d_{p_h}(v_{ijk})$.

The lemma thus follows. ◀

▶ **Lemma 11.** *D_h is disjoint from D_j.*

Proof. See the full paper for the proof. ◀

The above proves that $S \subseteq \mathcal{D}_k(i,j)$. The following lemma finally proves that $S \cup \{D_i, D_j\}$ is a strongly convex independent set (see the full paper for the proof). This completes the proof of the backward direction.

▶ **Lemma 12.** *$S \cup \{D_i, D_j\}$ is a strongly convex independent set.*

3.3 Algorithm implementation

We now discuss how to implement the algorithm. We focus on computing W^*. By slightly modifying the algorithm using a standard backtracking technique, an actual maximum-weight independent set can also be recovered within asymptotically the same time complexity.

To compute W^*, Lemma 4 implies that it suffices to compute $f(i,j,k)$ for all canonical triples (i,j,k), including the case $k = 0$. To this end, by Equation (1), we must determine an order for solving the subproblems such that, when computing a value $f(i,j,k)$, all values $f(i,l,j)$ and $f(l,j,i)$ for $D_l \in \mathcal{D}'_k(i,j)$ have already been computed.

We process pairs (i,j) in increasing order of j, and for each fixed j in decreasing order of i: for $j = 2, \ldots, n$, we consider $i = j-1, j-2, \ldots, 1$ in this order. If $D_i \cap D_j \neq \emptyset$, we simply set $f(i,j,0) = -W_i - W_j$. Otherwise, we compute $f(i,j,0)$ using Equation (1). Next, for each disk $D_k \in \mathcal{D}(j,i)$ such that (i,j,k) is canonical, we compute $f(i,j,k)$ using Equation (1). With this ordering, every f-value appearing on the right-hand side of Equation (1) involves indices that lie between i and j in cyclic order, and hence has already been computed.

Using Equation (1), each subproblem $f(i,j,k)$ can be computed in $O(n)$ time by scanning all disks $D_l \in \mathcal{D}'_k(i,j)$. Since there are $O(n^3)$ subproblems, the total running time of the algorithm is $O(n^4)$.

An improved solution. We now improve the running time to $O(n^3 \log n)$. More specifically, we show that for any fixed pair (i,j), the $O(n)$ subproblems $f(i,j,k)$ for all relevant indices k can be computed in a total of $O(n \log n)$ time. This improvement is achieved by exploiting a monotonicity property of the sets $\mathcal{D}'_k(i,j)$ with respect to a suitable ordering of the indices k.

Consider a pair (i,j) such that D_i does not intersect D_j, and let (i,j,k) be a canonical triple. For each disk $D_l \in \mathcal{D}'_k(i,j)$, we define its *cost* as

$$cost(D_l) = f(i,l,j) + f(l,j,i) + W_l.$$

Define $\mathcal{D}'(i,j)$ as the set of disks $D \in \mathcal{D}(i,j)$ such that $\{D, D_i, D_j\}$ forms a strongly convex independent set. Then $\mathcal{D}'_k(i,j)$ consists precisely of those disks in $\mathcal{D}'(i,j)$ that are disjoint from the disk D_{ijk}. By Equation (1), computing $f(i,j,k)$ reduces to finding the disk of maximum cost among all disks in $\mathcal{D}'(i,j)$ that are disjoint from D_{ijk}.

We process a fixed pair (i,j) using the following approach, which computes $f(i,j,k)$ for all $k \in K$, where K denotes the set of indices k such that (i,j,k) is a canonical triple. The total running time of this procedure is $O(n \log n)$.

Without loss of generality, we assume that the centers p_i and p_j lie on the same horizontal line, with p_i to the left of p_j. We first compute the set K and the set $\mathcal{D}'(i,j)$, which can be done in $O(n)$ time.

For each $k \in K$, since (i,j,k) is a canonical triple, the set $\{D_i, D_j, D_k\}$ is strongly convex, and thus the vertex v_{ijk} exists by Lemma 1. The following lemma, which establishes a monotonicity property of the sets $\mathcal{D}'_k(i,j)$, is central to our approach.

▶ **Lemma 13.** *Consider any $k, k' \in K$ with $v_{ijk} \prec v_{ijk'}$. For any disk $D_h \in \mathcal{D}'(i,j)$, if D_h is disjoint from D_{ijk}, then D_h is also disjoint from $D_{ijk'}$. Consequently, $\mathcal{D}'_k(i,j) \subseteq \mathcal{D}'_{k'}(i,j)$.*

Proof. Consider a disk $D_h \in \mathcal{D}'(i,j)$ such that D_h is disjoint from D_{ijk}. Our goal is to show that D_h is also disjoint from $D_{ijk'}$, or equivalently, $d_{p_h}(v_{ijk'}) > r_{ijk'}$.

Since D_h is disjoint from D_{ijk}, we have $d_{p_h}(v_{ijk}) > r_{ijk}$. As $D_h \in \mathcal{D}'(i,j)$, $\{D_h, D_i, D_j\}$ is a strongly convex independent set. Therefore, v_{ijh} exists by Lemma 1. Since $d_{p_h}(v_{ijk}) > r_{ijk} = d_{p_i}(v_{ijk})$, applying Lemma 7 (with $v = v_{ijk}$) gives $v_{ijh} \prec v_{ijk}$.

On the other hand, since $v_{ijk} \prec v_{ijk'}$, we have $v_{ijh} \prec v_{ijk'}$. Consequently, applying Lemma 7 (with $v = v_{ijk'}$) gives $d_{p_h}(v_{ijk'}) > d_{p_i}(v_{ijk'}) = r_{ijk'}$. This proves the lemma. ◀

In light of Lemma 13, our algorithm proceeds as follows. We sort all indices $k \in K$ according to the $\prec$ order on γ_{ij}; let $k_1, k_2, \ldots, k_g$ be the resulting sequence, where $g = |K|$. Since $g \leq n$, this sorting step takes $O(n \log n)$ time. Next, for each disk $D_h \in \mathcal{D}'(i,j)$, we compute the smallest index $t \in [1,g]$ such that D_h is disjoint from D_{ijk_t}. By Lemma 13, this can be done in $O(\log n)$ time via binary search on the sequence $k_1, k_2, \ldots, k_g$. We then assign this value t as a *label* to D_h.

Recall that our goal is to compute $f(i,j,k_t)$ for all $t = 1, 2, \ldots, g$, where each $f(i,j,k_t)$ is equal to the maximum cost among all disks disjoint from D_{ijk_t}. We process the indices in increasing order of t. For convenience, let $f(i,j,k_0) = 0$. For each $t \geq 1$, we initialize $f(i,j,k_t) = f(i,j,k_{t-1})$, and then examine all disks $D_l \in \mathcal{D}'(i,j)$ whose label is exactly t. For each such disk, if $f(i,j,k_t) < cost(D_l)$, we update $f(i,j,k_t) = cost(D_l)$. In this manner, all values $f(i,j,k_t)$ for $t = 1, 2, \ldots, g$ can be computed in a total of $O(n)$ time.

This completes the processing for a fixed pair (i,j), with a total running time of $O(n \log n)$. Since there are $O(n^2)$ choices of pairs (i,j), the overall running time of the algorithm is $O(n^3 \log n)$. The following lemma summarizes our result.

▶ **Lemma 14.** *Given a set $\mathcal{D}$ of n weighted disks in the plane such that the disks are in strongly convex position and $\mathcal{D}$ admits a maximum-weight independent set that is strongly convex, a maximum-weight independent set of $\mathcal{D}$ can be computed in $O(n^3 \log n)$ time.*

4 The strongly-convex-input case

In this section, we consider the strongly-convex-input case, in which the input disks of $\mathcal{D}$ are in strongly convex position but $\mathcal{D}$ does not necessarily admit a maximum-weight independent set that is strongly convex (see Figure 6).

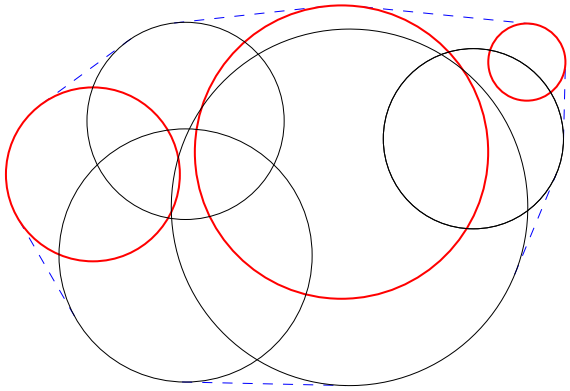


Figure 6 Illustrating an example where the input set of disks is strongly convex, but the only maximum-weight independent set (shown in red) is not strongly convex (assuming the weight of each disk is 1).

To handle this case, we reduce it to the setting in Section 3. To this end, we construct a set B of $O(n)$ *auxiliary points* satisfying the following two properties:

1. No point of B is contained in any disk of $\mathcal{D}$.
2. For any subset $\mathcal{D}' \subseteq \mathcal{D}$, $\mathcal{D}' \cup B$ is strongly convex (in particular, $\mathcal{D} \cup B$ is strongly convex).

Each point of B is treated as a special disk and assigned a weight of 1. Let $S = \mathcal{D} \cup B$. By the first property, any maximum-weight independent set of S must include all points of B. Therefore, if S' is a maximum-weight independent set of S, then $S' \setminus B$ is a maximum-weight independent set of $\mathcal{D}$. In this way, computing a maximum-weight independent set of $\mathcal{D}$ reduces to computing a maximum-weight independent set of S. Moreover, by the second property, S is strongly convex and admits a maximum-weight independent set that is strongly convex. Consequently, computing a maximum-weight independent set of S becomes an instance of the case in Section 3 and can be solved in $O(n^3 \log n)$ time by Lemma 14.

In the remainder of this section, we focus on constructing a set B of $O(n)$ auxiliary points satisfying the above two properties, and show that such a set can be computed in $O(n \log n)$ time.

To simplify the discussion, we assume that each disk in $\mathcal{D}$ has positive radius. The same approach extends to the case where some disks degenerate to points, although the exposition becomes more involved. We also assume that $n \geq 3$, since smaller instances can be handled directly in $O(1)$ time. Because $\mathcal{D}$ is strongly convex, each disk contributes exactly one arc to the boundary $\partial\mathcal{H}(\mathcal{D})$. As in Section 3, let $\mathcal{D} = \langle D_1, \ldots, D_n \rangle$ denote the cyclic order of disks appearing on $\partial\mathcal{H}(\mathcal{D})$ in counterclockwise order.

4.1 Defining B

For each disk $D_i \in \mathcal{D}$, D_i contributes a single arc α_i to the boundary $\partial\mathcal{H}(\mathcal{D})$. Let a_i denote the midpoint of α_i, and let ℓ_i be the line through a_i that is tangent to D_i. Thus, ℓ_i is also tangent to $\mathcal{H}(\mathcal{D})$ at a_i. We orient ℓ_i so that D_i lies to the left of ℓ_i (see Figure 7).

For each pair of consecutive disks D_i and D_{i+1} along $\partial\mathcal{H}(\mathcal{D})$ (if $i = n$, then we let $i + 1$ take modulo n, which equals 1), we proceed as follows. Let $\rho_{i,i+1}$ be the directed common tangent line of D_i and D_{i+1} such that both disks lie to the left of $\rho_{i,i+1}$. By the definition of the disk order along $\partial\mathcal{H}(\mathcal{D})$, the convex hull $\mathcal{H}(\mathcal{D})$ lies entirely to the left of $\rho_{i,i+1}$, and the segment of $\rho_{i,i+1}$ between the two tangency points on D_i and D_{i+1} forms an edge of $\mathcal{H}(\mathcal{D})$.

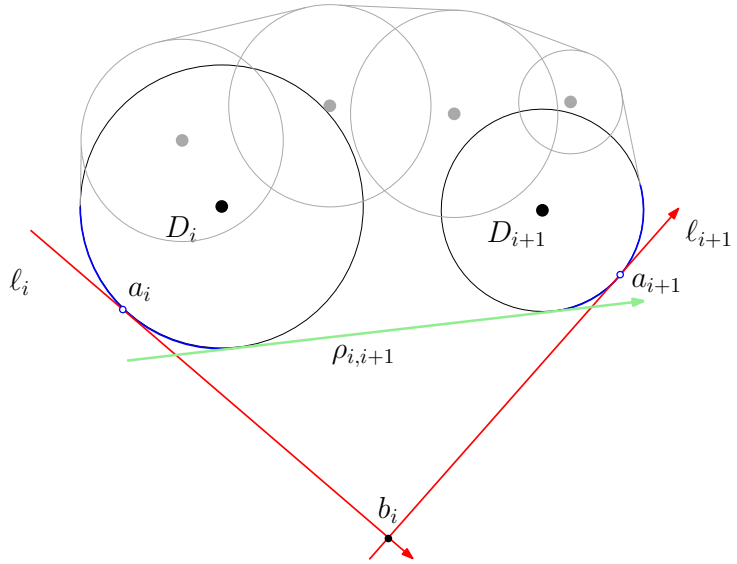


Figure 7 Illustrating the definition of b_i.

Since ℓ_i is tangent to D_i at the midpoint of α_i, the line ℓ_i must intersect $\rho_{i,i+1}$. Similarly, ℓ_{i+1} intersects $\rho_{i,i+1}$. Moreover, because $n \geq 3$, the lines ℓ_i and ℓ_{i+1} intersect at a point b_i that lies to the right of $\rho_{i,i+1}$ (see Figure 7).

We define B as the set of all such intersection points b_i, for $1 \leq i \leq n$. Thus, $|B| = n$.

If we treat each point b_i as a special disk, then this construction places b_{i-1}, D_i, and b_i in a degenerate configuration, since ℓ_i is tangent to all three. Note that our algorithm in Section 3 also works for such degenerate cases. Alternatively, if desired, one could perturb each point b_i infinitesimally toward $\rho_{i,i+1}$ so that the set $S = \mathcal{D} \cup B$ is in general position. In what follows, we retain the definition of B above and allow degeneracies, as this simplifies the exposition.

Since the convex hull $\mathcal{H}(\mathcal{D})$ can be computed in $O(n \log n)$ time [26], the set B can also be constructed in $O(n \log n)$ time.

The following lemma proves that B has the two desired properties.

▶ **Lemma 15.** *The set B has the two desired properties.*

Proof. See the full paper for the proof. ◀

For reference purpose, the following lemma summarizes our result in this section.

▶ **Lemma 16.** *Given a set $\mathcal{D}$ of n weighted disks in strongly convex position in the plane, a maximum-weight independent set of $\mathcal{D}$ can be computed in $O(n^3 \log n)$ time.*

5 The general case

In this section, we finally consider the general case in which $\mathcal{D}$ is not necessarily strongly convex (but is still convex). In particular, a disk may contribute more than one (maximal) arc to $\partial\mathcal{H}(\mathcal{D})$.

To solve the problem, we reduce it to the case in Section 4. To this end, we create a set Z of $O(n)$ *auxiliary points* as special disks with each disk assigned a weight equal to 1 such that the following two properties hold:

1. No point of Z is contained in any disk of $\mathcal{D}$.
2. $\mathcal{D} \cup Z$ is strongly convex.

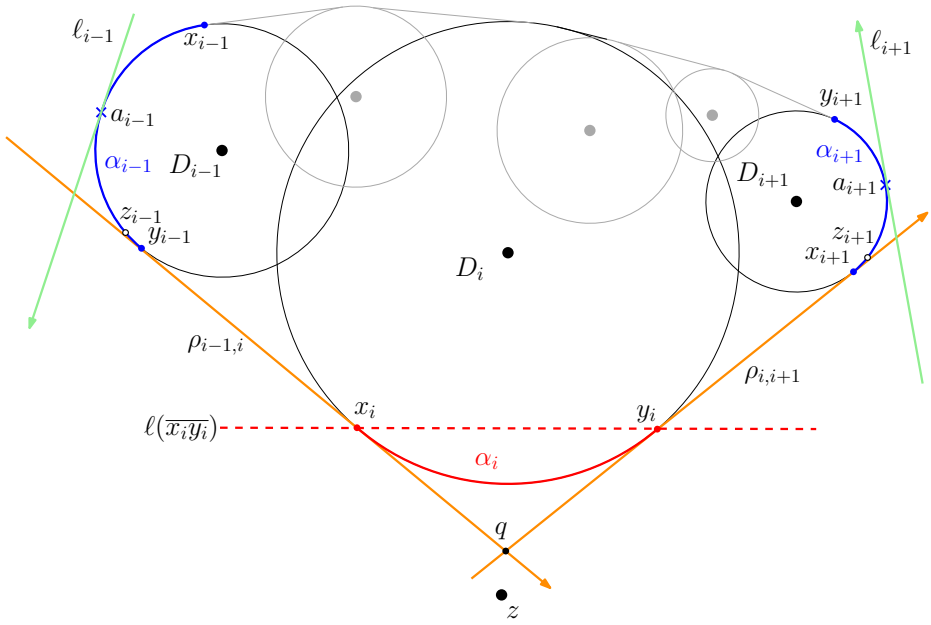


Figure 8 Illustration of the definition of z.

Let $S = \mathcal{D} \cup Z$. As in the reduction in Section 4, due to the first property, it suffices to compute a maximum-weight independent set in S. This is an instance of the problem in Section 4 due to the second property, and thus the problem can be solved in $O(n^3 \log n)$ time by Lemma 16.

In the rest of this section, we will focus on defining a set Z of $O(n)$ points satisfying the above two properties. We will also show that computing Z can be done in $O(n^2 \log n)$ time.

Consider a disk D_i that has more than one arc on $\partial\mathcal{H}(\mathcal{D})$. Note that at most one such arc can have a radian measure larger than or equal to π, and all other arcs have median measures strictly smaller than π. Let α_i be an arc of D_i on $\partial\mathcal{H}(\mathcal{D})$ whose radian measure is less than π.

Let α_{i-1} and α_{i+1} be the two disk arcs on $\mathcal{H}(\mathcal{D})$ adjacent to α_i clockwise and counterclockwise, respectively; let D_{i-1} and D_{i+1} be the disks where these arcs lie on, respectively (see Figure 8). Note that it is possible that D_{i-1} (resp., D_{i+1}) is a single point. We first assume that both D_{i-1} and D_{i+1} have radii larger than 0. We will discuss the other case later, which can be handled similarly. Refer to Figure 8 for an illustration of the notation introduced below.

For each $j \in \{i-1, i, i+1\}$, let x_j and y_j be the clockwise and counterclockwise endpoints of α_j, respectively. By definition, $\overline{y_{i-1}x_i}$ is an edge of $\mathcal{H}(\mathcal{D})$ and its supporting line is tangent to D_{i-1} at y_{i-1} and tangent to D_i at x_i. Let $\rho_{i-1,i}$ denote the supporting line of $\overline{y_{i-1}x_i}$. We orient $\rho_{i-1,i}$ so that $\mathcal{H}(\mathcal{D})$ is on the left side of $\rho_{i-1,i}$. Similarly, let $\rho_{i,i+1}$ be the oriented supporting line of $\overline{y_ix_{i+1}}$ so that $\mathcal{H}(\mathcal{D})$ is on its left side.

Let $\ell(\overline{x_iy_i})$ denote the line containing x_i and y_i. Without loss of generality, we assume that $\ell(\overline{x_iy_i})$ is horizontal such that x_i is left of y_i. Thus, the arc α_i is below $\ell(\overline{x_iy_i})$. Since the radian measure of α_i is less than π, $\rho_{i-1,i}$ must intersect $\rho_{i,i+1}$ at a point q strictly below $\ell(\overline{x_iy_i})$.

Let a_{i-1} be the middle point of the arc α_{i-1}. Let ℓ_{i-1} be an oriented line tangent to D_{i-1} at a_{i-1} such that $\mathcal{H}(\mathcal{D})$ is on the left of ℓ_{i-1}. Similarly, we define a_{i+1} and ℓ_{i+1} for D_{i+1}.

Note that by definition, q must be to the left of both ℓ_{i-1} and ℓ_{i+1}. Let q' be a point strictly below q but infinitesimally close to q. Then, q' is strictly to the right of both $\rho_{i-1,i}$ and $\rho_{i,i+1}$, and also strictly to the left of both ℓ_{i-1} and ℓ_{i+1}. This means that the region Q is not empty, where Q is the common intersection of the following four open halfplanes: the open halfplane to the right of $\rho_{i-1,i}$, the open halfplane to the right of $\rho_{i,i+1}$, the open halfplane to the left of ℓ_{i-1}, and the open halfplane to the left of ℓ_{i+1}.

We pick an arbitrary point z from Q and use it as an auxiliary point; we show below that z can be used to eliminate the arc α_i.

Let z_{i-1} be a point on ∂D_{i-1} such that $\overline{z_{i-1}z}$ is tangent to D_{i-1} at z_{i-1} and D_{i-1} is to the left of the directed line containing $\overline{z_{i-1}z}$ with the direction from z_{i-1} to z. Since z is strictly to the left of ℓ_{i-1} and strictly to the right of $\rho_{i-1,i}$, the tangent point z_{i-1} must be in the interior of the arc $\partial_{[a_{i-1},y_{i-1}]}D_{i-1}$, which is a subarc of α_{i-1}.

Symmetrically, let z_{i+1} be a point on ∂D_{i+1} such that $\overline{z_{i+1}z}$ is tangent to D_{i+1} at z_{i+1} and D_{i+1} is to the right of the directed line containing $\overline{z_{i+1}z}$ with the direction from z_{i+1} to z. Since z is strictly to the left of ℓ_{i+1} and strictly to the right of $\rho_{i,i+1}$, the tangent point z_{i+1} must be in the interior of the arc $\partial_{[x_{i+1},a_{i+1}]}D_{i+1}$, which is a subarc of α_{i+1}.

Consider the convex hull $\mathcal{H}(\mathcal{D}\cup\{z\})$. Its boundary can be obtained from $\partial\mathcal{H}(\mathcal{D})$ by replacing $\partial_{[z_{i-1},z_{i+1}]}\mathcal{H}(\mathcal{D})$ with $\overline{z_{i-1}z}\cup\overline{zz_{i+1}}$. In particular, no point of α_i appears on $\partial\mathcal{H}(\mathcal{D}\cup\{z\})$. Hence, every disk of $\mathcal{D}$ has the same number of arcs appearing in $\partial\mathcal{H}(\mathcal{D}\cup\{z\})$ as in $\partial\mathcal{H}(\mathcal{D})$ except that D_i has one less arc than before. Furthermore, since z is vertex of $\mathcal{H}(\mathcal{D}\cup\{z\})$, $\mathcal{D}\cup\{z\}$ is still convex.

The above assumes that both D_{i-1} and D_{i+1} are disks of radii larger than zero. The other case can be handled similarly. For example, if D_{i-1} is a point p_{i-1}, then following the same way as above, we have $\alpha_{i-1}=x_{i-1}=y_{i-1}=a_{i-1}=z_{i-1}=p_{i-1}$. Hence, p_{i-1} is still on $\partial\mathcal{H}(\mathcal{D}\cup\{z\})$. Therefore, it is still the case that $\mathcal{D}\cup\{z\}$ is convex and every disk of $\mathcal{D}$ has the same number of arcs appearing in $\partial\mathcal{H}(\mathcal{D}\cup\{z\})$ as in $\partial\mathcal{H}(\mathcal{D})$ except that D_i has one less arc than before. If D_{i+1} is a point, we can handle the situation similarly.

In summary, we can find a point z such that $\mathcal{D}\cup\{z\}$ is convex and every disk of $\mathcal{D}$ has the same number of arcs appearing in $\partial\mathcal{H}(\mathcal{D}\cup\{z\})$ as in $\partial\mathcal{H}(\mathcal{D})$ except that D_i has one less arc than before. As $\mathcal{H}(\mathcal{D})$ can be computed in $O(n\log n)$ time [26], such a point z can be found in $O(n\log n)$ time. We add z to Z. Note that z is outside every disk of $\mathcal{D}$.

If $\mathcal{D}\cup\{z\}$ is not strongly convex, then by following the same method we can find another auxiliary point with respect to $\mathcal{D}\cup\{z\}$ to eliminate another arc. Since $\partial\mathcal{H}(\mathcal{D})$ has $O(n)$ disk arcs [26], we can find a set Z of $O(n)$ auxiliary points such that $\mathcal{D}\cup Z$ is strongly convex (and no point of Z is inside any disk of $\mathcal{D}$). As computing each auxiliary point can be done in $O(n\log n)$ time, computing Z takes $O(n^2\log n)$ time in total (it may be possible to reduce the time to $O(n\log n)$ with a more careful implementation, but $O(n^2\log n)$ time suffices for our purpose).

The following theorem summarizes our result.

▶ **Theorem 17.** *Given a set $\mathcal{D}$ of n weighted disks in convex position in the plane, a maximum-weight independent set of $\mathcal{D}$ can be computed in $O(n^3\log n)$ time.*

The following corollary will be used in Section 6 to solve a dispersion problem.

▶ **Corollary 18.** *Given a set $\mathcal{D}$ of n weighted disks in convex position in the plane and a parameter $r>0$, one can compute in $O(n^3\log n)$ time a maximum-cardinality subset $\mathcal{D}'\subseteq\mathcal{D}$ such that the distance between any two disks in $\mathcal{D}'$ is greater than r.*

Proof. For each disk $D\in\mathcal{D}$, let $D(r)$ be the disk with the same center as D and with radius $r/2$ larger than that of D. For any subset $\mathcal{D}'\subseteq\mathcal{D}$, define $S(\mathcal{D}')=\{D(r):D\in\mathcal{D}'\}$. It is not difficult to see that for any subset $\mathcal{D}'\subseteq\mathcal{D}$, the distance between any two disks in $\mathcal{D}'$ is greater

than r if and only if $S(\mathcal{D}')$ is an independent set. Hence, the problem reduces to finding a maximum-cardinality independent set from $S(\mathcal{D})$. Since $\mathcal{D}$ is in convex position, $S(\mathcal{D})$ must also be in convex position because each disk of $\mathcal{D}$ grows by the same radius to obtain $S(\mathcal{D})$. Therefore, if we assign each disk of $S(\mathcal{D})$ a weight equal to 1, then we can compute a maximum-cardinality independent set of $S(\mathcal{D})$ in $O(n^3 \log n)$ time by Theorem 17. ◀

6 The dispersion problem

Let $\mathcal{D}$ be a set of n disks in convex position in the plane and let $k > 0$ be an integer. For any two disks $D, D' \in \mathcal{D}$, we define the *disk distance* $\text{dist}(D, D')$ as the minimum distance between any two points $q \in D$ and $q' \in D'$. The *dispersion problem* for $\mathcal{D}$ asks for a subset $\mathcal{D}' \subseteq \mathcal{D}$ of k disks maximizing the minimum pairwise distance $\text{dist}(D, D')$ of any two disks $D, D' \in \mathcal{D}'$.

Let r^* denote the optimal objective value. It is not difficult to see that r^* is equal to $\text{dist}(D, D')$ for two disks $D, D' \in \mathcal{D}$. Hence, r^* belongs to the set $R = \{\text{dist}(D, D') : D, D' \in \mathcal{D}\}$. Clearly, $|R| = O(n^2)$.

Given a number $r \geq 0$, the *decision problem* is to determine whether $r < r^*$, or equivalently, whether $\mathcal{D}$ has a subset of k disks whose minimum pairwise distance is greater than r. By Corollary 18, the decision problem can be solved in $O(n^3 \log n)$ time. To find r^*, we first compute R and then do binary search in R using the decision algorithm. This yields an $O(n^3 \log^2 n)$-time algorithm to compute r^*. We can also find an optimal subset by Corollary 18, as explained in the proof of the following theorem.

▶ **Theorem 19.** *Given a set of n disks in convex position in the plane and an integer k, one can find a subset of k disks maximizing their minimum pairwise distance in $O(n^3 \log^2 n)$ time.*

Proof. We first compute r^* as discussed above. Once r^* is computed, we apply the algorithm of Corollary 18 to produce an optimal subset with $r = r^*$. The algorithm of Corollary 18 will first compute a set $S(\mathcal{D})$ of disks and then apply the algorithm of Theorem 17 on $S(\mathcal{D})$. Here we need to slightly modify the algorithm of Theorem 17 by treating each disk of $S(\mathcal{D})$ as an open disk. For example, two disks are considered to be disjoint if they are outer tangent to each other. ◀

References

1 Pankaj K. Agarwal and Nabil H. Mustafa. Independent set of intersection graphs of convex objects in 2D. *Computational Geometry: Theory and Applications*, 34(2):83–95, 2006. doi:10.1016/j.comgeo.2005.12.001.

2 Pankaj K. Agarwal, Mark H. Overmars, and Micha Sharir. Computing maximally separated sets in the plane. *SIAM Journal on Computing*, 36(3):815–834, 2006. doi:10.1137/S0097539704446591.

3 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.

4 Alok Aggarwal, Leonidas J. Guibas, James Saxe, and Peter W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989. doi:10.1007/BF02187749.

5 Alberto Apostolico, Mikhail J. Atallah, and Susanne E. Hambrusch. New clique and independent set algorithms for circle graphs. *Discrete Applied Mathematics*, 36(1):1–24, 1992. doi:10.1016/0166-218X(92)90200-T.

6 Paul Balister, Béla Bollobás, Amites Sarkar, and Mark Walters. Connectivity of random k-nearest-neighbour graphs. *Advances in Applied Probability*, 37(1):1–24, 2005. doi:10.1239/aap/1113402397.

7 Prosenjit Bose, Paz Carmi, J Mark Keil, Anil Maheshwari, Saeed Mehrabi, Debajyoti Mondal, and Michiel Smid. Computing maximum independent set on outerstring graphs and their relatives. *Computational Geometry: Theory and Applications*, 103:101852, 2022. doi:10.1016/j.comgeo.2021.101852.

8 Onur Çağı rıcıand Bodhayan Roy. Maximum clique of disks in convex position. In *Abstracts of Computational Geometry: Young Researchers Forum (CG:YRF)*, pages 21:1–21:3, 2018. URL: https://www.computational-geometry.org/old/YRF/cgyrf2018.pdf.

9 Timothy M. Chan. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete and Computational Geometry*, 48:373–392, 2012. doi:10.1007/s00454-012-9417-5.

10 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, Micha Sharir, and Emo Welzl. Improved bounds on weak ϵ-nets for convex sets. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 495–504, 1993. doi:10.1145/167088.167222.

11 Jongmin Choi, Jaegun Lee, and Hee-Kap Ahn. Efficient k-center algorithms for planar points in convex position. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 262–274, 2023. doi:10.1007/978-3-031-38906-1_18.

12 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

13 Gautam K. Das, Guilherme D. da Fonseca, and Ramesh K. Jallu. Efficient independent set approximation in unit disk graphs. *Discrete Applied Mathematics*, 280:63–70, 2020. doi:10.1016/j.dam.2018.05.049.

14 Gautam K. Das, Minati De, Sudeshna Kolay, Subhas C. Nandy, and Susmita Sur-Kolay. Approximation algorithms for maximum independent set of a unit disk graph. *Information Processing Letters*, 115:439–446, 2015. doi:10.1016/j.ipl.2014.11.002.

15 Stefan Felsner, Rudolf Müller, and Lorenz Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74(1):13–32, 1997. doi:10.1016/S0166-218X(96)00013-3.

16 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972. doi:10.1137/0201013.

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

18 Luis H. Herrera and Pablo Pérez-Lantero. On the intersection graph of the disks with diameters the sides of a convex n-gon. *Applied Mathematics and Computation*, 411:126472, 2021. doi:10.1016/j.amc.2021.126472.

19 Clemens Huemer and Pablo Pérez-Lantero. The intersection graph of the disks with diameters the sides of a convex n-gon. *Discrete Mathematics*, 343(2):111669, 2020. doi:10.1016/j.disc.2019.111669.

20 Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

21 Andrzej Lingas. On approximation behavior and implementation of the greedy triangulation for convex planar point sets. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG)*, pages 72–79, 1986. doi:10.1145/10515.10523.

22 Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, Sekharipuram S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995. doi:10.1002/net.3230250205.

23 Tomomi Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings of the 2nd Japanese Conference on Discrete and Computational Geometry (JCDCG)*, pages 194–200, 1998. doi:10.1007/978-3-540-46515-7_16.

24 Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, pages 234–244, 1994. doi:10.1145/190809.190336.

25 Charles E. Perkins and Pravin Bhagwat. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, 1999. doi:10.1109/MCSA.1999.749281.

26 David Rappaport. A convex hull algorithm for discs, and applications. *Computational Geometry: Theory and Applications*, 1(3):171–187, 1992. doi:10.1016/0925-7721(92)90015-K.

27 Dana S. Richards and Jeffrey S. Salowe. A rectilinear steiner minimal tree algorithm for convex point sets. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 201–212, 1990. doi:10.1007/3-540-52846-6_90.

28 Paweł Rzążewski and Bartosz Walczak. Polynomial-time recognition and maximum independent set in burling graphs. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 445–460, 2026. doi:10.1007/978-3-032-11835-6_32.

29 Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing*, 14(2):448–468, 1985. doi:10.1137/0214034.

30 Vishwanath R. Singireddy, Manjanna Basappa, and Joseph S.B. Mitchell. Algorithms for k-dispersion for points in convex position in the plane. In *Proceedings of the 9th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, pages 59–70, 2023. doi:10.1007/978-3-031-25211-2_5.

31 Anastasiia Tkachenko and Haitao Wang. Computing dominating sets in disk graphs with centers in convex position. In *Proceedings of the 17th Latin American Theoretical Informatics Symposium (LATIN)*, page to appear, 2025. doi:10.48550/arXiv.2601.22609.

32 Anastasiia Tkachenko and Haitao Wang. Computing maximum cliques in unit disk graphs. In *Proceedings of the 37th Canadian Conference on Computational Geometry (CCCG)*, pages 283–291, 2025. URL: https://cccg-wads-2025.eecs.yorku.ca/cccg-papers/6B2.pdf.

33 Anastasiia Tkachenko and Haitao Wang. Dominating set, independent set, discrete k-center, dispersion, and related problems for planar points in convex position. In *Proceedings of the 42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 73:1–73:20, 2025. doi:10.4230/LIPIcs.STACS.2025.73.

34 Da-Wei Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information processing letters*, 28(6):281–286, 1988. doi:10.1016/0020-0190(88)90174-3.